TECHNISCHE UNIVERSITÄT MÜNCHEN

TUM SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY

Thread-Modular Abstract Interpretation: The Local Perspective

Michael Benedikt Schwarz

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz:
Prof. Dr. Thomas Neumann

Prüfende der Dissertation:
1. Prof. Dr. Helmut Seidl
2. Prof. Dr. Antoine Miné
3. Assoc. Prof. Dr. Pietro Ferrara

Die Dissertation wurde am 19.12.2024 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 28.04.2025 angenommen.

Acknowledgments

This thesis would not have been possible without the support of many people — to all of whom I am deeply grateful.

First, I want to thank my advisor Prof. Dr. Helmut Seidl for his continued support and guidance throughout the work that eventually became this thesis. To all my colleagues here at TUM and at UTartu, thank you for the countless laughs shared at many *Gobcon*s and lunches. Special thanks go to all my coauthors, without whom much of the work done during my past 5 years at TUM would not have been possible. Special thanks also go to Prof. Dr. Vesal Vojdani for agreeing to proof-read an early draft of the section on local traces and for the many helpful comments and suggestions.

Thank you to the external members of my committee, Prof. Dr. Antoine Miné and Prof. Dr. Pietro Ferrara for agreeing to review this thesis!

Last but certainly not least, thank you to my friends & family — for everything!

On a perhaps slightly less personal note, I gratefully acknowledge that this work was supported in part by Deutsche Forschungsgemeinschaft (DFG) — 378803395/2428 ConVeY.

Abstract

Multi-threaded software is notoriously hard to write, rendering static analysis techniques that scale to real-world multi-threaded programs while remaining sufficiently precise all the more sought-after. We propose *thread-modular* analyses of the values of global variables in multi-threaded programs, which we obtain as abstract interpretations of the *local* trace semantics — a new thread-modular concrete semantics focussing on the *local perspectives* of threads. We cast two existing non-relational analyses into our framework, enabling a principled comparison between them — and spurring the development of three novel, more precise, non-relational analyses. Regarding relational analyses, we propose new analyses that once more take a *local perspective* in that they track, for each mutex, relationships within clusters of globals only written when the respective mutex is held. Surprisingly, we find that larger clusters can lead to less precise information and thus propose to track all subclusters. For some weakly relational domains, e.g., the widely-used *octagon* domain, we find that — thanks to a property we dub *2-decomposability* — the most precise result is already attained when considering subclusters of size at most 2. To seamlessly increase the precision of any local trace–based analysis, we propose to track further abstractions of the computational history, which we call *digests*, that can be used to exclude spurious interactions between threads and refine the analysis. We instantiate this scheme by giving digests computing thread *id*s, enabling the analyses to take may-happen-in-parallel information into account when determining read values of globals and evaluate all analyses — with and without refinement — on a set of real-world benchmarks, both w.r.t. runtime and precision.

Zusammenfassung

Die Entwicklung nebenläufiger Programme stellt bekanntermaßen eine große Herausforderung dar, weshalb statische Analysetechniken, die auch auf Programme realistischer Größe anwendbar sind und dabei hinreichend präzise bleiben, umso gefragter sind. Wir entwickeln thread-modulare Analysen der Werte globaler Variablen in nebenläufigen Programmen und stützen diese Analysen auf die Abstrakte Interpretation der Local Trace Semantik — eine thread-modulare konkrete Semantik, die den Fokus auf die lokalen Perspektiven der Threads legt. Wir drücken zwei bestehende nicht-relationale Analysen in unserem Framework aus, was einen prinzipiellen Vergleich zwischen ihnen ermöglicht — und die Entwicklung dreier neuer, präziseren, nicht-relationalen Analysen anregt. Was relationale Analysen betrifft, so schlagen wir neue Analysen vor, die wiederum eine lokale Perspektive einnehmen, insofern als dass sie für jede Mutex Beziehungen innerhalb von Clustern von globalen Variablen verwalten, welche nur geschrieben werden, wenn die entsprechende Mutex gehalten wird. Überraschenderweise stellen wir fest, dass größere Cluster zu weniger präzisen Informationen führen können, und schlagen daher vor, Informationen über alle Subcluster zu verwalten. Für einige schwach relationale Domänen, wie z. B. die weit verbreitete *Octagon*-Domäne, stellen wir fest, dass — dank einer Eigenschaft, die wir als *2-Decomposability* bezeichnen — das genaueste Ergebnis bereits bei der Betrachtung von Subclustern mit einer Größe von höchstens 2 erzielt wird. Zur nahtlosen Erhöhung der Präzision jeder auf Local Traces basierenden Analyse schlagen wir vor, weitere Abstraktionen der Berechnungshistorie zu verwalten, welche wir als *Digests* bezeichnen und die verwendet werden können, um unerwünschte Interaktionen zwischen Threads auszuschließen und somit die Analyse zu verfeinern. Wir geben als Beispiel Digests für die Berechnung von Thread *Id*s an, mit deren Hilfe die Analysen may-happen-in-parallel Informationen bei der Bestimmung der für globale Variablen gelesenen Werte berücksichtigen können. Schließlich vergleichen wir sowohl die Präzision als auch die Laufzeit aller Analysen — mit und ohne Verfeinerung — auf einer Reihe von realen Benchmarks.

List of Original Publications

This thesis encompasses the content of the following two original publications and expands on them with previously unpublished material.

- Schwarz, M., Saan, S., Seidl, H., Apinis, K., Erhard, J., Vojdani, V.: Improving thread-modular abstract interpretation. In: Dragoi, C., Mukherjee, S., Namjoshi, K.S. (eds.) Static Analysis - 28th International Symposium, SAS 2021, Chicago, IL, USA, October 17-19, 2021, Proceedings, Lecture Notes in Computer Science, vol. 12913, pp. 359–383, Springer (2021)
- Schwarz, M., Saan, S., Seidl, H., Erhard, J., Vojdani, V.: Clustered relational thread-modular abstract interpretation with local traces. In: Wies, T. (ed.) Programming Languages and Systems - 32nd European Symposium on Programming, ESOP 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings, Lecture Notes in Computer Science, vol. 13990, pp. 28–58, Springer (2023)

On top of the support from Deutsche Forschungsgemeinschaft (DFG), the Estonian co-authors of both publications were supported in part by the Estonian Research Council grant PSG61, and the Estonian Centre of Excellence in IT (EXCITE), funded by the European Regional Development Fund.

Conversely, some material first proposed in this thesis, particularly on the topic of digests, has been included into a publication that appeared after the thesis submission:

- Schwarz, M., Erhard, J.: The digest framework: concurrency-sensitivity for abstract interpretation. Int. J. Softw. Tools Technol. Transf. **26**(6), 727–746 (2024), doi: 10.1007/S10009-024-00773-Y

Contents

1 Introduction

Just a few years ago, the statement that software truly permeates every aspect of our lives would have required a well-crafted argument and might even have sparked some pushback from those less in tune with digital technology. Now, that many of us carry around often even several small computers in our pockets and on our wrists wherever we go, and it is hard to find a household device of which no WiFi-enabled version and corresponding custom app exists, this statement has become so widely accepted it almost borders on a trope. Nevertheless, this fact has wide-reaching implications: While a smartphone rebooting while the user is attempting to watch the latest TikTok[1] videos, a smart dishwasher running the regular program instead of the super-energy-saver option selected on the app by its user, or a navigation program crashing before displaying the fastest way to work given current traffic conditions, may all be best described as minor inconveniences, similar problems in slightly different settings can have much more dire consequences. Replace the user watching short-form video content with a trader about to input a time-critical order, the smart dishwasher with a smart medical device controlling an IV, and place the navigation program inside an autonomous vehicle — now the consequences of software going wrong range from significant economic losses to injuries and potentially even the loss of life.

It is precisely in settings where the consequences of missing a bug in a program can be dire, where techniques that go beyond extensive testing and best practices such as code audits and reviews, have established a strong foothold. One such technique that has enjoyed tremendous success, e.g., in the domain of control software for airplanes, is abstract interpretation as introduced by Cousot and Cousot [30]. Unlike other techniques such as symbolic execution or model checking, it can be used to *prove* the *absence* of bugs in programs, and it is possible to design analyses that always terminate. While this, at first glance, seems to be in conflict with Rice's theorem stating that all non-trivial properties about programs are undecidable, it is not. The problem of undecidability is side-stepped by *overapproximating* the concrete semantics of programs. If such an overapproximation of all executions does not violate the property of interest, neither will any concrete execution. Therefore, if the abstract interpreter claims the program to be safe, it is safe (*no false negatives*). The converse is not true though: due to the overapproximation, an abstract interpretation may flag potential bugs that do not exist in the concrete (*false positives*). The perhaps most wide-spread use of abstract interpretation is for showing the *absence* of certain kinds of bugs (often called *runtime errors*), ranging from integer overflows to out-of-bounds accesses to arrays or null-pointer dereferences.

[1] Substitute this for whatever app is *en vogue* now if you're reading this later than 2025.

Since their advent several decades ago, multi-core processors must now be considered more or less standard in many settings, whether in servers, mobile phones, high-end embedded devices, or desktop computers.[2] To harness the additional computational power, also more and more software became multi-threaded — naturally leading to a search for techniques for certifying that *multi-threaded* software is free of certain classes of bugs. It was observed early on that naively applying the analysis techniques employed for the abstract interpretation of single-threaded programs is not the method of choice: The sheer multitude of possible interleavings, even when considering relatively modest-sized programs, leads to an explosion of control states to distinguish. Instead, the concept of *thread-modular* analysis was proposed, where one tries to analyze each thread in isolation as much as possible, overapproximating the effects of other threads, while hopefully remaining precise enough to establish properties of interest.

One particular style of writing concurrent software is called *pthread-style* (after the Pthreads library), where threads are explicitly *created* and *joined*, and, among other idioms, locks referred to as *mutexes* are used to ensure mutual exclusion for shared resources such as global variables or the heap. In this thesis, we work on thread-modular abstract interpretation for such programs. Our analyses aim to find information about the values of global variables. On top of being relevant in its own right to, e.g., establish assertions, good information about the values of global variables is indispensable for establishing a wide range of other properties, ranging from establishing that no out-of-bounds accesses to arrays happen to properties such as the absence of data races, e.g., when control-flow depends on the values of global variables.

A common theme throughout this thesis is consequently taking a *local perspective* whenever possible: Instead of considering a concrete semantics where some global observer exists that knows about all actions of all threads, we take a *local* perspective of the concrete semantics where each thread is only aware of the things in its own past, and those parts of the past of other threads that it has learned about by communication. Whenever threads communicate, the local perspectives of the threads involved in the communication are combined to yield a new *local* perspective. Many approaches to the analysis of the values of globals have been proposed, ranging from rather simple to quite sophisticated. We first cast two well-known approaches into a common framework, which allows us to perform a principled comparison between them. Building on this, we further propose novel analyses improving on either style, as well as a general mechanism to improve the precision of analyses by taking further abstractions of the *local* computational past into account. In this way, we obtain novel analyses for which we give principled soundness proofs. As tracking relationships between program variables has been shown to be indispensable for proving some properties of programs, we also propose novel thread-modular relational analyses. Here, we take the *local* perspective in that we do not track arbitrary relationships between globals but instead focus only on those that are mediated by common protecting mutexes, which allows

[2] For instance, according to the *Steam Hardware & Software Survey: November 2024* [132] fewer than 1% of machines accessing the gaming platform had only 1 core, with the majority having 4 or more cores.

us to associate these invariants with the mutexes they are mediated by. We provide an experimental evaluation comparing both runtimes and precisions of all analyses. Having implementations in a common framework ensures these comparisons are about the essences of the analyses and independent of incidental aspects, such as the approach to inter-procedural analysis or the implementations of domains, one is faced with when making a more global comparison between implementations in different frameworks.

1.1 A Primer on Complete Lattices and Scott-continuity

We have deliberately decided to not provide a lengthy introduction to abstract interpretation in this thesis. We will instead assume readers to have some level of familiarity with abstract interpretation as would, e.g., be attained after perusing the first few chapters of an introductory textbook [86, 102, 115]. A comprehensive overview of most of the developments in the field, on the other hand, is provided by Cousot [29] in his abstract interpretation textbook. In this thesis, we will instead briefly recall key notions in the places where we use them, and provide definitions in so far as they are referenced in later parts of this work.

In this spirit, this section serves as a quick primer on complete lattices, and recalls the definition of Scott-continuity and some fixpoint theorems, in so far as we rely on them later. Readers familiar with these may want to skip this section and jump to the section on the (perhaps less standard) notion of side-effecting constraint systems right away.

Definition 1 (Complete Lattice)**.** *A set S equipped with a partial order $\sqsubseteq$ where any subset $S' \subseteq S$ has a* least *upper bound (w.r.t. $\sqsubseteq$) denoted by $\bigsqcup S'$ is called a complete lattice $(S, \sqsubseteq)$.*

It is easily verified that a complete lattice also has a greatest (w.r.t. $\sqsubseteq$) lower bound for any subset $S' \subseteq S$ denoted by $\bigsqcap S'$. By convention, the least upper bound of the entire set is denoted by $\top = \bigsqcup S$ whereas the least upper bound of the empty set is denoted $\bot = \bigsqcup \emptyset$ and corresponds to the greatest lower bound of S. We obtain that $\forall s \in S : \bot \sqsubseteq s \sqsubseteq \top$. We denote by $a \sqcup b$ the result of $\bigsqcup\{a, b\}$, and similarly for $\sqcap$.

Example 1. *The powerset 2^U of some set U forms a complete lattice with $\sqsubseteq = \subseteq$, $\sqcup = \cup$, $\sqcap = \cap$, $\top = U$ and $\bot = \emptyset$.* □

Theorem 1 (Knaster-Tarski Fixpoint Theorem)**.** *For a complete lattice $(S, \sqsubseteq)$ and a monotonic function $f : S \to S$, the set of all fixpoints of f forms a complete lattice w.r.t. $\sqsubseteq$.*

As a corollary, one obtains the existence of a *least* fixpoint for any monotonic f. However, in the later proofs, we will require a stronger property, namely that the Kleene fixpoint theorem is applicable, and thus the least solution does not only exist but is given by the least-upper bound of the Kleene iterates. For this theorem to apply, a stronger property than monotonicity is required, namely that the function be *Scott-continuous* [54]. We recap some basic definitions and propositions for *Scott-continuous* functions, in so far as they are of relevance to this thesis.

Definition 2 (Directed Subset)**.** *A directed subset of some complete lattice is a non-empty set of lattice elements, such that each pair of elements has an upper bound in the set.*

Definition 3 (Scott-Continuity)**.** *A function $f : S \to T$ from one complete lattice into another complete lattice is called Scott-continuous, if, for all directed subsets $\mathcal{D}$,*

$$f\left(\bigsqcup_{D\in\mathcal{D}} D\right) = \bigsqcup_{D\in\mathcal{D}} f\,D$$

As Scott-continuity implies monotonicity, by the Knaster-Tarski fixpoint theorem, a least fixpoint exists for each Scott-continuous function.

Theorem 2 (Kleene Fixpoint Theorem)**.** *For a Scott-continuous function f the least fixpoint is the least-upper-bound of the ascending Kleene chain of f given by*

$$\bot \sqsubseteq f\,\bot \sqsubseteq f\,f\,\bot \sqsubseteq f^n\,\bot \sqsubseteq \dots$$

Lastly, we list some propositions about composition of Scott-continuous functions that will be needed in later proofs.

Proposition 1. *Consider a function $f : S \times S \to S$ and a complete lattice S. The following two statements are equivalent:*

(1) f is Scott-continuous on $S \times S$.

(2) f is Scott-continuous in each argument, i.e., for all elements $a \in S$, both functions $f_{a_0}\,x = f(a,x) : S \to S$ and $f_{a_1}\,x = f(x,a) : S \to S$ are Scott-continuous.

Proof. $(1) \Rightarrow (2)$ obviously holds, for $(2) \Rightarrow (1)$ see Lemma 2.9 in Gierz et al. [54]. □

Proposition 2. *The composition $g \circ f$ of two Scott-continuous functions $f : S \to T$ and $g : T \to U$ is itself Scott-continuous.*

Proof. For $\mathcal{D}$ a directed subset, by Scott-continuity of f

$$(g \circ f)(\sqcup_{D\in\mathcal{D}} D) = g(f(\sqcup_{D\in\mathcal{D}} D)) = g(\sqcup_{D\in\mathcal{D}} f(D))$$

Next, we show that $\mathcal{D}' = \{f\,D \mid D \in \mathcal{D}\}$ is a directed subset in T. Consider two elements $f\,D_1$ and $f\,D_2$ in $\mathcal{D}'$. Then $f\,D_1 \sqcup f\,D_2 = f(D_1 \sqcup D_2)$ (by Scott-continuity of f), and as $\mathcal{D}$ is directed, we have $D_1 \sqcup D_2 \in \mathcal{D}$, and thus $(f\,D_1) \sqcup (f\,D_2) \in \mathcal{D}'$ and $\mathcal{D}'$ is a directed subset. Then, as g is Scott-continuous, we get

$$(g \circ f)(\sqcup_{D\in\mathcal{D}} D) = g(\sqcup_{D\in\mathcal{D}} f(D)) = \sqcup_{D\in\mathcal{D}} g(f(D)) = \sqcup_{D\in\mathcal{D}} (g \circ f)(D)$$

□

Proposition 3. *The set of Scott-continuous functions forms a complete lattice. Thus, given a set F of Scott-continuous functions, the function $\bigsqcup_{f\in F} f$ is also Scott-continuous.*

Proof. See Lemma 2.4 in Gierz et al. [54]. □

We will use complete lattices in both the concrete and abstract semantics, and use the results about Scott-continuous functions in Chapter 6 when providing soundness proofs of our analyses.

1.2 Side-Effecting Constraint Systems

Side-effecting systems of constraints, as first proposed by Seidl et al. [113] and later studied in detail by Apinis et al. [9], are a powerful framework for describing abstract interpretation-based analyses. Among other things, they allow conveniently expressing varying degrees of context sensitivity for interprocedural analysis [9, 43] and elegantly handling non-local control flow [106]. In the context of this thesis, their most salient feature is that they allow for conveniently formalizing the accumulation of flow- (as well as context-) insensitive information during an analysis that is, in principle, flow- (as well as context-) sensitive [113, 133, 134].

Assume that X is a set of *unknowns* where for each $x \in \mathsf{X}$, $\mathcal{D}_x$ is a complete lattice of possible (abstract or concrete) values of x. Let $\mathcal{D}$ denote the disjoint union of all sets $\mathcal{D}_x$. Let $\mathsf{X} \Rightarrow \mathcal{D}$ denote the subset of all type-correct mappings of the set of mappings $\mathsf{X} \to \mathcal{D}$, i.e., those η where $\eta\, x \in \mathcal{D}_x$. The ordering on the individual $\mathcal{D}_x$ is lifted point-wise to $\mathsf{X} \Rightarrow \mathcal{D}$. Technically, a (side-effecting) constraint takes the form $x \sqsupseteq f_x$ where $x \in \mathsf{X}$ is the left-hand side and the right-hand side

$$f_x : (\mathsf{X} \Rightarrow \mathcal{D}) \to ((\mathsf{X} \Rightarrow \mathcal{D}) \times \mathcal{D}_x)$$

takes a mapping $\eta : \mathsf{X} \Rightarrow \mathcal{D}$, while returning a collection of side-effects to other unknowns in X together with the contribution to the left-hand side. We remark that there may be zero, one, or several constraints and thus right-hand sides for any given unknown (left-hand side). While we in principle assume all mappings to be total, we usually represent them by giving only those bindings that are non-$\bot$, and will later talk about a constraint causing side-effects to some unknowns to mean those unknowns that appear with a non-$\bot$ contribution in the first component.

Let $\mathcal{C}$ denote a set of such constraints. A mapping $\eta : \mathsf{X} \Rightarrow \mathcal{D}$ is called *solution* of $\mathcal{C}$ if for all constraints $x \sqsupseteq f_x$ of $\mathcal{C}$, it holds for $(\eta', d) = f_x\, \eta$ that $\eta \sqsupseteq \eta'$ and $\eta\, x \sqsupseteq d$; that is, all side-effects of the right-hand side and its contribution to the left-hand side are accounted for by η. Assuming that all right-hand side functions are monotonic, the system $\mathcal{C}$ is known to have a *least* solution by the Knaster-Tarski fixpoint theorem.

Example 2. *Consider $X = \{w, x, y, z\}$, where $\mathcal{D}_w = \mathcal{D}_x = \mathcal{D}_y = \mathcal{D}_z$ is the powerset lattice $2^{\{a,b,c\}}$ ordered by $\subseteq$ and consider the following constraints and right-hand sides:*

$$\begin{array}{lcl@{\qquad}lcl} [x] & \sqsupseteq & f_x & f_x\,\eta & = & (\emptyset, \{a\}) \\ [x] & \sqsupseteq & f'_x & f'_x\,\eta & = & (\emptyset, \{c\} \cap \eta\,[z]) \\ [y] & \sqsupseteq & f_y & f_y\,\eta & = & (\{[z] \mapsto \eta\,[y]\}, \{b\} \sqcup \eta\,[x]) \\ [z] & \sqsupseteq & f_z & f_z\,\eta & = & (\emptyset, \bot) \end{array}$$

While f_x, f'_x, and f_z do not cause any side-effects, f_y causes a side-effect to the unknown $[z]$. Consider $\eta_1 = \{[w] \mapsto \top, [x] \mapsto \top, [y] \mapsto \top, [z] \mapsto \top\}$. While η_1 is a solution of the side-effecting constraint system, it is not the least *solution: The least solution is given by $\{[w] \mapsto \bot, [x] \mapsto \{a\}, [y] \mapsto \{a, b\}, [z] \mapsto \{a, b\}\}$. We remark that in the least solution the value for the unknown $[w]$ that does not appear on the left-hand side of any constraint and does not receive any side-effects is $\bot$.* □

Note that we enclose unknowns into $[\cdot]$ to clarify that we are referring to *unknowns* as the set of unknowns often coincides with some other set: For example, in later chapters, for a program point u, we refer to the corresponding unknown by $[u]$.

We remark that in the side-effecting constraint systems considered in the rest of this thesis, an unknown x that receives a value by side-effect will not have a right-hand side, and thus receive all of its values only by side-effect.

To compute (least) solutions of side-effecting constraint systems, one can rely on fixpoint solvers. Several families of solvers have either been adapted to work with side-effects (e.g., the family of Top-Down solvers [114, 131]) or have been developed with side-effecting constraint systems in mind in the first place — as is the case for the SLR family of solvers [7]. Many of these solvers are implemented in the static analysis framework Goblint, which we will also use for our experiments.

1.3 Outline

The rest of this thesis is structured as follows: Aside from providing necessary definitions, e.g., of the programming language considered and the program representation, Chapter 2 introduces the concept of local traces, which elegantly captures the semantics of concurrent programs. These local traces serve as the reference semantics throughout the rest of the thesis. Chapter 2 also introduces a generic framework to refine the constraint systems according to abstractions of reaching local traces (*digests*). Chapter 3 characterizes the abstract domains used throughout later parts of this thesis, and introduces the notion of *2-decomposability*. Next, Chapter 4 details novel non-relational (Section 4.1) and relational (Section 4.2) static analyses of the values of global variables. Chapter 5 provides details on the implementation of these analyses in the static analyzer Goblint, listing all the additional features covered by the implementation when compared to the version presented in the previous chapter, and provides a comprehensive evaluation both in terms of runtimes and precision of the analyses. Chapter 6 provides principled soundness proofs for all novel analyses presented in this work. Lastly, Chapter 7 puts this thesis in the larger context of existing work in the field, and Chapter 8 outlines some promising directions for future work and concludes.

We remark that, at the back of the thesis, there is a list of all symbols along with a short definition that readers may find helpful to refer to when trying to recall the precise meaning of some symbol.

2 A Local Trace Semantics

The perhaps classical semantics to describe the behavior of multi-threaded programs is some flavor of *interleaving semantics* — at least for the sequentially consistent case [75], with which this thesis is also concerned. It considers all ways in which the actions taken by different threads may interleave. There, states may be given by maps from thread *id*s to local configurations of the corresponding threads and another component containing information on resources shared between threads [42]. A trace may then consist of a sequence of such states interleaved with pairs denoting which thread took what action. In this way, all the actions appearing in such a trace are totally ordered w.r.t. each other. This is the case even for actions taken by different threads that are independent of each other, e.g., actions only working on local variables of individual threads.

The key idea behind the local trace semantics, then, is to avoid detailed reasoning based on interleavings and instead, describe the program behavior based on the *local perspectives* of the threads of the program: Each thread should only know about the operations it has performed itself and those parts of the history of other threads that it has learned about by communicating with them (via having executed a pair of *observing* and *observable* actions). When executing such actions, the perspectives of different threads are combined — provided they are compatible — to yield a new perspective.

Example 3. *Consider for illustration, e.g., the program in Fig. 2.1 and the control-flow graphs of its thread templates given in Fig. 2.2. A graphical representation of some of its local traces is given in Fig. 2.3. Each local trace records for each thread the sequence of configurations it has reached, as well as additional dependencies between these configurations. In this example, one of the types of additional dependencies is* $\rightarrow_c$ *which is the* create order *and goes from the configuration preceding a thread create action to the first configuration of the newly created thread, as seen, e.g., in the trace (c) where it connects the configuration of the main thread before the call to* create *to the first configuration of thread* t_1*. The other type of additional dependency is* $\rightarrow_{m_g}$*, the* mutex order *for the mutex* m_g *that relates lock and unlock operations of this mutex with each other. An edge* $\rightarrow_{m_g}$ *from (the configuration following)* initMT *to a lock operation as seen in three of the local traces denotes that this lock is the first lock of the mutex in this execution. In the figure, the last configuration of the* ego *thread in each of the local traces is highlighted. As remarked before, each thread only knows its own history and the part of the history of other threads it has learned about by communicating. For example, in the local traces (b) and (d), the main thread does not know whether the other thread has executed its local action* $y = 1$ *yet. It only learns that this has happened through communication with the other thread — in this example by acquiring the mutex* m_g *— at which point that action by the other becomes part of the local perspective of the main thread, as shown in local trace (e).* □

```
main:                          t1:
    initMT;                        y = 1;
    x = create(t1);                lock(m_g);
    lock(m_g);                     g = 2;
    g = 1;                         unlock(m_g);
    unlock(m_g);                   return;
    return;
```

Figure 2.1: A simple toy program with local variables x and y and global variable g.

The rest of this chapter is structured as follows: Section 2.1 provides some key definitions, while Section 2.2 introduces the local trace semantics and gives two fixpoint characterizations (one global and one local) that are then shown to be equivalent. Section 2.3 introduces *digests* which are abstractions of the history used to split unknowns and shows a constraint system refined with digests to be equivalent to the original system. Section 2.4 then describes the set of actions used for our core subset of a C-like imperative language, whereas Section 2.5 describes the concrete formalism for local traces we employ, and Section 2.6 gives some examples of local traces using this formalism. Section 2.7 then gives some example digests that are tailored to the set of actions considered here and are abstractions of the locking history. Section 2.8 proposes a digest that amounts to computing thread *id*s for dynamically created threads. This digest is of a particular form where it can be computed based on the current thread and its creation history only, making it applicable in a wider setting. Section 2.9 finally characterizes this notion of *ego-lane digest*.

When compared to the description of the local trace semantics in our earlier work [107] we consider additional actions here, provide more detailed proofs about the different formulations of local traces and also provide the notion of digests for refining analyses already at the level of the concrete semantics and give refined analyses directly, instead of describing refinement as a wrapper around analyses. Also, the notion of *ego-lane* digests is original to this thesis.

2.1 Definitions

While we will later consider a core subset of an imperative, C-like, programming language, for now, we remain generic in the specific actions supported by the programming language and only fix some essential things such as the distinction between local and global variables. In this way, proofs can be conducted at a higher level of abstraction and the obtained results do not just apply for one particular instantiation of the local trace semantics with actions, but for *many* such instantiations.

Let us consider two finite, disjoint sets $\mathcal{X}$ and $\mathcal{G}$ of local and global variables, respectively. Let $\mathcal{V}ars = \mathcal{X} \cup \mathcal{G}$ denote the union of these sets, i.e., all variables appearing in the program. The understanding here is that each thread has its own copy of local

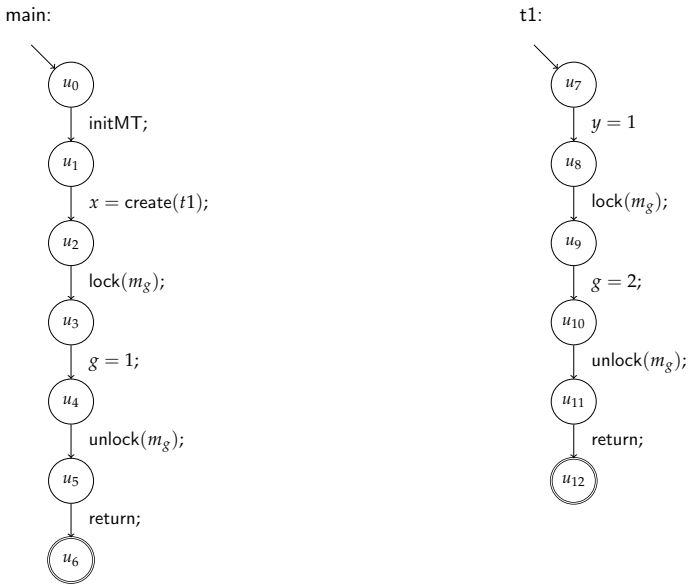


Figure 2.2: CFGs for the thread templates from Fig. 2.1.

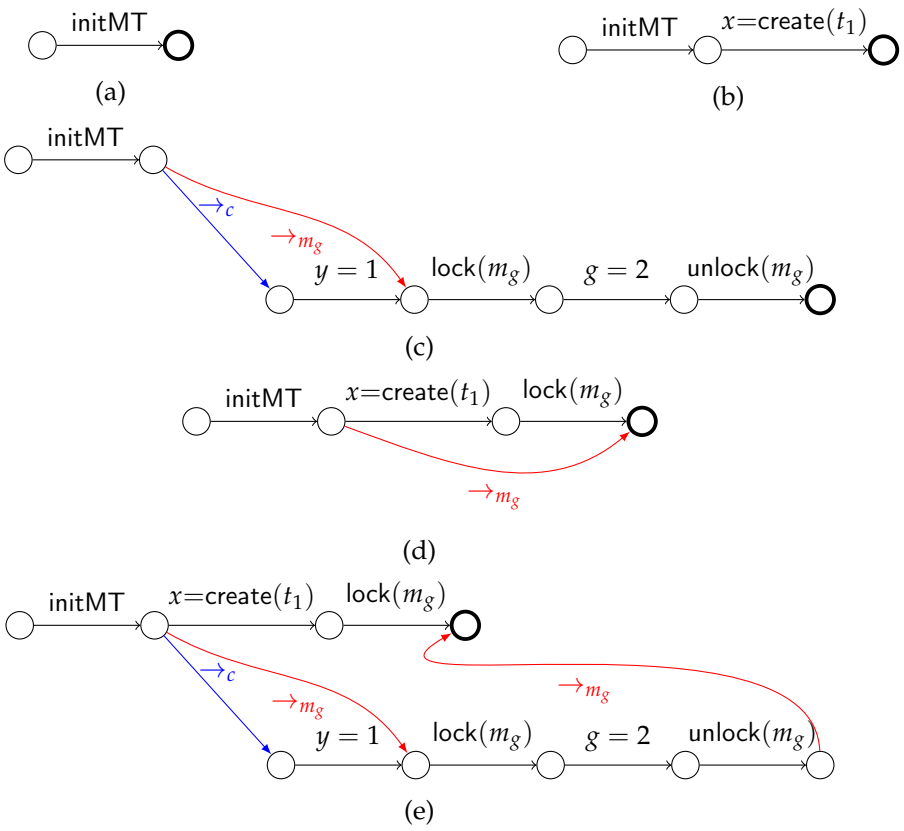


Figure 2.3: Some local traces of the program from Fig. 2.1.

variables, whereas global variables are shared between all threads. Variables may be of built-in type to compute with, i.e., of type **int**,[1] or have type thread *id*. We denote the set of values of type **int** by $\mathcal{V}_{\mathbf{int}}$, the set of values of type thread *id* by $\mathcal{V}_{\mathbf{tid}}$, and their disjoint union by $\mathcal{V}$. The values of type thread *id* can be copied between variables of appropriate type, be compared with other thread *id*s for equality, or be passed to actions designated as taking thread *id*s. We implicitly assume that all programs are well-typed, i.e., a variable either always holds thread *id*s or **int** values. We assume that there is one particular local variable $\mathsf{self} \in \mathcal{X}$ holding the thread *id* of the current thread. This variable is assigned to at the start of the main thread and when starting a new thread. We assume these assignments happen implicitly, and omit them from the programs. At the start of program execution, we assume that global variables have the value 0 for global variables of type **int**, and the value $0_{\mathbf{tid}}$ for global variables of type thread *id*. For convenience, we will use 0 to denote both initial values.

Local variables, on the other hand, may initially have any value. A *local program state* thus is a (type-correct) mapping $\sigma : \mathcal{X} \Rightarrow \mathcal{V}$ where $\sigma\,\mathsf{self} \in \mathcal{V}_{\mathbf{tid}}$. Let Σ denote the set of all local program states.

Let $\mathcal{A}ct$ and $\mathcal{N}$ denote some set of actions and the set of program points respectively. Each of the finitely many *thread templates* (from which threads may be dynamically created once, several times, or never) is then represented by

- some (finite) control-flow graph (CFG) where each edge $e \in \mathcal{E}$ is of the form (u, act, u') for program points u and u' and an action act, where there is at most one edge between any two program points u and u'; as well as
- a program point within the control-flow graph where execution is meant to start. We require that a path from the node corresponding to this program point to all other program points in the CFG exists.[2]

W.l.o.g., we assume the control-flow graphs belonging to different thread templates to be disjoint.[3] We can, therefore, identify a thread template with the program point where its execution starts, and do so in the following. We assume there is a special thread template main starting at program point u_0, which is where we assume execution starts at the beginning of the program. Furthermore, we assume that only one instance of this thread is created at runtime and that the first action taken by this thread is always some special action $\mathsf{initMT} \in \mathcal{A}ct$ that is used to initialize the multi-threaded environment and appears in each program exactly once. Lastly, we assume that the program point u_0 has no incoming edges, i.e., initMT is statically known to be executed at most once. We often give examples as programs from a C-like language, employing control structures

[1] For simplicity, we consider programs operating on mathematical integers here and thus do not concern ourselves with issues such as overflows.

[2] Note that this does not imply that every program point is reachable in program executions, but only that there are no nodes in the CFG that are not connected to the start point.

[3] As there are only finitely many thread *templates*, whenever this is not the case, we may duplicate shared parts of the control-flow graphs to obtain disjointness.

such as `if` and `while` and giving the names of program points — where required — using the syntax C-like languages use for labels. The translation to a control-flow graph is straightforward for all of our examples and thus not explicitly given.

2.2 Local Traces

As hinted at in the introduction, we identify some actions as *observable* by other threads when executing corresponding *observing* actions (see Table 2.1). The intuition is that a thread learns about operations performed by the thread executing an *observable* action, when it executes a corresponding *observing* action. We partition the set of actions into

- those that create a new thread (*creating*),
- those that are *observable*,
- those that are *observing*,
- those that are *local*, i.e., neither of the above.

where a helper function $\tau_{\mathcal{A}ct} : \mathcal{A}ct \to \{\mathsf{creating}, \mathsf{observable}, \mathsf{observing}, \mathsf{local}\}$ can be used to determine the type of a given action. For convenience, we denote the set of all creating actions by $\mathcal{A}ct_{\mathsf{creating}} = \{\mathsf{act} \mid \mathsf{act} \in \mathcal{A}ct, \tau_{\mathcal{A}ct}\,\mathsf{act} = \mathsf{creating}\}$, and accordingly for $\mathcal{A}ct_{\mathsf{observable}}$ as well as for $\mathcal{A}ct_{\mathsf{observing}}$ and $\mathcal{A}ct_{\mathsf{local}}$.

For now, we do not specify which observable, observing, and local actions there are. However, we require $\mathsf{initMT} \in \mathcal{A}ct_{\mathsf{observable}}$ and fix the set of creating actions to be of the form $x = \mathsf{create}(u)$; where $x \in \mathcal{X}$ is a local variable receiving the thread *id* of the newly created thread, and $u \in \mathcal{N}$ defines the thread template for the newly created thread (identified by its first program point).

Apart from that, instantiating this general scheme to concrete sets of actions, and detailing the associated consistency requirements is deferred to Section 2.4. Similarly, we do not fix a concrete formalism and representation for local traces here, and instead describe their properties at a high level of abstraction. A concrete formalism which describes local traces as graphs and fleshes out all the details is supplied in Section 2.5.

Table 2.1: Observable and observing actions and which concurrency primitive they relate to. The primitives targeted by this thesis are in bold font.

Observable Action	Observing Action	Programming Concept
$\mathsf{unlock}(a)$	$\mathsf{lock}(a)$	**Mutex**, Monitor, ...
return	$x'{=}\mathsf{join}(x')$	**Thread Returning / Joining**
$g = x / x = g$	$g = x / x = g$	**Writing/Reading a global variable**
`signal(c)`	`wait(c)`	**Condition Variables**
`send(chan,v)`	`x = receive(chan)`	Channel-Based Concurrency, Sockets, ...
`set_value`	`get`	Futures / Promises

Let $\mathcal{T}$ denote the set of *local traces*. A local trace should be understood as the *local perspective* of a particular thread, the *ego* thread, on the global execution of the system. Each local trace $t \in \mathcal{T}$ ends in some program point u having attained some local program state σ. This pair (u, σ) can be extracted from a local trace t via the function

$$\mathsf{sink} : \mathcal{T} \to \mathcal{N} \times \Sigma$$

We remark that, by construction, the thread *id* contained in $\sigma\,\mathsf{self}$ is the *id* of the ego thread. For convenience, we also define functions $\mathsf{id} : \mathcal{T} \to \mathcal{V}_{\mathbf{tid}}$ and $\mathsf{loc} : \mathcal{T} \to \mathcal{N}$ to return the thread *id* and the program point of the unique sink node, respectively. Furthermore, we abbreviate getting the value of a local variable x from the sink node of a local trace t by $t(x)$, i.e., $t(x) = v$ for $\mathsf{sink}\, t = (u, \sigma)$ and $\sigma\, x = v$.

We assume that there is some set init of initial local traces, i.e., local traces in which exactly one thread is started, namely one derived from the template main. Thus, in particular,

$$t \in \mathsf{init} \implies \mathsf{loc}\, t = u_0 \wedge \mathsf{id}\, t = i_0 \tag{2.1}$$

for the start point u_0 for the thread template main where $i_0 \in \mathcal{V}_{\mathbf{tid}}$ is the thread *id* of the initial thread. Typically, this set init will contain local traces for all possible combinations of values of locals (other than self), as we usually assume that locals may have any value at program start.

For every local trace where the ego thread has already made at least one step, there is a last action in $\mathcal{A}ct$ executed by the ego thread. It can be extracted by the function $\mathsf{last} : \mathcal{T} \to \mathcal{A}ct \cup \{\bot\}$. For local traces in init or local traces where the ego thread has just been started, last returns $\bot$.

W.l.o.g., we assume that every program point u has at most one outgoing edge, along which a new thread is created. We further assume that the new thread receives as its initial configuration the configuration of the creating thread — with the exception that the variable self receives a *fresh* thread *id*. Accordingly, we require a function $\mathsf{new} : \mathcal{N} \to \mathcal{T} \to 2^{\mathcal{T}}$ so that $\mathsf{new}\, u_1\, t$ either returns the empty set, namely, when the creation of a thread which starts at u_1 is not possible for t, or a set $\{t_1\}$ for a single trace t_1 if such thread creation is possible. In the latter case,

$$\mathsf{last}(t_1) = \bot \qquad \mathsf{sink}(t_1) = (u_1, \sigma_1) \tag{2.2}$$

where for $\mathsf{sink}(t) = (u, \sigma)$, $\sigma_1 = \sigma \oplus \{\mathsf{self} \mapsto \nu(t)\}$ for some function $\nu : \mathcal{T} \to \mathcal{V}_{\mathbf{tid}}$ providing us with a fresh thread *id*. As thread *id*s are unique for a given creation history in $\mathcal{T}$, one may, e.g., identify the set $\mathcal{V}_{\mathbf{tid}}$ with $\mathcal{T}$ and let ν be the identity function. We remark that from the informal description above, the following requirement follows

$$t' \in \mathsf{new}\, u_1\, t \implies \exists u' \in \mathcal{N}, x \in \mathcal{X} : (\mathsf{loc}\, t, x = \mathsf{create}(u_1), u') \in \mathcal{E} \tag{2.3}$$

For each edge $e = (u, \mathsf{act}, v)$, we also require an operation $[\![e]\!]_{\mathcal{T}} : \mathcal{T}^k \to 2^{\mathcal{T}}$ where the arity k is 2 for actions that are observing, and 1 for actions which are not. The

returned set of local traces is either empty (i.e., the operation is undefined for the given arguments), singleton (the operation is defined and deterministic), or a larger set (the operation is non-deterministic, e.g., when reading unknown input from a user). For edges with actions that are not observing, this function extends a local trace (provided as the first argument) by executing the corresponding action. For observing actions, the second argument is a further local trace ending in a corresponding observable action, which is additionally incorporated into the local trace, provided both local traces are consistent with the action taken and each other. Such consistency requirements may, e.g., demand that both local traces can be incorporated into a joint computation history. As the formal definition depends on the set of considered actions as well as the details of the representation of local traces, it is deferred to Section 2.5. Instead, we illustrate the notion with an example here.

Example 4. *Consider again the example program from Fig. 2.1, the corresponding CFGs in Fig. 2.2, and some of its local traces in Fig. 2.3. Here, applying* $[\![(u_1, x = \mathsf{create}(t_1), u_2)]\!]_{\mathcal{T}}$ *to the local trace (a) yields the single local trace (b). Applying* $[\![(u_2, \mathsf{lock}(m_g), u_3)]\!]_{\mathcal{T}}$ *to the local trace (b) and the local trace (a) yields the single local trace (d), applying it to (b) and (c) yields the single local trace (e). On the other hand, applying* $[\![(u_2, \mathsf{lock}(m_g), u_3)]\!]_{\mathcal{T}}$ *to (b) and (e) yields an empty set of local traces, as these two local traces are not consistent — among other reasons because the ego thread in (b) would learn about its own future from (e).* □

2.2.1 Global Constraint System

The set of all local traces for a given program can then be computed as the least solution to the following constraint system with the single unknown $[T]$, which takes values from $2^{\mathcal{T}}$ with the order given by $\subseteq$. We give the constraint system as a side-effecting constraint system right away. However, no side-effects are used here, and the system thus is equivalent to an ordinary constraint system.

$$\begin{array}{lcll} [T] & \supseteq & \mathbf{fun}\,\_ \to (\emptyset, \mathsf{init}) & \\ [T] & \supseteq & \mathbf{fun}\,\eta \to (\emptyset, \mathsf{new}\,u_1\,\eta\,[T]), & (u_1 \in \mathcal{N}) \\ [T] & \supseteq & \mathbf{fun}\,\eta \to (\emptyset, [\![e]\!]_{\mathcal{T}}(\eta\,[T], \ldots, \eta\,[T])), & (e \in \mathcal{E}) \end{array} \tag{2.4}$$

For clarity of presentation, we here (and subsequently) abbreviate the longish formula $\bigcup\{f(t_0, \ldots, t_{k-1}) \mid t_0 \in T_0, \ldots, t_{k-1} \in T_{k-1}\}$ for functions $f : \mathcal{T}^k \to 2^{\mathcal{T}}$ and subsets $T_0, \ldots, T_{k-1} \subseteq \mathcal{T}$ by $\bar{f}(T_0, \ldots, T_{k-1})$. Where the type is clear from the context, we also refer to this lifted version by f instead of by $\bar{f}$.

The constraint system (2.4) globally collects all local traces into $[T]$. This happens by, starting from the initial set init of local traces, successively adding the local traces resulting from applying the right-hand side for each edge of the program (as well as the function new) to all (combinations of) local traces discovered in the previous iterations.

Proposition 4. *The constraint system* (2.4) *that globally collects all local traces has a least solution.*

Proof. We observe that the powerset of local traces forms a complete lattice — as does the powerset of any set. Further, as all right-hand sides are defined point-wise, they are monotonic. Thus, by the fixpoint theorem of Knaster-Tarski (recalled as Theorem 1), the system has a unique least solution. □

This unique solution serves as the definition of all (valid) local traces (relative to the definitions of the functions $[\![e]\!]_{\mathcal{T}}$ and new) and, thus, as our reference trace semantics. However, in the later proofs, we require a stronger property, namely that the Kleene fixpoint theorem (as recalled in Theorem 2) is applicable, and thus the least solution is given by the least-upper bound of the Kleene iterates. The Kleene fixpoint theorem applies to functions that are *Scott-continuous* (as recalled in Definition 3).

Proposition 5. *The lifted functions* $(\textbf{\textit{fun}}\ \_ \to \mathit{init}) : 2^{\mathcal{T}} \to 2^{\mathcal{T}}$, $[\![(u, \mathit{act}, u')]\!]_{\mathcal{T}} : 2^{\mathcal{T}} \to 2^{\mathcal{T}}$ *for* $\mathit{act} \notin \mathcal{A}ct_{\mathit{observing}}$ *and* $[\![(u, \mathit{act}, u')]\!]_{\mathcal{T}} : (2^{\mathcal{T}} \times 2^{\mathcal{T}}) \to 2^{\mathcal{T}}$ *for observing actions, as well as the function* $\mathit{new}\, u_1 : 2^{\mathcal{T}} \to 2^{\mathcal{T}}$ *are Scott-continuous.*

Proof. The function always returning init is constant and thus trivially Scott-continuous. For non-observing actions along an edge e, and some directed set $\mathcal{D}$, we verify:

$$\overline{[\![e]\!]_{\mathcal{T}}}(\bigcup_{D\in\mathcal{D}} D) = \bigcup_{t\in(\bigcup_{D\in\mathcal{D}})} [\![e]\!]_{\mathcal{T}}\, t = \bigcup_{D\in\mathcal{D}} \bigcup_{t\in D} [\![e]\!]_{\mathcal{T}}\, t = \bigcup_{D\in\mathcal{D}} \overline{[\![e]\!]_{\mathcal{T}}}\, D$$

The same argument holds for right-hand sides $\mathsf{new}\, u_1$. For observable actions, we show that the function is Scott-continuous for each of its arguments. W.l.o.g., consider fixing the first argument to some set T.

$$\begin{aligned} \overline{[\![e]\!]_{\mathcal{T}}}(T)(\bigcup_{D\in\mathcal{D}} D) &= \bigcup_{t_0\in T} \bigcup_{t_1\in(\bigcup_{D\in\mathcal{D}})} [\![e]\!]_{\mathcal{T}}\, t_0\, t_1 = \bigcup_{D\in\mathcal{D}} \bigcup_{t_1\in D} \bigcup_{t_0\in T} [\![e]\!]_{\mathcal{T}}\, t_0\, t_1 \\ &= \bigcup_{D\in\mathcal{D}} \overline{[\![e]\!]_{\mathcal{T}}}\, T\, D \end{aligned}$$

The proof for the second argument proceeds accordingly, and thus — by Proposition 1 — it follows that the function $[\![\bar{e}]\!]_{\mathcal{T}}$ for observing actions also is Scott-continuous. □

Proposition 6. *The right-hand side function of constraint system* (2.4) *over the powerset lattice of local traces* $2^{\mathcal{T}}$ *is Scott-continuous.*

Proof. We observe that a mapping that maps the only unknown $[T]$ to an element in the powerset of local traces can equivalently be considered just an element of this powerset. Then, we combine all constraints f from (2.4) into one function $F = \bigsqcup_{f\in F} f$ given by

$$\begin{aligned} F\, X = \;& \bigsqcup_{u_1\in\mathcal{N}} (\mathsf{new}\, u_1\, X) \sqcup \left(\bigsqcup_{(u,\mathsf{act},v)\in\mathcal{E},\mathsf{act}\notin\mathcal{A}ct_{\mathsf{observable}}} [\![(u, \mathsf{act}, v)]\!]_{\mathcal{T}}\, X\right) \\ & \sqcup \left(\bigsqcup_{(u,\mathsf{act},v)\in\mathcal{E},\mathsf{act}\in\mathcal{A}ct_{\mathsf{observable}}} ([\![(u, \mathsf{act}, v)]\!]_{\mathcal{T}} \circ \mathsf{dup})\, X\right) \sqcup (\textbf{fun}\ \_ \to \mathsf{init}) \end{aligned}$$

With the observation that the function $\mathsf{dup}(x) = (x, x) : \mathcal{T} \to \mathcal{T} \times \mathcal{T}$ is Scott-continuous, F is the least upper bound of (compositions of) functions that are Scott-continuous (Proposition 5), F is also Scott-continuous by Propositions 2 and 3. □

Proposition 7. *The least solution of the constraint system* (2.4) *is obtained as the least-upper bound of all Kleene iterates.*

Proof. Follows from Kleene's fixpoint theorem (Theorem 2) as the right-hand side function is Scott-continuous and the powerset of local traces forms a complete lattice. □

2.2.2 Localized Constraint System

As a first step towards a constraint system more amenable to abstraction with the end goal of constructing an abstract interpretation, we first provide a *localized* constraint system for local traces. The idea is to have not just one unknown, but several unknowns, corresponding to program points and *observable* actions. At the unknowns corresponding to program points all local traces ending in the respective program point are stored, whereas all local traces in which the last action is observable are additionally stored at the unknown corresponding to this observable action. These unknowns do not have right-hand sides of their own but instead receive all of their values by side-effects. More formally, the set of unknowns here is given by $\{[u] \mid u \in \mathcal{N}\} \cup \{[\mathsf{act}] \mid \mathsf{act} \in \mathcal{A}ct_{\mathsf{observable}}\}$. The constraint system is then given by

$$\begin{array}{lcll} [u_0] & \supseteq & \mathbf{fun}\ \_ \to (\emptyset, \mathsf{init}) & \\ [u'] & \supseteq & [\![(u, \mathsf{act}, u')]\!]_{\mathcal{LT}} & ((u, \mathsf{act}, u') \in \mathcal{E}) \end{array} \tag{2.5}$$

where the right-hand sides $[\![\cdot]\!]_{\mathcal{LT}}$ are defined in terms of the original right-hand sides $[\![\cdot]\!]_{\mathcal{T}}$, distinguishing between the different types of actions. We will now give such right-hand sides, employing an OCaml-like syntax, which we will also use in the rest of this thesis.

Recall that, in order to deal with **thread creation**, the set $\mathcal{A}ct$ provides the actions $x = \mathsf{create}(u_1)$; for $u_1 \in \mathcal{N}$ the first program point in the thread template used to create the thread, and $x \in \mathcal{X}$ a local variable which is meant to receive the thread *id* of the created thread. For creating actions, the right-hand side is given by

$$\begin{array}{lcl} [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{LT}}\, \eta & = & \mathbf{let}\ T = [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\eta\,[u])\ \mathbf{in} \\ & & (\{[u_1] \mapsto \mathsf{new}\, u_1\, (\eta\,[u])\}, T) \end{array}$$

Thus, on top of computing the local trace reaching the subsequent program point, the local trace of the newly started thread is computed and side-effected to the unknown corresponding to the program point where this new thread starts.

For **local** actions act, the right-hand side consists of computing the local trace reaching the endpoint of the control-flow edge by applying $[\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}$, thus

$$[\![(u, \mathsf{act}, u')]\!]_{\mathcal{LT}}\, \eta \quad = \quad (\emptyset, [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\, (\eta\,[u]))$$

Before turning to the right-hand sides for edges corresponding to observable and observing actions, we first introduce a function $\mathsf{observes} : \mathcal{A}ct \to 2^{\mathcal{A}ct}$ that describes for each *observing* action the set of *observable* actions that it may observe. Thus, if an action x is in the set returned by $\mathsf{observes}\, y$, it means that the observing action y may observe the observable action x.

For an **observable** action act, then, on top of computing the local trace for the control flow successor, the set of such traces is side-effected to an unknown $[\mathsf{act}]$ collecting all local traces with last action act. Thus,

$$\begin{array}{lcl} [\![(u, \mathsf{act}, u')]\!]_{\mathcal{LT}}\, \eta & = & \mathbf{let}\ T = [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta\,[u])\ \mathbf{in} \\ & & (\{[\mathsf{act}] \mapsto T\}, T) \end{array}$$

(1) $[\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}$ has type $\mathcal{T} \times \mathcal{T} \to 2^{\mathcal{T}}$ for act observing, $\mathcal{T} \to 2^{\mathcal{T}}$ for other actions

(2) $t \in [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(t_0, \ldots, t_{k-1}) \implies \mathsf{loc}(t_0) = u, \quad \mathsf{last}(t) = \mathsf{act} \quad \text{and} \quad \mathsf{loc}(t) = u'$

(3) $t \in [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(t_0, t_1) \implies \mathsf{last}(t_1) \in \mathsf{observes}(\mathsf{act})$

Figure 2.4: Requirements on $[\![\cdot]\!]_{\mathcal{T}}$, where requirements (2) and (3) are needed for the localized constraint system to be correct.

Conversely, for an **observing** action act, the unknowns for corresponding observable actions need to be considered in order for them to be incorporated.

$$[\![(u, \mathsf{act}, u')]\!]_{\mathcal{LT}}\,\eta = \left(\emptyset, \bigcup_{\substack{\mathsf{act}' \in \\ \mathsf{observes\,act}}} [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta\,[u], \eta\,[\mathsf{act}'])\right)$$

Thus, in this constraint system, $[\![\cdot]\!]_{\mathcal{T}}$ is not applied blindly to all possible combinations of local traces but instead some insights into the characteristics of local traces for which $[\![e]\!]_{\mathcal{T}}$ for some given edge e may yield a non-empty set are exploited to only apply $[\![e]\!]_{\mathcal{T}}$ in those cases. These conditions are outlined in Fig. 2.4, where condition (2) states that $[\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}$ only is defined for traces ending in u, and condition (3) states that executing an observing action only yields a new local trace if the local trace supplied as the second argument actually ends in a *corresponding* observable action.

We next relate this localized constraint system to the running example considered throughout this section.

Example 5. *Consider again the program from Fig. 2.1, some of its local traces (Fig. 2.3), and a solution η of the localized constraint system. Then, each local trace from the figure is part of the set of local traces associated with the unknown corresponding to its sink node. For example, (a) $\in \eta\,[u_1]$ and (b) $\in \eta\,[u_2]$. Considering $\mathsf{unlock}(m_g)$ and initMT to be observable, and $\mathsf{lock}(m_g)$ to be observing of both, the local trace (a) is additionally stored at the unknown $[\mathsf{initMT}]$, and the local trace (c) is additionally stored at the unknown $[\mathsf{unlock}(m_g)]$. When constructing the successors for the local traces associated with the unknown $[u_2]$, i.e., prolonging those local traces with the effect of executing the action $\mathsf{lock}(m_g)$, the unknowns $[\mathsf{initMT}]$ and $[\mathsf{unlock}(m_g)]$ are consulted, as these actions are in the set $\mathsf{observes}(\mathsf{lock}(m_g))$. Combining the local trace (b) stored at $[u_2]$ with the local trace (a) stored at $[\mathsf{initMT}]$ yields the local trace (d). Conversely, combining (b) with (c) stored at the unknown $[\mathsf{unlock}(m_g)]$, yields the local trace (e). Thus, both (d) and (e), are in $\eta\,[u_3]$.* □

To relate solutions of the constraint systems (2.4) and (2.5) to each other, we first need to establish that (2.5) has a least solution and that it can be attained as the least upper bound of the Kleene iterates.

We first observe that, for any set of unknowns X, the set of all maps $\mathsf{X} \to 2^{\mathcal{T}}$ forms a complete lattice with least element $\bot = \{x \mapsto \emptyset \mid x \in \mathsf{X}\}$, greatest element $\top = \{x \mapsto \mathcal{T} \mid x \in \mathsf{X}\}$, and the least upper bound defined point-wise.

Consider a family of functions $\pi_{[x]} : (\mathsf{X} \to 2^{\mathcal{T}}) \to 2^{\mathcal{T}}$, which for an unknown $[x] \in \mathsf{X}$ extracts its value from a map. This function is Scott-continuous. We alternatively denote this function applied to a mapping η for an unknown $[x]$ by $\eta\,[x]$. Also, consider a family of functions $\mathsf{flat}_{[x]} : ((\mathsf{X} \to 2^{\mathcal{T}}) \times 2^{\mathcal{T}}) \to (\mathsf{X} \to 2^{\mathcal{T}})$, which for an unknown $[x] \in \mathsf{X}$ is given by $\mathsf{flat}_{[x]}(S, V) = S \cup \{[x] \mapsto V\}$ with the $\cup$ on maps defined point-wise and $\emptyset$ bindings omitted for clarity. This function is also Scott-continuous.

Proposition 8. *The right-hand side function of constraint system* (2.5) *over the lattice given above is Scott-continuous.*

Proof. We first re-write the right-hand sides and collect all of them in one constraint.

$$F\,\eta = \quad \mathbf{fun}\,\_ \to \mathsf{flat}_{[u_0]}(\emptyset, \mathsf{init}) \sqcup \bigsqcup_{(u,\mathsf{act},u') \in \mathcal{E}} \left(\mathsf{flat}_{[u']}\left(\llbracket (u, \mathsf{act}, u') \rrbracket_{\mathcal{LT}}\,\eta\right)\right)$$

As shown before, the least upper bound of Scott-continuous functions is Scott-continuous. As the function containing init is once more constant, it also is Scott-continuous. It thus remains to show that $\mathsf{flat}_{[u']}\left(\llbracket (u, \mathsf{act}, u') \rrbracket_{\mathcal{LT}}\,\eta\right)$ is Scott-continuous for all possible types of edges. As $\mathsf{flat}_{[x]}$ is Scott-continuous for all $x \in \mathsf{X}$, it thus suffices to study whether the individual $\llbracket \cdot \rrbracket_{\mathcal{LT}}$ are. For thread creation, local actions, and observable actions, this follows directly from their definition, and the Scott-continuity of $\llbracket \cdot \rrbracket_{\mathcal{T}}$, new, and $\pi_{[x]}$ for $[x] \in \mathsf{X}$. The same holds for observing actions, with the observation that

$$\begin{aligned} \llbracket (u, \mathsf{act}, u') \rrbracket_{\mathcal{LT}}\,\eta &= (\emptyset, \textstyle\bigcup_{\mathsf{act}' \in \mathsf{observes}\,\mathsf{act}} \llbracket (u, \mathsf{act}, u') \rrbracket_{\mathcal{T}}(\eta\,[u], \eta\,[\mathsf{act}'])) \\ &= \textstyle\bigcup_{\mathsf{act}' \in \mathsf{observes}\,\mathsf{act}} (\emptyset, \llbracket (u, \mathsf{act}, u') \rrbracket_{\mathcal{T}}(\eta\,[u], \eta\,[\mathsf{act}'])) \end{aligned}$$

Then, F is given as the least upper bound of (compositions of) functions that are Scott-continuous, and is thus also Scott-continuous. □

Proposition 9. *The constraint system* (2.5) *has a least solution which is obtained as the least-upper bound of all Kleene iterates.*

Next, we relate the least solutions of constraint systems (2.4) and (2.5) to each other to show the completeness of the localized constraint system.

Theorem 3. *Let η denote the least solution of the global constraint system* (2.4)*, and $\bar{\eta}$ denote the least solution of the localized constraint system* (2.5)*. Then, provided all right-hand sides $\llbracket \cdot \rrbracket_{\mathcal{T}}$ fulfill the conditions (1) to (3) from Fig. 2.4,*

1. $\bar{\eta}\,[u] = \{t \in \eta\,[T] \mid \mathit{loc}(t) = u\}$ *for all* $u \in \mathcal{N}$;
2. $\bar{\eta}\,[\mathit{act}] = \{t \in \eta\,[T] \mid \mathit{last}(t) = \mathit{act}\}$ *for all* $\mathit{act} \in \mathcal{A}\mathit{ct}_{\mathit{observable}}$.

Proof. The proof is by fixpoint induction. Here — to simplify the proof — we consider the contributions of the constraints one at a time. This can be seen as an instance of a chaotic fixpoint iteration rather than Kleene iteration in that, when considering the contributions of one constraint, we already take the contributions of all previous constraints into account. This is justified by the equivalence between Kleene iteration and chaotic iteration for Scott-continuous right-hand sides over complete lattices [29, 31].
Consider then the i-th approximation η^i to the least solution of (2.4) as well as the i-the approximation $\bar{\eta}^i$ to the least solution of (2.5).
Let us call property (1) that

$$\begin{array}{lcll} \bar{\eta}^i\,[u] & = & \{t \in \eta^i\,[T] \mid \mathsf{loc}\,t = u\} & (u \in \mathcal{N}) \\ \bar{\eta}^i\,[\mathsf{act}] & = & \{t \in \eta^i\,[T] \mid \mathsf{last}\,t = \mathsf{act}\} & (\mathsf{act} \in \mathcal{A}ct_{\mathsf{observable}}) \end{array}$$

For $i = 0$, the value of all unknowns in both constraint systems is equal to $\emptyset$, and property (1) holds.

First, we consider the constraints corresponding to **initialization**. The constraint in the localized constraint system is given by

$$[u_0] \supseteq \mathbf{fun}\,\_ \to (\emptyset, \mathsf{init})$$

The constraint in the global system is given by

$$[T] \supseteq \mathbf{fun}\,\_ \to (\emptyset, \mathsf{init})$$

Let us denote by $(\eta^{i+1}\,[T])^{\oplus}$ the new contribution to $[T]$ and by $(\bar{\eta}^{i+1}\,[u_0])^{\oplus}$ the new contribution to $[u_0]$.

$$\begin{array}{lcll} \{t \in (\eta^{i+1}\,[T])^{\oplus} \mid \mathsf{loc}\,t = u_0\} & = & \{t \in \mathsf{init} \mid \mathsf{loc}\,t = u_0\} & \\ & = & \mathsf{init} & \text{(by (2.1))} \\ & = & (\bar{\eta}^{i+1}\,[u_0])^{\oplus} & \end{array}$$

Since neither constraint system causes any side-effects here, by induction hypothesis, if property (1) holds for the i-th approximations, and these constraints are considered, it also holds for the $(i+1)$-th approximation.

Next, for a constraint corresponding to an edge $(u, \mathsf{act}, u') \in \mathcal{E}$ with a **local** action act. The constraint in the localized constraint system is then given by

$$[u'] \quad \supseteq \quad [\![(u, \mathsf{act}, u')]\!]_{\mathcal{LT}}$$

with

$$[\![(u, \mathsf{act}, u')]\!]_{\mathcal{LT}}\,\eta \quad = \quad (\emptyset, [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\,(\eta\,[u]))$$

As act is local, by Prop (1) from Fig. 2.4, the constraint in the global system is given by

$$[T] \quad \supseteq \quad \mathbf{fun}\,\eta \to (\emptyset, [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta\,[T]))$$

Let us denote by $(\eta^{i+1}\,[T])^{\oplus}$ the new contribution to $[T]$ and by $(\bar{\eta}^{i+1}\,[u'])^{\oplus}$ the new contribution to $[u']$. Then, by induction hypothesis and totality of loc,

$$(\eta^{i+1}\,[T])^{\oplus} = [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta^{i}\,[T]) = [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\bigcup_{u'' \in \mathcal{N}} \bar{\eta}^{i}\,[u''])$$

Thus,

$$\begin{array}{lcll} (\eta^{i+1}\,[T])^{\oplus} & = & [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\bigcup_{u'' \in \mathcal{N}} \bar{\eta}^{i}\,[u'']) & \\ & = & [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\bar{\eta}^{i}\,[u]) & \text{(Prop (2) from Fig. 2.4)} \\ & = & (\bar{\eta}^{i+1}\,[u'])^{\oplus} & \end{array}$$

Furthermore, all resulting local traces end in u', as

$$\begin{array}{ll} & (\eta^{i+1}\,[T])^{\oplus} \\ = & \{t \in (\eta^{i+1}\,[T])^{\oplus} \mid \mathsf{loc}\, t = u'\} \cup \{t \in (\eta^{i+1}\,[T])^{\oplus} \mid \mathsf{loc}\, t \neq u'\} \\ = & \{t \in (\eta^{i+1}\,[T])^{\oplus} \mid \mathsf{loc}\, t = u'\} \cup \{t \in [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta^{i}\,[T]) \mid \mathsf{loc}\, t \neq u'\} \\ = & \{t \in (\eta^{i+1}\,[T])^{\oplus} \mid \mathsf{loc}\, t = u'\} \qquad \text{(Prop (2) from Fig. 2.4)} \end{array}$$

Since neither constraint system causes any side-effects here, by induction hypothesis, if property (1) holds for the i-th approximations, and these constraints are considered, it also holds for the $(i+1)$-th approximation.

Next, we consider a constraint corresponding to an edge $(u, \mathsf{act}, u') \in \mathcal{E}$ for an **observable** action act. The constraint in the localized constraint system is given by

$$[u'] \quad \supseteq \quad [\![(u, \mathsf{act}, u')]\!]_{\mathcal{LT}}$$

with

$$\begin{array}{lcl} [\![(u, \mathsf{act}, u')]\!]_{\mathcal{LT}}\,\eta & = & \textbf{let } T = [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta\,[u]) \textbf{ in} \\ & & (\{[\mathsf{act}] \mapsto T)\}, T) \end{array}$$

As act is observable, by Prop (1) from Fig. 2.4, the constraint in the global system is given by

$$[T] \quad \supseteq \quad \textbf{fun}\, \eta \to (\emptyset, [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta\,[T]))$$

Let us denote by $(\eta^{i+1}\,[T])^{\oplus}$ the new contribution to $[T]$ and by $(\bar{\eta}^{i+1}\,[u'])^{\oplus}$ and $(\bar{\eta}^{i+1}\,[\mathsf{act}])^{\oplus}$ the new contributions to $[u']$ and $[\mathsf{act}]$, respectively. For the contribution to the left-hand side, the same argument applies as in the case of a local action. We thus consider the side-effects caused by the localized constraint system. First, once more by induction hypothesis and totality of loc,

$$(\eta^{i+1}\,[T])^{\oplus} = [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta^{i}\,[T]) = [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\bigcup_{u'' \in \mathcal{N}} \bar{\eta}^{i}\,[u''])$$

Thus,

$$\begin{array}{lcll} & \{t \in (\eta^{i+1}\,[T])^{\oplus} \mid \mathsf{last}\, t = \mathsf{act}\} & & \\ = & \{t \in [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\bigcup_{u'' \in \mathcal{N}} \bar{\eta}^{i}\,[u'']) \mid \mathsf{last}\, t = \mathsf{act}\} & & \\ = & \{t \in [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\bar{\eta}^{i}\,[u]) \mid \mathsf{last}\, t = \mathsf{act}\} & \text{(Prop (2) from Fig. 2.4)} & \\ = & [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\bar{\eta}^{i}\,[u]) & \text{(Prop (2) from Fig. 2.4)} & \\ = & (\bar{\eta}^{i+1}\,[\mathsf{act}])^{\oplus} & & \end{array}$$

We remark that these equalities make use of the fact that $[\![\cdot]\!]_{\mathcal{T}}$ for sets of local traces is defined point-wise. Furthermore, all resulting local traces have last action in act, as

$$\begin{array}{lll} & (\eta^{i+1}\,[T])^{\oplus} & \\ = & \{t \in (\eta^{i+1}\,[T])^{\oplus} \mid \mathsf{last}\, t = \mathsf{act}\} \cup \{t \in (\eta^{i+1}\,[T])^{\oplus} \mid \mathsf{last}\, t \neq \mathsf{act}\} & \\ = & \{t \in (\eta^{i+1}\,[T])^{\oplus} \mid \mathsf{last}\, t = \mathsf{act}\} \cup \{t \in [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta^{i}\,[T]) \mid \mathsf{last}\, t \neq \mathsf{act}\} & \\ = & \{t \in (\eta^{i+1}\,[T])^{\oplus} \mid \mathsf{last}\, t = \mathsf{act}\} & \text{(Prop (2) from Fig. 2.4)} \end{array}$$

Therefore, by induction hypothesis, if property (1) holds for the i-th approximations, and these constraints are considered, it also holds for the $(i+1)$-th approximation.

Next, we consider a constraint corresponding to an edge $(u, \mathsf{act}, u') \in \mathcal{E}$ with an **observing** action act. The constraint in the localized constraint system is given by

$$[u'] \supseteq [\![(u, \mathsf{act}, u')]\!]_{\mathcal{LT}}$$

with

$$[\![(u, \mathsf{act}, v)]\!]_{\mathcal{LT}}\,\eta = \left(\emptyset, \bigcup_{\substack{\mathsf{act}' \in \\ \mathsf{observes\,act}}} [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta\,[u], \eta\,[\mathsf{act}'])\right)$$

The constraint in the global system is given by

$$[T] \supseteq \mathbf{fun}\,\eta \to (\emptyset, [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta\,[T], \eta\,[T]))$$

Let us denote by $(\eta^{i+1}\,[T])^{\oplus}$ the new contribution to $[T]$ and by $(\bar{\eta}^{i+1}\,[u'])^{\oplus}$ the new contributions to $[u']$. Then, once more by induction hypothesis and totality of loc,

$$(\eta^{i+1}\,[T])^{\oplus} = [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta^{i}\,[T], \eta^{i}\,[T]) = [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}((\textstyle\bigcup_{u'' \in \mathcal{N}} \bar{\eta}^{i}\,[u'']), \eta^{i}\,[T])$$

and, by definition,

$$(\bar{\eta}^{i+1}\,[u'])^{\oplus} \;=\; \bigcup_{\substack{\mathsf{act}' \in \\ \mathsf{observes\,act}}} [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\bar{\eta}^{i}\,[u], \bar{\eta}^{i}\,[\mathsf{act}'])$$

Thus,

$$\begin{array}{lll} & (\eta^{i+1}\,[T])^{\oplus} & \\ = & [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}((\textstyle\bigcup_{u'' \in \mathcal{N}} \bar{\eta}^{i}\,[u'']), \eta^{i}\,[T]) & \\ = & [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\bar{\eta}^{i}\,[u], \eta^{i}\,[T]) & \text{(Prop (2) from Fig. 2.4)} \\ = & [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\bar{\eta}^{i}\,[u], \{t \in \eta^{i}\,[T] \mid \mathsf{last}(t) \in \mathsf{observes\,act}\}) & \text{(Prop (3) from Fig. 2.4)} \\ = & [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\bar{\eta}^{i}\,[u], \textstyle\bigcup_{\mathsf{act}' \in \mathsf{observes\,act}} \{t \in \eta^{i}\,[T] \mid \mathsf{last}(t) = \mathsf{act}'\}) & \\ = & [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\bar{\eta}^{i}\,[u], \textstyle\bigcup_{\mathsf{act}' \in \mathsf{observes\,act}} \bar{\eta}^{i}\,[\mathsf{act}']) & \text{(IH)} \\ = & \bigcup_{\substack{\mathsf{act}' \in \\ \mathsf{observes\,act}}} [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\bar{\eta}^{i}\,[u], \bar{\eta}^{i}\,[\mathsf{act}']) & \\ = & (\bar{\eta}^{i+1}\,[u'])^{\oplus} & \end{array}$$

Furthermore, all resulting local traces end in u', as

$$\begin{array}{ll} & (\eta^{i+1}\,[T])^{\oplus} \\ = & \{t \in (\eta^{i+1}\,[T])^{\oplus} \mid \mathsf{loc}\,t = u'\} \cup \{t \in (\eta^{i+1}\,[T])^{\oplus} \mid \mathsf{loc}\,t \neq u'\} \\ = & \{t \in (\eta^{i+1}\,[T])^{\oplus} \mid \mathsf{loc}\,t = u'\} \cup \\ & \{t \in [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta^i\,[T], \eta^i\,[T]) \mid \mathsf{loc}\,t \neq u'\} \\ = & \{t \in (\eta^{i+1}\,[T])^{\oplus} \mid \mathsf{loc}\,t = u'\} \qquad \text{(Prop (2) from Fig. 2.4)} \end{array}$$

Since neither constraint system causes any side-effects here, by induction hypothesis, if property (1) holds for the i-th approximations, and these constraints are considered, it also holds for the $(i+1)$-th approximation.

Last, consider constraints corresponding to a **create** action along an edge $(u, x = \mathsf{create}(u_1), u') \in \mathcal{E}$. In the localized constraint system, the constraint is given by

$$[u'] \supseteq [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{LT}}$$

with

$$\begin{array}{lll} [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{LT}}\,\eta & = & \textbf{let}\ T = [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\eta\,[u])\ \textbf{in} \\ & & (\{[u_1] \mapsto \mathsf{new}\,u_1\,(\eta\,[u])\}, T) \end{array}$$

The relevant constraints in the global system are given by

$$\begin{array}{lll} [T] & \supseteq & \textbf{fun}\,\eta \to (\emptyset, [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\eta\,[T])) \\ [T] & \supseteq & \textbf{fun}\,\eta \to (\emptyset, \mathsf{new}\,u''\,\eta\,[T]) \qquad\qquad (u'' \in \mathcal{N}) \end{array}$$

By Eq. (2.3), for every trace $t' \in \mathsf{new}\,u''\,t$, there is a corresponding edge $(\mathsf{loc}\,t, x = \mathsf{create}(u''), u''') \in \mathcal{E}$. It thus suffices to consider constraints for each edge $(u, x = \mathsf{create}(u_1), u') \in \mathcal{E}$ together with the constraint containing $\mathsf{new}\,u_1$ where one consider as an argument only those local traces that end in u. Thus, the following constraints of the global system are considered:

$$\begin{array}{lll} [T] & \supseteq & \textbf{fun}\,\eta \to (\emptyset, [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\eta\,[T])) \\ [T] & \supseteq & \textbf{fun}\,\eta \to (\emptyset, \mathsf{new}\,u_1\,\{t \in \eta\,[T] \mid \mathsf{loc}\,t = u\}) \end{array}$$

Let us denote by $(\eta^{i+1}\,[T])^{\oplus}$ the new contribution to $[T]$ and by $(\bar{\eta}^{i+1}\,[u'])^{\oplus}$ and $(\bar{\eta}^{i+1}\,[u_1])^{\oplus}$ the new contributions to any $[u']$ and $[u_1]$, respectively. Then

$$\begin{array}{ll} & (\eta^{i+1}\,[T])^{\oplus} \\ = & [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\eta^i\,[T]) \cup \mathsf{new}\,u_1\,(\{t \in \eta^i\,[T] \mid \mathsf{loc}\,t = u\}) \\ = & [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\bigcup_{u'' \in \mathcal{N}} \bar{\eta}^i\,[u'']) \cup \mathsf{new}\,u_1\,(\bar{\eta}^i\,[u]) \qquad \text{(IH and loc total)} \end{array}$$

We now consider separately the cases where $u' = u_1$ and the case where $u' \neq u_1$. First,

for the case where $u' \neq u_1$. Then

$$
\begin{aligned}
& \{t \in (\eta^{i+1}[T]) \mid \mathsf{loc}\, t = u'\} \\
= \; & \{t \in [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\textstyle\bigcup_{u'' \in \mathcal{N}} \bar{\eta}^i[u'']) \mid \mathsf{loc}\, t = u'\} \cup \\
& \{t \in \mathsf{new}\, u_1\, (\bar{\eta}^i[u]) \mid \mathsf{loc}\, t = u'\} \\
= \; & \{t \in [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\textstyle\bigcup_{u'' \in \mathcal{N}} \bar{\eta}^i[u'']) \mid \mathsf{loc}\, t = u'\} && \text{(by Eq. (2.2))} \\
= \; & \{t \in [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\bar{\eta}^i[u]) \mid \mathsf{loc}\, t = u'\} && \text{(Prop (2) from Fig. 2.4)} \\
= \; & [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\bar{\eta}^i[u]) && \text{(Prop (2) from Fig. 2.4)} \\
= \; & (\bar{\eta}^{i+1}[u'])^{\oplus}
\end{aligned}
$$

and

$$
\begin{aligned}
& \{t \in (\eta^{i+1}[T]) \mid \mathsf{loc}\, t = u_1\} \\
= \; & \{t \in [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\textstyle\bigcup_{u'' \in \mathcal{N}} \bar{\eta}^i[u'']) \mid \mathsf{loc}\, t = u_1\} \\
& \cup \{t \in \mathsf{new}\, u_1\, (\bar{\eta}^i[u]) \mid \mathsf{loc}\, t = u_1\} \\
= \; & \{t \in \mathsf{new}\, u_1\, (\bar{\eta}^i[u]) \mid \mathsf{loc}\, t = u_1\} && \text{(Prop (2) from Fig. 2.4)} \\
= \; & \mathsf{new}\, u_1\, (\bar{\eta}^i[u]) && \text{(by Eq. (2.2))} \\
= \; & (\bar{\eta}^{i+1}[u_1])^{\oplus}
\end{aligned}
$$

and the property holds. Now for the case where $u' = u_1$.

$$
\begin{aligned}
& \{t \in (\eta^{i+1}[T]) \mid \mathsf{loc}\, t = u'\} \\
= \; & \{t \in [\![(u, x = \mathsf{create}(u'), u')]\!]_{\mathcal{T}}(\textstyle\bigcup_{u'' \in \mathcal{N}} \bar{\eta}^i[u'']) \mid \mathsf{loc}\, t = u'\} \\
& \cup \{t \in \mathsf{new}\, u'\, (\bar{\eta}^i[u]) \mid \mathsf{loc}\, t = u'\} \\
= \; & \{t \in [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\bar{\eta}^i[u]) \mid \mathsf{loc}\, t = u'\} \\
& \cup \{t \in \mathsf{new}\, u'\, (\bar{\eta}^i[u]) \mid \mathsf{loc}\, t = u'\} && \text{(Prop (2) from Fig. 2.4)} \\
= \; & [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\bar{\eta}^i[u]) \\
& \cup \{t \in \mathsf{new}\, u'\, (\bar{\eta}^i[u]) \mid \mathsf{loc}\, t = u'\} && \text{(Prop (2) from Fig. 2.4)} \\
= \; & [\![(u, x = \mathsf{create}(u'), u')]\!]_{\mathcal{T}}(\bar{\eta}^i[u]) \cup \mathsf{new}\, u'\, (\bar{\eta}^i[u]) && \text{(by Eq. (2.2))} \\
= \; & (\bar{\eta}^{i+1}[u'])^{\oplus}
\end{aligned}
$$

Furthermore, all resulting traces end either in u' or u_1 as

$$
\begin{aligned}
& (\eta^{i+1}[T])^{\oplus} \\
= \; & \{t \in (\eta^{i+1}[T])^{\oplus} \mid \mathsf{loc}\, t \in \{u', u_1\}\} \cup \\
& \{t \in (\eta^{i+1}[T])^{\oplus} \mid \mathsf{loc}\, t \notin \{u', u_1\}\} \\
= \; & \{t \in (\eta^{i+1}[T])^{\oplus} \mid \mathsf{loc}\, t \in \{u', u_1\}\} \cup \\
& \{t \in [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\eta^i[T]) \mid \mathsf{loc}\, t \notin \{u', u_1\}\} \cup \\
& \{t \in \mathsf{new}\, u_1\, (\{t \in \eta^i[T] \mid \mathsf{loc}\, t = u\}) \mid \mathsf{loc}\, t \notin \{u', u_1\}\} \\
= \; & \{t \in (\eta^{i+1}[T])^{\oplus} \mid \mathsf{loc}\, t \in \{u', u_1\}\} \cup \\
& \{t \in \mathsf{new}\, u_1\, (\{t \in \eta^i[T] \mid \mathsf{loc}\, t = u\}) \mid \mathsf{loc}\, t \notin \{u', u_1\}\} && \text{(Prop (2) from Fig. 2.4)} \\
= \; & \{t \in (\eta^{i+1}[T])^{\oplus} \mid \mathsf{loc}\, t \in \{u', u_1\}\} && \text{(by Eq. (2.2))}
\end{aligned}
$$

Therefore, by induction hypothesis, if property (1) holds for the i-th approximations, and these constraints are considered, it also holds for the $(i+1)$-th approximation. This concludes the case distinction over the different actions. Thus, altogether, if property (1) holds for the i-th approximations, it also holds for the $(i+1)$-th approximation. □

Thus, both the global and the localized constraint systems describe the same set of local traces when the appropriate side conditions from Fig. 2.4 are met. To construct appropriate abstractions, we will focus on the localized constraint system — as it provides us with flow-sensitivity within each thread without having to code program locations into abstract states by having a separate unknown for each program point.

2.3 Digests & Refined Constraint System

Massaging a constraint system to obtain dedicated unknowns per program point, which is part of what was done in the previous section, is one of the standard ways to obtain a flow-sensitive abstract interpretation once the domain of the unknowns is changed from sets of (local) traces to abstract values from some suitable domain — without having to include, e.g., a dedicated abstraction of the program counter in the abstract states.

In the multi-threaded setting, the step from the global to the localized constraint system is a step towards an analysis that is flow-sensitive within each thread. Later, we aim for analyses that go beyond this, and are, in some sense, sensitive in further abstractions of the computational past.

One way to do this would be to fix a specific refinement, construct a bespoke analysis considering this refinement, and then provide an intricate correctness argument relating abstract values for split unknowns of the constraint system arising from the analysis to the values of (non-split) unknowns of the localized concrete constraint system.

Here, we take a different approach and instead, just as we did when giving the localized constraint system, provide for the refinement already at the level of the concrete semantics. In this way, we can argue about general refinements according to abstractions of the computational history dubbed *digests*. Many of the analyses in later chapters as well as their soundness proofs will be generic in the digest, and thus allow for refinement according to any digest, as long as it satisfies some properties, i.e., is *admissible*. Thus, the soundness proof of an analysis is decoupled from proving the admissibility of some digest — which provides an elegant separation of concerns.

Let $\mathcal{A}$ be some set of information that can be extracted from a local trace by a (total) function $\alpha_{\mathcal{A}} : \mathcal{T} \to \mathcal{A}$. We call $\alpha_{\mathcal{A}}\, t \in \mathcal{A}$ the *digest* of some local trace t. Let $[\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}} : \mathcal{A}^k \to 2^{\mathcal{A}}$ be the effect on the digest when performing a k-ary action $\mathsf{act} \in \mathcal{A}ct$ for a control flow edge originating at u. We require for $e = (u, \mathsf{act}, v) \in \mathcal{E}$,

$$\forall A_0, \ldots, A_{k-1} \in \mathcal{A} \quad : \quad |[\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0, \ldots, A_{k-1})| \leq 1 \tag{2.6}$$

$$\forall t_0, \ldots, t_{k-1} \in \mathcal{T} \quad : \quad \alpha_{\mathcal{A}}([\![e]\!]_{\mathcal{T}}(t_0, \ldots, t_{k-1})) \subseteq [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(\alpha_{\mathcal{A}}\, t_0, \ldots, \alpha_{\mathcal{A}}\, t_{k-1}) \tag{2.7}$$

where $\alpha_{\mathcal{A}}$ is lifted element-wise to sets. While the first restriction ensures determinism, the second intuitively ensures that $[\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}$ soundly abstracts $[\![e]\!]_{\mathcal{T}}$.

For thread creation, we additionally require a helper function $\mathsf{new}^\sharp_{\mathcal{A}} : \mathcal{N} \to \mathcal{N} \to \mathcal{A} \to 2^{\mathcal{A}}$ that returns for a thread created at an edge originating from u and starting execution

at program point u_1 the new digest.

$$\forall A_0 \in \mathcal{A} \quad : \quad |\mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0| \leq 1 \tag{2.8}$$

$$\forall t_0 \in \mathcal{T} \quad : \quad \alpha_{\mathcal{A}}(\mathsf{new}\, u_1\, t_0) \subseteq \mathsf{new}^\sharp_{\mathcal{A}}\, \mathsf{loc}(t_0)\, u_1\, (\alpha_{\mathcal{A}}\, t_0) \tag{2.9}$$

Furthermore, we require that whenever there is a successor digest for the creating thread, there is also an appropriate digest for the newly created thread, i.e.,

$$[\![u, x{=}\mathsf{create}(u_1)]\!]^\sharp_{\mathcal{A}}(A_0) \neq \emptyset \implies \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0 \neq \emptyset \tag{2.10}$$

Also, we define for the initial digest at the start of the program

$$\mathsf{init}^\sharp_{\mathcal{A}} \quad = \quad \{\alpha_{\mathcal{A}}\, t \mid t \in \mathsf{init}\} \tag{2.11}$$

We will later refer to a set $\mathcal{A}$ together with suitable definitions of $\alpha_{\mathcal{A}}$, $[\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}$, $\mathsf{new}^\sharp_{\mathcal{A}}$, and $\mathsf{init}^\sharp_{\mathcal{A}}$ as the $\mathcal{A}-$*digest*. It will always be clear from the context whether we are referring to a single element $A \in \mathcal{A}$ or the whole system when talking about a digest.

Definition 4. *A digest $\mathcal{A}$ is* admissible*, when $\alpha_{\mathcal{A}}$, $\mathsf{new}^\sharp_{\mathcal{A}}$, and $[\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}$ fulfill the requirements* (2.6)*,* (2.7)*,* (2.8)*,* (2.9)*,* (2.10)*, and* (2.11)*.*

When a digest $\mathcal{A}$ is admissible, it is possible to perform *control-point splitting* according to the elements of $\mathcal{A}$. This means that unknowns $[u]$ for program points u are replaced with new unknowns $[u, A]$, $A \in \mathcal{A}$. Analogously, unknowns for observable actions $[\mathsf{act}]$ are replaced with unknowns $[\mathsf{act}, A]$ for $A \in \mathcal{A}$, yielding the following new constraint system:

$$\begin{aligned}
[u_0, A] \quad &\supseteq \quad \mathbf{fun}\, _ \to (\emptyset, \{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\}) \\
&\quad \text{for } A \in \mathcal{A} \\
[u', A'] \quad &\supseteq \quad ([\![([u, A_0], x = \mathsf{create}(u_1), u')]\!]_{\mathcal{LTA}}\, \eta) \\
&\quad \text{for } (u, x = \mathsf{create}(u_1), u') \in \mathcal{E}, A' \in [\![u, x = \mathsf{create}(u_1)]\!]^\sharp_{\mathcal{A}}(A_0) \\
[u', A'] \quad &\supseteq \quad ([\![([u, A_0], \mathsf{act}, u'), A_1]\!]_{\mathcal{LTA}}\, \eta) \\
&\quad \text{for } (u, \mathsf{act}, u') \in \mathcal{E}, \mathsf{act} \in \mathcal{A}ct_{\mathsf{observing}}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0, A_1) \\
[u', A'] \quad &\supseteq \quad ([\![([u, A_0], \mathsf{act}, [u', A'])]\!]_{\mathcal{LTA}}\, \eta) \\
&\quad \text{for } (u, \mathsf{act}, u') \in \mathcal{E}, \mathsf{act} \in \mathcal{A}ct_{\mathsf{observable}}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0) \\
[u', A'] \quad &\supseteq \quad ([\![([u, A_0], \mathsf{act}, u')]\!]_{\mathcal{LTA}}\, \eta) \\
&\quad \text{for } (u, \mathsf{act}, u') \in \mathcal{E}, \mathsf{act} \in \mathcal{A}ct_{\mathsf{local}}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0)
\end{aligned} \tag{2.12}$$

for $A_0, A_1 \in \mathcal{A}$. The right-hand sides in the resulting constraint system thus always additionally receive the digest of the local trace about to be extended, plus potentially some extra digests to be able to access the correct unknowns and/or to re-direct side-effects to the appropriate unknowns:

- When act is *observing*, the new right-hand side additionally gets the digest A_1 associated with the local traces that are to be incorporated;
- When act is *observable*, the digest A' of the resulting local trace is passed, so the side-effect can be redirected to the appropriate unknown.

The right-hand sides are then given for a create action by

$$\begin{aligned}[\![([u, A_0], x = \mathsf{create}(u_1), u')]\!]_{\mathcal{LTA}}\,\eta &= \textbf{let } T = [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\eta\,[u, A_0])\textbf{ in} \\ &\quad (\{[u_1, \mathsf{new}^\sharp_{\mathcal{A}}\,u\,u_1\,A_0] \mapsto \mathsf{new}\,u_1\,(\eta\,[u, A_0])\}, T)\end{aligned}$$

where, by abuse of notation, we denote by $\mathsf{new}^\sharp_{\mathcal{A}}\,u\,u_1\,A_0$ the single element of that set, which has at most one element by (2.8) and is non-empty here by (2.10). For observing actions act, the right-hand sides are given by

$$[\![([u, A_0], \mathsf{act}, u'), A_1]\!]_{\mathcal{LTA}}\,\eta = \left(\emptyset,\ \bigcup_{\substack{\mathsf{act}' \in \\ \mathsf{observes\,act}}} [\![(u, \mathsf{act}, v)]\!]_{\mathcal{T}}\,(\eta\,[u, A_0], \eta\,[\mathsf{act}', A_1])\right)$$

and for an observable action act by

$$\begin{aligned}[\![([u, A_0], \mathsf{act}, [u', A'])]\!]_{\mathcal{LTA}}\,\eta &= \textbf{let } T = [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta\,[u, A_0])\textbf{ in} \\ &\quad (\{[\mathsf{act}, A'] \mapsto T\}, T)\end{aligned}$$

and for all local actions act by

$$\begin{aligned}[\![([u, A_0], \mathsf{act}, u')]\!]_{\mathcal{LTA}}\,\eta &= \textbf{let } T = [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta\,[u, A_0])\textbf{ in} \\ &\quad (\emptyset, T)\end{aligned}$$

Before we can relate (least) solutions of this constraint system to least solutions of (2.5), we once again need to establish that the new constraint system has a least solution and that it is attained as the least upper bound of all Kleene iterates.

Proposition 10. *The right-hand side function of constraint system* (2.5) *over the lattice mapping (extended) unknowns to sets of local traces with the order as discussed in Section 2.2.2 is Scott-continuous.*

Proof. As a first step, once again all right-hand sides are collected together in one constraint.

$$\begin{aligned}F\,\eta &= (\textstyle\bigsqcup_{A \in \mathcal{A}}(\textbf{fun}\,\_ \to \mathsf{flat}_{[u_0, A]}(\emptyset, \{t \mid t \in \mathsf{init} A = \alpha_{\mathcal{A}}(t)\})))\sqcup \\ &\quad F_{\mathsf{creating}}\,\eta \sqcup F_{\mathsf{observing}}\,\eta \sqcup F_{\mathsf{observable}}\,\eta \sqcup F_{\mathsf{local}}\,\eta\end{aligned}$$

with

$$\begin{aligned}
F_{\mathsf{creating}}\,\eta &= \bigsqcup_{(u,x=\mathsf{create}(u_1),u')\in\mathcal{E},A'\in[\![u,x=\mathsf{create}(u_1)]\!]^\sharp_{\mathcal{A}}(A_0)} \Big(\mathsf{flat}_{[u',A']}\,([\![([u,A_0],x=\mathsf{create}(u_1),u')]\!]_{\mathcal{LTA}}\,\eta)\Big)\\
F_{\mathsf{observing}}\,\eta &= \bigsqcup_{(u,\mathsf{act},u')\in\mathcal{E},\mathsf{act}\in\mathcal{A}ct_{\mathsf{observing}},A'\in[\![u,\mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0,A_1)} \Big(\mathsf{flat}_{[u',A']}\,([\![([u,A_0],\mathsf{act},u'),A_1]\!]_{\mathcal{LTA}}\,\eta)\Big)\\
F_{\mathsf{observable}}\,\eta &= \bigsqcup_{(u,\mathsf{act},u')\in\mathcal{E},\mathsf{act}\in\mathcal{A}ct_{\mathsf{observable}},A'\in[\![u,\mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0)} \Big(\mathsf{flat}_{[u',A']}\,([\![([u,A_0],\mathsf{act},[u',A'])]\!]_{\mathcal{LTA}}\,\eta)\Big)\\
F_{\mathsf{local}}\,\eta &= \bigsqcup_{(u,\mathsf{act},u')\in\mathcal{E},\mathsf{act}\in\mathcal{A}ct_{\mathsf{local}},A'\in[\![u,\mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0)} \Big(\mathsf{flat}_{[u',A']}\,([\![([u,A_0],\mathsf{act},u')]\!]_{\mathcal{LTA}}\,\eta)\Big)
\end{aligned}$$

for $A_0, A_1 \in \mathcal{A}$. As the least upper-bound of Scott-continuous functions is Scott-continuous, $\mathsf{flat}_{[x]}$ is Scott-continuous, it once more suffices to check whether the individual functions per edge ($[\![\cdot]\!]_{\mathcal{LTA}}$ here) are also Scott-continuous. This once more follows directly from their definition, and the Scott-continuity of $[\![\cdot]\!]_{\mathcal{T}}$, new, and $\pi_{[x]}$ for $[x] \in \mathsf{X}$, where the same insight into the right-hand side for observable actions already used in the proof of Proposition 8 is used again. Then, F is given as the least upper bound of (compositions of) functions that are Scott-continuous, and is thus also Scott-continuous. □

Proposition 11. *The constraint system* (2.12) *has a least solution which is obtained as the least-upper bound of all Kleene iterates.*

Next, we relate the least solution of a constraint system (2.12) for an admissible digest $\mathcal{A}$ to the least solution of the constraint system (2.5) to establish soundness and completeness of the family of refined constraint systems.

Let η be the unique least solution of the constraint system from (2.5). We construct from it a mapping $\bar{\eta}$ from the unknowns of constraint system (2.12) to $2^{\mathcal{T}}$ by

$$\begin{aligned}
\bar{\eta}\,[u,A] &= \eta\,[u] \cap \mathcal{T}_A && (u \in \mathcal{N}, A \in \mathcal{A})\\
\bar{\eta}\,[\mathsf{act},A] &= \eta\,[\mathsf{act}] \cap \mathcal{T}_A && (\mathsf{act} \in \mathcal{A}ct, \mathsf{act}\ \text{observable}, A \in \mathcal{A})
\end{aligned}$$

where $\mathcal{T}_A$ denotes the subset of local traces where $\{t \mid t \in \mathcal{T}, \alpha_{\mathcal{A}}(t) = A\}$.

Theorem 4. *Provided that the digest $\mathcal{A}$ is admissible, $\bar{\eta}$ is the least solution of the constraint system* (2.12) *if and only if η is the least solution of the constraint system* (2.5).

Proof. The proof is by fixpoint induction along the same lines as the proof of Theorem 3. Consider the i-th approximation η^i to the least solution of (2.5) as well as the i-the approximation $\bar{\eta}^i$ to the least solution of (2.12).

Let us call property (1) that

$$\begin{aligned}
\bar{\eta}^i\,[u,A] &= \eta^i\,[u] \cap \mathcal{T}_A && (u \in \mathcal{N}, A \in \mathcal{A})\\
\bar{\eta}^i\,[\mathsf{act},A] &= \eta^i\,[\mathsf{act}] \cap \mathcal{T}_A && (\mathsf{act} \in \mathcal{A}ct_{\mathsf{observable}}, A \in \mathcal{A})
\end{aligned}$$

For $i = 0$, the value of all unknowns in both constraint systems is equal to $\emptyset$ and property (1) holds.

Next, we show that for constraints corresponding to a control-flow edge as well as the constraint for initialization executed in lock-step, provided that property (1) holds before the update, it still holds after the update. For this, it suffices to consider the contribution of the constraint as well as its side-effects (if any are triggered).

First, consider the constraints corresponding to the **initialization**

$$\begin{array}{lll} [u_0, A] & \supseteq & \textbf{fun}\ \_ \to (\emptyset, \{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\}) \text{ for } A \in \mathcal{A} \\ [u_0] & \supseteq & \textbf{fun}\ \_ \to (\emptyset, \mathsf{init}) \end{array}$$

The right-hand side does not access any unknowns; it thus suffices to verify that

$$\{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\} = \mathsf{init} \cap \mathcal{T}_A$$

holds for all $A \in \mathcal{A}$. Thus, if property (1) holds for the i-th approximations, and these constraints are considered, it also holds for the $(i+1)$-th approximation.

Now, consider a **local** action act and an edge $(u, \mathsf{act}, u') \in \mathcal{E}$. The family of constraints in the refined constraint system is given by

$$[u', A'] \quad \supseteq \quad [\![([u, A_0], \mathsf{act}, u')]\!]_{\mathcal{LTA}} \quad \text{for } A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0)$$

with

$$\begin{array}{lll} [\![([u, A_0], \mathsf{act}, u')]\!]_{\mathcal{LTA}}\, \eta & = & \textbf{let } T = [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta\,[u, A_0]) \textbf{ in} \\ & & (\emptyset, T) \end{array}$$

The constraints of the localized system are given by

$$[u'] \quad \supseteq \quad [\![(u, \mathsf{act}, u')]\!]_{\mathcal{LT}} \qquad ((u, \mathsf{act}, u') \in \mathcal{E})$$

with

$$[\![(u, \mathsf{act}, u')]\!]_{\mathcal{LT}}\, \eta \quad = \quad (\emptyset, [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\,(\eta\,[u]))$$

Let us denote by $(\eta^{i+1}\,[u'])^{\oplus}$ the new contribution to $[u']$ and by $(\bar{\eta}^{i+1}\,[u', A'])^{\oplus}$ the new contributions to $[u', A']$. Then, by induction hypothesis and α total,

$$(\eta^{i+1}\,[u'])^{\oplus} = [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\,(\eta^i\,[u]) = [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\,\left(\bigcup_{A \in \mathcal{A}} \bar{\eta}^i\,[u, A]\right)$$

and

$$(\bar{\eta}^{i+1}\,[u', A'])^{\oplus} \quad = \quad \bigcup_{A_0 \in \mathcal{A}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0)} [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\,(\bar{\eta}^i\,[u, A_0])$$

Thus,

$$\begin{array}{lll}
& (\eta^{i+1}\,[u'])^{\oplus} \cap \mathcal{T}_{A'} & \\
= & [\![(u,\mathsf{act},u')]\!]_{\mathcal{T}}\ \left(\bigcup_{A_0\in\mathcal{A}} \bar{\eta}^i\,[u,A_0]\right) \cap \mathcal{T}_{A'} & \\
= & \left(\bigcup_{A_0\in\mathcal{A}} [\![(u,\mathsf{act},u')]\!]_{\mathcal{T}}(\bar{\eta}^i\,[u,A_0])\right) \cap \mathcal{T}_{A'} & \\
= & \left(\left(\bigcup_{A_0\in\mathcal{A},A'\in[\![u,\mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0)} [\![(u,\mathsf{act},u')]\!]_{\mathcal{T}}(\bar{\eta}^i\,[u,A_0])\right) \cap \mathcal{T}_{A'}\right) \cup & \\
& \left(\left(\bigcup_{A_0\in\mathcal{A},A'\notin[\![u,\mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0)} [\![(u,\mathsf{act},u')]\!]_{\mathcal{T}}(\bar{\eta}^i\,[u,A_0])\right) \cap \mathcal{T}_{A'}\right) & \\
= & \left(\left(\bigcup_{A_0\in\mathcal{A},A'\in[\![u,\mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0)} [\![(u,\mathsf{act},u')]\!]_{\mathcal{T}}(\bar{\eta}^i\,[u,A_0])\right) \cap \mathcal{T}_{A'}\right) \cup \emptyset & \text{(by (2.7))} \\
= & \bigcup_{A_0\in\mathcal{A},A'\in[\![u,\mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0)} [\![(u,\mathsf{act},u')]\!]_{\mathcal{T}}(\bar{\eta}^i\,[u,A_0]) & \text{(by (2.6))} \\
= & (\bar{\eta}^{i+1}\,[u',A'])^{\oplus} &
\end{array}$$

Since neither constraint system causes any side-effects here, if property (1) holds for the i-th approximations, and these constraints are considered, it also holds for the $(i+1)$-th approximation.

Now consider an **observable action** act along an edge $(u,\mathsf{act},u') \in \mathcal{E}$. The family of constraints in the refined constraint system is given by

$$[u',A'] \quad\supseteq\quad [\![([u,A_0],\mathsf{act},[u',A'])]\!]_{\mathcal{LTA}} \quad \text{for } A' \in [\![u,\mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0)$$

with

$$\begin{array}{rcl}
[\![([u,A_0],\mathsf{act},[u',A'])]\!]_{\mathcal{LTA}}\,\eta & = & \textbf{let } T = [\![(u,\mathsf{act},u')]\!]_{\mathcal{T}}(\eta\,[u,A_0]) \textbf{ in} \\
& & (\{[\mathsf{act},A'] \mapsto T\},T)
\end{array}$$

The constraints of the localized system are given by

$$[u'] \quad\supseteq\quad [\![(u,\mathsf{act},u')]\!]_{\mathcal{LT}}$$

with

$$\begin{array}{rcl}
[\![(u,\mathsf{act},v)]\!]_{\mathcal{LT}}\,\eta & = & \textbf{let } T = [\![(u,\mathsf{act},v)]\!]_{\mathcal{T}}(\eta\,[u]) \textbf{ in} \\
& & (\{[\mathsf{act}] \mapsto T\},T)
\end{array}$$

For the contribution to the left-hand sides, the same argument as given for the case of a local action applies. We thus turn to the triggered side-effects: Let us denote by $(\eta^{i+1}\,[\mathsf{act}])^{\oplus}$ the new contribution (via side-effect) to $[\mathsf{act}]$ and by $(\bar{\eta}^{i+1}\,[\mathsf{act},A'])^{\oplus}$ the new contributions to $[\mathsf{act},A']$. Then, by induction hypothesis and α total,

$$(\eta^{i+1}\,[\mathsf{act}])^{\oplus} \quad=\quad [\![(u,\mathsf{act},u')]\!]_{\mathcal{T}}\,(\eta^i\,[u]) = [\![(u,\mathsf{act},u')]\!]_{\mathcal{T}}\ \left(\bigcup_{A\in\mathcal{A}} \bar{\eta}^i\,[u,A]\right)$$

and

$$(\bar{\eta}^{i+1}\,[\mathsf{act},A'])^{\oplus} \quad=\quad \bigcup_{A_0\in\mathcal{A},A'\in[\![u,\mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0)} [\![(u,\mathsf{act},u')]\!]_{\mathcal{T}}\,(\bar{\eta}^i\,[u,A_0])$$

Thus,

$$(\eta^{i+1}\,[\mathsf{act}])^{\oplus} \cap \mathcal{T}_{A'} \quad=\quad (\bar{\eta}^{i+1}\,[\mathsf{act},A'])^{\oplus}$$

once again by the same reasoning as for the contribution to the left-hand side in case of local actions. Therefore, if property (1) holds for the i-th approximations, and these constraints are considered, property (1) also holds for the $(i+1)$-th approximation.

Now consider an **observing action** act along an edge $(u, \mathsf{act}, u') \in \mathcal{E}$. The family of constraints in the refined constraint system is then given by

$$[u', A'] \supseteq [\![([u, A_0], \mathsf{act}, u'), A_1]\!]_{\mathcal{LTA}} \qquad \text{for } A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0, A_1)$$

with

$$[\![([u, A_0], \mathsf{act}, u'), A_1]\!]_{\mathcal{LTA}}\, \eta = \left(\emptyset, \bigcup_{\substack{\mathsf{act}' \in \\ \text{observes act}}} [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\, (\eta\, [u, A_0], \eta\, [\mathsf{act}', A_1]) \right)$$

The constraints of the localized system are given by

$$[u'] \quad \supseteq \quad [\![(u, \mathsf{act}, u')]\!]_{\mathcal{LT}}$$

with

$$[\![(u, \mathsf{act}, u')]\!]_{\mathcal{LT}}\, \eta = \left(\emptyset, \bigcup_{\substack{\mathsf{act}' \in \\ \text{observes act}}} [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}} (\eta\, [u], \eta\, [\mathsf{act}']) \right)$$

Let us denote by $(\eta^{i+1}\, [u'])^{\oplus}$ the new contribution to $[u']$ and by $(\bar{\eta}^{i+1}\, [u', A'])^{\oplus}$ the new contributions to $[u', A']$. Then

$$\begin{array}{ll} & (\eta^{i+1}\, [u'])^{\oplus} \\ = & \bigcup_{\substack{\mathsf{act}' \in \\ \text{observes act}}} [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\, (\eta^i\, [u], \eta^i\, [\mathsf{act}']) \\ = & \bigcup_{\substack{\mathsf{act}' \in \\ \text{observes act}}} [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\, \left(\bigcup_{A \in \mathcal{A}} \bar{\eta}^i\, [u, A], \bigcup_{A \in \mathcal{A}} \bar{\eta}^i\, [\mathsf{act}', A]\right) \quad \text{(by IH and } \alpha \text{ total)} \end{array}$$

and

$$\begin{array}{ll} & (\bar{\eta}^{i+1}\, [u', A'])^{\oplus} \\ = & \bigcup_{A_0 \in \mathcal{A}, A_1 \in \mathcal{A}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0, A_1)} \bigcup_{\substack{\mathsf{act}' \in \\ \text{observes act}}} [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\, \left(\bar{\eta}^i\, [u, A_0], \bar{\eta}^i\, [\mathsf{act}', A_1]\right) \end{array}$$

Thus,

$$\begin{array}{ll} & (\eta^{i+1}\, [u'])^{\oplus} \cap \mathcal{T}_{A'} \\ = & \left(\bigcup_{\substack{\mathsf{act}' \in \\ \text{observes act}}} [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\, \left(\left(\bigcup_{A \in \mathcal{A}} \bar{\eta}^i\, [u, A]\right), \left(\bigcup_{A \in \mathcal{A}} \bar{\eta}^i\, [\mathsf{act}', A]\right) \right) \right) \cap \mathcal{T}_{A'} \\ = & \bigcup_{\substack{\mathsf{act}' \in \\ \text{observes act}}} \left([\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\, \left(\left(\bigcup_{A \in \mathcal{A}} \bar{\eta}^i\, [u, A]\right), \left(\bigcup_{A \in \mathcal{A}} \bar{\eta}^i\, [\mathsf{act}', A]\right) \right) \cap \mathcal{T}_{A'} \right) \end{array}$$

Now, consider for each $\mathsf{act}' \in \mathsf{observes\,act}$:

$$\begin{array}{ll} & [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\, \left(\left(\bigcup_{A \in \mathcal{A}} \bar{\eta}^i\, [u, A]\right), \left(\bigcup_{A \in \mathcal{A}} \bar{\eta}^i\, [\mathsf{act}', A]\right) \right) \cap \mathcal{T}_{A'} \\ = & \bigcup_{A_0 \in \mathcal{A}, A_1 \in \mathcal{A}} \left([\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\, \left(\bar{\eta}^i\, [u, A_0], \bar{\eta}^i\, [\mathsf{act}', A_1] \right) \cap \mathcal{T}_{A'} \right) \\ = & \bigcup_{A_0 \in \mathcal{A}, A_1 \in \mathcal{A}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0, A_1)} \left([\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\, \left(\bar{\eta}^i\, [u, A_0], \bar{\eta}^i\, [\mathsf{act}', A_1] \right) \cap \mathcal{T}_{A'} \right) \cup \\ & \bigcup_{A_0 \in \mathcal{A}, A_1 \in \mathcal{A}, A' \notin [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0, A_1)} \left([\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\, \left(\bar{\eta}^i\, [u, A_0], \bar{\eta}^i\, [\mathsf{act}', A_1] \right) \cap \mathcal{T}_{A'} \right) \\ = & \bigcup_{A_0 \in \mathcal{A}, A_1 \in \mathcal{A}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0, A_1)} \left([\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\, \left(\bar{\eta}^i\, [u, A_0], \bar{\eta}^i\, [\mathsf{act}', A_1] \right) \cap \mathcal{T}_{A'} \right) \cup \\ & \emptyset \qquad \text{(by (2.7))} \\ = & \bigcup_{A_0 \in \mathcal{A}, A_1 \in \mathcal{A}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0, A_1)} \left([\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\, \left(\bar{\eta}^i\, [u, A_0], \bar{\eta}^i\, [\mathsf{act}', A_1] \right) \right) \qquad \text{(by (2.6))} \end{array}$$

Inserting into the equation above, we obtain

$$\begin{array}{rl} & (\eta^{i+1}\,[u'])^{\oplus} \cap \mathcal{T}_{A'} \\ = & \bigcup_{\substack{\mathsf{act}' \in \\ \text{observes act}}} \left([\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\ \left(\left(\bigcup_{A \in \mathcal{A}} \bar{\eta}^i\,[u, A] \right), \left(\bigcup_{A \in \mathcal{A}} \bar{\eta}^i\,[\mathsf{act}', A] \right) \right) \cap \mathcal{T}_{A'} \right) \\ = & \bigcup_{\substack{\mathsf{act}' \in \\ \text{observes act}}} \bigcup_{A_0 \in \mathcal{A}, A_1 \in \mathcal{A}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0, A_1)} \left([\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\ \left(\bar{\eta}^i\,[u, A_0], \bar{\eta}^i\,[\mathsf{act}', A_1] \right) \right) \\ = & \bigcup_{A_0 \in \mathcal{A}, A_1 \in \mathcal{A}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0, A_1)} \bigcup_{\substack{\mathsf{act}' \in \\ \text{observes act}}} [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\,(\eta\,[u, A_0], \eta\,[\mathsf{act}', A_1]) \\ = & (\bar{\eta}^{i+1}\,[u', A'])^{\oplus} \end{array}$$

Since neither constraint system causes any side-effects here, if property (1) holds for the i-th approximations, and constraints corresponding to an observing action are considered, it also holds for the $(i+1)$-th approximation.

It thus remains to consider **creating** actions. The family of constraints in the refined constraint system for an edge $(u, x = \mathsf{create}(u_1), u') \in \mathcal{E}$ is then given by

$$[u', A'] \supseteq [\![([u, A_0], x = \mathsf{create}(u_1), u')]\!]_{\mathcal{LTA}} \qquad \text{for } A' \in [\![u, x = \mathsf{create}(u_1)]\!]^\sharp_{\mathcal{A}}(A_0)$$

with

$$\begin{array}{rcl} [\![([u, A_0], x = \mathsf{create}(u_1), u')]\!]_{\mathcal{LTA}}\,\eta & = & \textbf{let } T = [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\eta\,[u, A_0]) \textbf{ in} \\ & & (\{[u_1, \mathsf{new}^\sharp_{\mathcal{A}}\,u\,u_1\,A_0] \mapsto \mathsf{new}\,u_1\,(\eta\,[u, A_0])\}, T) \end{array}$$

where we once again denote by $\mathsf{new}^\sharp_{\mathcal{A}}\,u\,u_1\,A_0$ the single member of that set, which exists by (2.8) and (2.10). The constraints of the localized system are given by

$$[u'] \quad \supseteq \quad [\![(u, \mathsf{act}, u')]\!]_{\mathcal{LT}}$$

with

$$\begin{array}{rcl} [\![(u, x = \mathsf{create}(u_1), v)]\!]_{\mathcal{LT}}\,\eta & = & \textbf{let } T = [\![(u, x = \mathsf{create}(u_1), v)]\!]_{\mathcal{T}}(\eta\,[u]) \textbf{ in} \\ & & (\{[u_1] \mapsto \mathsf{new}\,u_1\,(\eta\,[u])\}, T) \end{array}$$

Let us denote by $(\eta^{i+1}\,[u'])^{\oplus}$ the new contribution to $[u']$ and by $(\bar{\eta}^{i+1}\,[u', A'])^{\oplus}$ the new contributions to $[u', A']$. Then by induction hypothesis and α total,

$$\begin{array}{rcl} (\eta^{i+1}\,[u'])^{\oplus} & = & [\![(u, x = \mathsf{create}(u_1), v)]\!]_{\mathcal{T}}(\eta^i\,[u]) \\ & = & [\![(u, x = \mathsf{create}(u_1), v)]\!]_{\mathcal{T}}\left(\bigcup_{A \in \mathcal{A}} \bar{\eta}^i\,[u, A] \right) \end{array}$$

and

$$(\bar{\eta}^{i+1}\,[u', A'])^{\oplus} \quad = \quad \bigcup_{A_0 \in \mathcal{A}, A' \in [\![u, x = \mathsf{create}(u_1)]\!]^\sharp_{\mathcal{A}}(A_0)} [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}\,(\bar{\eta}^i\,[u, A_0])$$

Thus,

$$(\eta^{i+1}\,[u'])^{\oplus} \cap \mathcal{T}_{A'} \quad = \quad (\bar{\eta}^{i+1}\,[u', A'])^{\oplus}$$

once again by the same reasoning as for the contribution to the left-hand side in case of local actions. Next, we consider the side-effects caused by such constraints: Let

us denote by $(\eta^{i+1}[u_1])^{\oplus}$ the new contribution to $[u_1]$ and by $(\bar{\eta}^{i+1}[u_1, A'])^{\oplus}$ the new contributions to $[u_1, A']$. Then, by induction hypothesis and α total,

$$(\eta^{i+1}[u_1])^{\oplus} = \mathsf{new}\, u_1\, (\eta^i[u]) = \mathsf{new}\, u_1\, \left(\bigcup_{A\in\mathcal{A}} \bar{\eta}^i[u, A]\right)$$

and

$$(\bar{\eta}^{i+1}[u_1, A'])^{\oplus} \;=\; \bigcup_{A\in\mathcal{A}, A'\in \mathsf{new}^\sharp_{\mathcal{A}}\ u\ u_1\ A} \mathsf{new}\, u_1\, (\bar{\eta}^i[u, A])$$

Thus

$$\begin{array}{lll}
& (\eta^{i+1}[u_1])^{\oplus} \cap \mathcal{T}_{A'} & \\
= & \mathsf{new}\, u_1\, \left(\bigcup_{A\in\mathcal{A}} (\bar{\eta}^i[u, A])\right) \cap \mathcal{T}_{A'} & \\
= & \left(\bigcup_{A\in\mathcal{A}} \mathsf{new}\, u_1\, (\bar{\eta}^i[u, A])\right) \cap \mathcal{T}_{A'} & \\
= & \left(\left(\bigcup_{A\in\mathcal{A}, A'\in \mathsf{new}^\sharp_{\mathcal{A}}\ u\ u_1\ A} \mathsf{new}\, u_1\, (\bar{\eta}^i[u, A])\right) \cap \mathcal{T}_{A'}\right) \cup & \\
& \left(\left(\bigcup_{A\in\mathcal{A}, A'\notin \mathsf{new}^\sharp_{\mathcal{A}}\ u\ u_1\ A} \mathsf{new}\, u_1\, (\bar{\eta}^i[u, A])\right) \cap \mathcal{T}_{A'}\right) & \\
= & \left(\left(\bigcup_{A\in\mathcal{A}, A'\in \mathsf{new}^\sharp_{\mathcal{A}}\ u\ u_1\ A} \mathsf{new}\, u_1\, (\bar{\eta}^i[u, A])\right) \cap \mathcal{T}_{A'}\right) \cup \emptyset & \text{(by (2.9))} \\
= & \bigcup_{A\in\mathcal{A}, A'\in \mathsf{new}^\sharp_{\mathcal{A}}\ u\ u_1\ A} \mathsf{new}\, u_1\, (\bar{\eta}^i[u, A]) & \text{(by (2.8))} \\
= & (\bar{\eta}^{i+1}[{}'_1, A'])^{\oplus} &
\end{array}$$

Therefore, if property (1) holds for the i-th approximations, and these constraints are considered, it also holds for the $(i+1)$-th approximation. This concludes the case distinction over the different actions. Thus, altogether, if property (1) holds for the i-th approximations, it also holds for the $(i+1)$-th approximation. □

Fig. 2.5 provides an overview over the different constraint systems for local traces, which were shown to be equivalent to each other, and the respective unknowns they employ.

Remark 1. *Given two admissible digests $\mathcal{A}^1$ and $\mathcal{A}^2$, we define the product digest $\mathcal{A}^1 \times \mathcal{A}^2$ with k-ary operations lifted as follows*

$$[\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}^1\times\mathcal{A}^2}\left((A^1_0, A^2_0), \ldots, (A^1_{k-1}, A^2_{k-1})\right) = \begin{cases} \left\{\left(\mathcal{A}^{1\prime}, \mathcal{A}^{2\prime}\right)\right\} & \text{for } [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}_1}(A^1_0, \ldots, A^1_{k-1}) = \left\{A^{1\prime}\right\} \\ & \text{and } [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}_2}(A^2_0, \ldots, A^2_{k-1}) = \left\{A^{2\prime}\right\} \\ \emptyset & \text{otherwise} \end{cases}$$

with $\mathsf{new}^\sharp_{\mathcal{A}}$ and $\mathsf{init}^\sharp_{\mathcal{A}}$ lifted point-wise to sets of tuples, and $\alpha_{\mathcal{A}}$ defined by the product of the individual abstraction functions. The resulting product digest then also is admissible.

Remark 2. *As a consequence of Eq.* (2.6)*, digests cannot track information arising from non-deterministic actions taken by the program, such as which value is returned for a non-deterministic assignment or input provided from the user. This limitation is only needed for technical reasons and can be lifted fairly easily at the expense of more intricate proofs.*

Name	Notation	Unknowns
Global Constraint System	$[\![\cdot]\!]_{\mathcal{T}}$	{ [T] }
Localized Constraint System	$[\![\cdot]\!]_{\mathcal{LT}}$	$\mathcal{N} \cup \mathcal{A}ct_{\mathsf{observable}}$
Refined Constraint System (for a given digest $\mathcal{A}$)	$[\![\cdot]\!]_{\mathcal{LTA}}$	$\{[u, A] \mid u \in \mathcal{N}, A \in \mathcal{A}\} \cup$ $\{[\mathsf{act}, A] \mid \mathsf{act} \in \mathcal{A}ct_{\mathsf{observable}}, A \in \mathcal{A}\}$

Figure 2.5: Constraint systems for the concrete semantics.

2.4 Considered Set of Actions

While the results from previous sections are quite generic in the set of actions supported by the programming language (except in examples), the concrete formalism for local traces, instances of digests, as well as analyses are naturally specific to the set of considered actions. For the analyses presented in this thesis, we consider a core subset of an imperative, C-like, programming language. In Chapter 5, we discuss how the implementation supports additional constructs, which do not form part of the core subset, and places where the semantics of C programs deviates from the idealized view taken here. In this section, we describe the set of actions supported by this imperative C-like language, detailing which are **local**, **observing**, **observable**, and **creating**. Fig. 2.6 provides a quick overview of these actions.

Thread Creation

As detailed in the previous section, the considered creating actions are $x = \mathsf{create}(u_1)$ for $x \in \mathcal{X}, u_1 \in \mathcal{N}$ where u_1 determines which thread template is used, and x received the thread *id* of the newly created thread. Thread creation is **creating**.

Initialization of the Environment

As outlined previously, we assume for technical reasons that there is an action initMT that is the first action performed by the main thread. This action can be understood to initialize the multi-threaded environment. This action is **observable**.

Assignments Involving Local Variables

There are actions corresponding to writing a local variable of the form $x = e$, where $x \in \mathcal{X}$ and the expression e is from some set $\mathcal{E}xp$ of expressions, which we assume contains accesses to local variables, some set of suitable arithmetic constants, arithmetic operations, and comparison operations. Additionally, we assume there is a special expression ? that represents a non-deterministic assignment. We do not detail the operations here, but only detail the assumptions we make: We assume all occurring expressions to be type-correct and that for all local writes the type of the variable x on

Creating	$x = \mathsf{create}(u_1)$	$x \in \mathcal{X}, u_1 \in \mathcal{N}$
Observable	initMT	
	$\mathsf{unlock}(a)$	$a \in \mathcal{M}$
	$\mathsf{signal}(s)$	$s \in \mathcal{S}$
	return	
Observing	$x = \mathsf{join}(x')$	$x \in \mathcal{X}, x' \in \mathcal{X}$
	$\mathsf{wait}(s)$	$s \in \mathcal{S}$
	$\mathsf{lock}(a)$	$a \in \mathcal{M}$
Local	$x = g$	$x \in \mathcal{X}, g \in \mathcal{G}$
	$g = x$	$x \in \mathcal{X}, g \in \mathcal{G}$
	$x = e$	$x \in \mathcal{X}, e \in \mathcal{E}xp$
	$\mathsf{Pos}(e)$	$e \in \mathcal{E}xp \setminus \{?\}$
	$\mathsf{Neg}(e)$	$e \in \mathcal{E}xp \setminus \{?\}$

Figure 2.6: Considered actions by type with observes relation indicated by arrows.

the left-hand side matches the type of the expression e on the right-hand side. Recall that, here, thread *id*s can only be copied from one variable to another and compared for equality — limiting the shape of expressions. We assume an evaluation function $[\![\cdot]\!]_{\mathcal{E}xp}$ is given so that for each local program state $\sigma : \Sigma$ and expression $e \not\equiv ?$, $[\![e]\!]_{\mathcal{E}xp}\,\sigma$ returns a value in $\mathcal{V}$. For convenience, we here encode the results of comparisons as integers where 0 denotes *false* and every non-zero value *true*. These actions are **local**.

Guards on Local Variables

The set of actions provides actions $\mathsf{Pos}(e)$ and $\mathsf{Neg}(e)$ for $e \in \mathcal{E}xp \setminus \{?\}$ of integer type. The intended semantics is that $[\![(u, \mathsf{Pos}(e), u')]\!]_{\mathcal{T}}$ yields a new local when e evaluates to a non-zero value, whereas $[\![(u, \mathsf{Neg}(e), u')]\!]_{\mathcal{T}}$ yields a new local trace when e evaluates to 0. The local program state stays unmodified in both cases. These actions are **local**.

Locking and Unlocking of Mutexes

Our language provides actions for locking and unlocking mutexes that ensure mutual exclusion. For simplicity, we only consider a fixed finite set $\mathcal{M}$ of mutexes. If, instead, a semantics with dynamically created mutexes were to be formalized, we could identify mutexes, e.g., via the local trace of the creating thread (as we did for threads). For each mutex $a \in \mathcal{M}$, the set of actions provides operations $\mathsf{lock}(a)$ and $\mathsf{unlock}(a)$. These operations are assumed to return no value, i.e., do always succeed. Additionally, we assume that $\mathsf{unlock}(a)$ for $a \in \mathcal{M}$ is only called by a thread currently holding the lock of a, and that mutexes are not re-entrant; i.e., trying to lock a mutex already held is undefined. Actions $\mathsf{unlock}(a)$ for $a \in \mathcal{M}$ are **observable**, actions $\mathsf{lock}(a)$ are **observing**,

with

$$\mathsf{observes}\,(\mathsf{lock}(a)) = \{\mathsf{initMT}, \mathsf{unlock}(a)\} \qquad \text{for } a \in \mathcal{M}$$

where the action initMT is observed whenever the ego thread is the first one to acquire the mutex a, and an action $\mathsf{unlock}(a)$ is observed otherwise.

Reading from and Writing to Globals

For reading from and writing to globals, we consider for $x \in \mathcal{X}$ and $g \in \mathcal{G}$ the actions $g = x$; (copy value of the local x into the global g) and $x = g$; (copy value of the global g into the local x) only. Thus, $g = g + 1$; for global g is not directly supported by our language but must be simulated by reading from g into a local, followed by incrementing the local and copying the resulting value back into g.

We assume for the concrete semantics that program execution is always *sequentially consistent*, and that both reads and writes to globals are *atomic*. The latter is enforced by introducing a dedicated mutex $m_g \in \mathcal{M}$ for each global g (sometimes referred to as the *atomicity mutex* for g) which is acquired before g is accessed and subsequently released. This means that each access `A` to g occurs as `lock(m_g); A; unlock(m_g);`. Under this proviso, the current value of each global g read by some thread can be determined just by inspection of the current local trace, and these operations are **local**.

Returning and Joining

For technical reasons, we assume that there is a special local variable $\mathsf{ret} \in \mathcal{X}$ that the returning thread assigns a value to and that the value of this variable is passed to the thread calling join. We introduce the following actions:

- return; terminates the current thread, and
- x=join(x'); where $x, x' \in \mathcal{X}$ and x' is a local variable holding a thread *id*; blocks the ego thread, until the thread with the given thread *id* has terminated. As in Pthreads, at most one thread may call join for a given thread *id*. The value of ret at the endpoint of the joined thread is then assigned to x.

The action return is **observable**, whereas joins are **observing** with

$$\mathsf{observes}\,(x = \mathsf{join}(x')) = \{\mathsf{return}\} \qquad \text{for } x, x' \in \mathcal{X}$$

Waiting and Signaling

Extending our language and semantics with waiting and signaling is not completely routine, partly because there is a variety of related concepts known by these names. Here, we follow more closely the line of *condition variables* as provided by Pthreads [24] and restrict ourselves to signaling and waiting only (broadcasting could also be handled, but would require a generalization of our notion of local traces). Deviating from the

semantics of PTHREADS, we do not consider spurious wakeups, i.e., a wait returning, despite the condition variable not being signaled.

Let $\mathcal{S}$ denote a finite set of condition variables. If instead, a semantics with dynamically created condition variables were to be formalized, we could once more identify condition variables by the local trace of the creating thread. The set of actions contains actions $\mathsf{signal}(s)$ and $\mathsf{wait}(s)$ for each $s \in \mathcal{S}$. Access to a condition variable $s \in \mathcal{S}$ should be protected by some mutex. Here, we model that by assuming that the program, instead of a single wait action (which would take a condition variable s and mutex m), always executes the sequence `unlock(m); wait(s); lock(m);` where the action $\mathsf{wait}(s)$ blocks until a signal for s is received. In that case, it proceeds with re-acquiring the protecting mutex m. In the local trace semantics, a thread executing the *wait* action for condition variable s thus expects a matching signal to have been executed after its $\mathsf{unlock}(m)$. We remark that, as in PTHREADS, it is not necessary to hold the mutex to call signal. Actions signal(s) are **observable**, whereas actions wait(s) are **observing** with

$$\mathsf{observes}\,(\mathsf{wait}(s)) = \{\mathsf{signal}(s)\} \qquad \text{for } s \in \mathcal{S}$$

We remark that to simulate a semantics that allows for spurious wakeups, a program may be modified to non-deterministically branch between a no-op and executing the wait, i.e., the sequence `unlock(m); x=?; if (x) { wait(s); } lock(m);` for a fresh local variable x instead of the sequence outlined above.

2.5 Formalism for Local Traces

Next, we give the formalism we use for local traces, instantiated to the actions from Section 2.4 — thereby completely specifying the semantics of $[\![\cdot]\!]_{\mathcal{T}}$, for which thus far only some requirements that need to be met have been presented. This concrete concurrency semantics imposes *restrictions* onto when binary actions are defined. In particular, a binary operation $[\![e]\!]_{\mathcal{T}}$ may only be defined for a pair of local traces t_0 and t_1 if certain parts of t_0 and t_1 represent the same computation.

We first turn to the requirements for a sequence of states for a single thread to be consistent within itself and later extend this to the interactions between different threads.

A *raw* trace of a single thread $i \in \mathcal{V}_{\mathbf{tid}}$ is a sequence $\lambda = \bar{u}_0 \mathsf{act}_1 \ldots \bar{u}_{n-1} \mathsf{act}_n \bar{u}_n$ for states $\bar{u}_j = (u_j, \sigma_j) \in \mathcal{N} \times \Sigma$ with $\sigma_j\,\mathsf{self} = i$, and actions $\mathsf{act}_j \in \mathcal{A}ct$ corresponding to the local state transitions of the thread i starting in the configuration $\bar{u}_0$ and executing actions act_j. In the sequence, every action $\mathsf{lock}(m)$, $\mathsf{wait}(s)$, and $x = \mathsf{join}(x')$ is assumed to succeed, and when accessing a global g, any value may be read. The same applies to the return value of an action $x{=}\mathsf{join}(x')$. In particular, this means that for every substring $(u_i, \sigma_i)\mathsf{act}_{i+1}(u_{i+1}, \sigma_{i+1})$ appearing in λ, there is a corresponding edge $(u_i, \mathsf{act}_{i+1}, u_{i+1}) \in \mathcal{E}$. Furthermore, there are requirements on local states depending on the action act_{i+1}:

(S1) If it is observable ($\mathsf{act}_{i+1} \in \mathcal{A}ct_{\mathsf{observable}}$), a write to a global ($\mathsf{act}_{i+1} \in \{g = x \mid g \in \mathcal{G}, x \in \mathcal{X}\}$), a lock operation ($\mathsf{act}_{i+1} \in \{\mathsf{lock}(a) \mid a \in \mathcal{M}\}$), a wait operation

$(\mathsf{act}_{i+1} \in \{\mathsf{wait}(s) \mid s \in \mathcal{S}\})$, or a guard $(\mathsf{act}_{i+1} \in \{\mathsf{Pos}(e) \mid e \in \mathcal{E}xp \setminus \{?\}\} \cup \{\mathsf{Neg}(e) \mid e \in \mathcal{E}xp \setminus \{?\}\})$, then $\sigma_i = \sigma_{i+1}$, i.e, the local state is unmodified.

(S2) If $\mathsf{act}_{i+1} \equiv \mathsf{Pos}(e)$, then $[\![e]\!]_{\mathcal{E}xp}\,\sigma_i \neq 0$; and if $\mathsf{act}_{i+1} = \mathsf{Neg}(e)$, then $[\![e]\!]_{\mathcal{E}xp}\,\sigma_i = 0$, i.e., the edge corresponding to the guard is only taken when the guard holds.

(S3) If $\mathsf{act}_{i+1} \equiv (x = e)$, and $e \not\equiv ?$, then $\sigma_{i+1} = \sigma_i \oplus \{x \mapsto [\![e]\!]_{\mathcal{E}xp}\,\sigma_i\}$ where $\oplus$ denotes destructive updates of a map, i.e., the local states agree on the values of all variables except for the one that is assigned, and the assigned value is consistent with the local state. For $e \equiv ?$, we have $\sigma_{i+1} \in \{\sigma_i \oplus \{x \mapsto v\} \mid v \in \mathcal{V}_\tau\}$ where τ corresponds to the type of x.

(S4) If $\mathsf{act}_{i+1} \equiv (x = g)$ or $\mathsf{act}_{i+1} = (x = \mathsf{join}(x'))$, we have once more $\sigma_{i+1} \in \{\sigma_i \oplus \{x \mapsto v\} \mid v \in \mathcal{V}_\tau\}$, i.e., the local states agree on all values except those that are modified.

One can view λ as an acyclic graph whose nodes are the 3-tuples $(j, u_j, \sigma_j), j = 0, \ldots, n$, and whose edges are $((j-1, u_{j-1}, \sigma_{j-1}), \mathsf{act}_j, (j, u_j, \sigma_j)), j = 1, \ldots, n$. Let $V(\lambda)$ and $E(\lambda)$ denote the set of nodes and edges of this graph, respectively. Let $\Lambda(i)$ denote the set of all individual raw traces for thread i, and Λ the union of all these sets.

A *raw global trace* is an acyclic graph $\tau = (\mathbf{V}, \mathbf{E})$ where $\mathbf{V} = \bigcup\{V(\lambda_i) \mid i \in I\}$ and $\mathbf{E} = \bigcup\{E(\lambda_i) \mid i \in I\}$ for a set I of thread *id*s and raw local traces $\lambda_i \in \Lambda(i)$. We demand that $i_0 \in I$, i.e., each raw global trace contains a raw local trace for the main thread. We remark that due to containing thread *id*s in $\sigma\,\mathsf{self}$, the nodes are unique, even when the numbering is applied per thread. On the set $\mathbf{V}$, we define the *program order* as the set of all pairs $\bar{u} \to_p \bar{u}'$ for which there is an edge $(\bar{u}, \mathsf{act}, \bar{u}')$ in $\mathbf{E}$. To formalize our notion of local traces, we extend the program order to a *causality order*, which additionally takes the order in which threads are created and joined, as well as the order in which mutexes are acquired and released, and the order in which waits and signals are executed into account.

For $a \in \mathcal{M}$, let $a^+ \subseteq \mathbf{V}$ denote the set of nodes $\bar{u}$ where an incoming edge labeled $\mathsf{lock}(a)$ exists, i.e., $\exists x\,(x, \mathsf{lock}(a), \bar{u}) \in \mathbf{E}$, and a^- analogously for $\mathsf{unlock}(a)$. We require that every mutex that is unlocked has been locked before by the same thread, and thus demand that for each $\bar{u} \in a^-$ appearing in the local trace there is a $\bar{u}' \in a^+$ such that $\bar{u}' \to_p^\star \bar{u}$ (where $\to_p^\star$ denotes the transitive closure of $\to_p$) and there is no unlock of a between the two actions, i.e. $\nexists \bar{u}'' \in a^- : \bar{u}' \to_p^\star \bar{u}'' \to_p^\star \bar{u}$.

By the same token, let s^+ the set of nodes with an incoming edge $\mathsf{signal}(s)$ and s^- those with an incoming edge $\mathsf{wait}(s)$. Analogously, let J and R denote the sets of nodes in $\mathbf{V}$ having an incoming edge labeled $x{=}\mathsf{join}(x')$, and return, respectively, for any local variables x, x'. On the other hand, let C denote the set of nodes with an *outgoing* edge labeled $x{=}\mathsf{create}(u_1)$ (for any local variable x and program point u_1). Let S denote the set of minimal nodes w.r.t. $\to_p^\star$, i.e., the points at which threads start and let $\mathbf{0}$ the node $(0, u_0, \sigma_0)$ where $\sigma_0\,\mathsf{self} = i_0$. Recall that, by convention, the initial thread executes the action initMT before any other actions. Let $\mathbf{1}$ denote the node with an incoming edge corresponding to initMT.

A *global trace* t then is represented by a tuple $(\tau, \rightarrow_c, \rightarrow_j, (\rightarrow_a)_{a\in\mathcal{M}}, (\rightarrow_s)_{s\in\mathcal{S}})$ where τ is a raw global trace and the relations $\rightarrow_c$, $\rightarrow_j$, $\rightarrow_a$ ($a \in \mathcal{M}$), $\rightarrow_s$ ($s \in \mathcal{S}$) are called the create, join, locking, and signaling orders and record additional dependencies between the different thread configurations. The *causality order* $\leq$ of t then is defined as the reflexive and transitive closure of the union

$$\rightarrow_p \cup \rightarrow_c \cup \rightarrow_j \cup \bigcup_{a\in\mathcal{M}} \rightarrow_a \cup \bigcup_{s\in\mathcal{S}} \rightarrow_s$$

These orders need to satisfy the following properties:

- **Causality order** $\leq$ is a partial order with unique least element $\mathbf{0}$;
- **Create order**: $\rightarrow_c \subseteq C \times (S \setminus \{\mathbf{0}\})$: $\forall s \in (S \setminus \{\mathbf{0}\}) : |\{z \mid z \rightarrow_c s\}| = 1$, that is, every thread except the initial thread is created by exactly one appearance of a $\mathsf{create}(...)$ action in the trace and $\forall x \in C : |\{z \mid x \rightarrow_c z\}| \leq 1$, i.e., each appearance of a $\mathsf{create}(...)$ action in the trace creates at most one thread. Additionally, for $((j-1, u_{j-1}, \sigma_{j-1}), x{=}\mathsf{create}(v), (j, u_j, \sigma_j)) \in \mathbf{E}$ and $(j-1, u_{j-1}, \sigma_{j-1}) \rightarrow_c (0, v, \sigma'_0)$, we have $\sigma'_0 = \sigma_{j-1} \oplus \{\mathsf{self} \mapsto i'\}$ for some thread *id* i' where $\sigma_j\, x = i'$, i.e., the creating and the created thread agree on the thread *id* of the created thread and the value of all locals is copied over. It remains to tie the thread *id* i' to the result of calling the function ν used to compute new thread *id*s: Let us refer to the node $(j-1, u_{j-1}, \sigma_{j-1})$ by c. Consider the subgraph of t that is induced by the set of all nodes x, s.t. $x \leq c$, i.e., the maximal subgraph with c as the maximal element w.r.t. the causality order $\leq$. We denote this sub-graph by $\downarrow_c (t)$, and demand that it is a local trace and that $i' = \nu(\downarrow_c (t))$.
- **Join order**: $\rightarrow_j \subseteq R \times J$: $\forall j' \in J : |\{z \mid z \rightarrow_j j'\}| = 1$ and $\forall r' \in R : |\{z | r' \rightarrow_j z\}| \leq 1$, that is, each join action in the traces joins exactly one thread also appearing in the trace and each thread is joined at most once. Additionally, for $((j-1, u_{j-1}, \sigma_{j-1}), x{=}\mathsf{join}(x'), (j, u_j, \sigma_j)) \in \mathbf{E}$ and $((j'-1, u_{j'-1}, \sigma'_{j'-1}),$ $\mathsf{return}, (j', u_{j'}, \sigma'_{j'})) \in \mathbf{E}$ and $(j', u_{j'}, \sigma'_{j'}) \rightarrow_j (j, u_j, \sigma_j)$: $\sigma_{j-1}\, x' = \sigma'_{j'-1}\, \mathsf{self}$, and $\sigma_j\, x = \sigma'_{j'-1}\, \mathsf{ret}$, i.e., the thread that is being joined has the thread *id* stored in x' and after the join, the return value is assigned to x;
- **Locking order**: $\forall a \in \mathcal{M} : \rightarrow_a \subseteq (a^- \cup \mathbf{1}) \times a^+$: $\forall x \in (a^- \cup \mathbf{1}) : |\{z \mid x \rightarrow_a z\}| \leq 1$ and $\forall y \in a^+ : |\{z \mid z \rightarrow_a y\}| = 1$, that is, for a mutex a every lock is preceded by exactly one unlock (or it is the first lock) of a, and each unlock is directly followed by at most one lock. A consequence of defining the lock order in this way is that mutexes are not re-entrant, as we have laid out in the exposition.
- **Signaling order**: $\forall s \in \mathcal{S} : \rightarrow_s \subseteq (s^+ \times s^-)$: $\forall x \in s^+ : |\{z \mid x \rightarrow_s z\}| \leq 1$ and $\forall y \in s^- : |\{z \mid z \rightarrow_s y\}| = 1$, i.e., for a condition variable s, every wait is preceded by exactly one signal to s, and each signal to s directly precedes at most one wait on s. Furthermore, the $\mathsf{signal}(s)$ needs to happen concurrently to the $\mathsf{wait}(s)$ for the

signal not to be lost. Thus, consider for each $s^+ \rightarrow_s s^-$ the immediate predecessor w of s^- w.r.t. program order (where $w \rightarrow_p s^-$). The edge corresponding to the action $\mathsf{wait}(s)$ then originates at w. s^+ must then not appear in the maximal subgraph of t with w as the maximal element w.r.t. the causality order $\leq$. We thus demand that $s^+ \notin \downarrow_w (t)$.

Additionally, we require a consistency condition on values read from globals, corresponding to our assumption of sequential consistency (see Section 2.4). For $((j-1, u_{j-1}, \sigma_{j-1}), x = g, (j, u_j, \sigma_j)) \in \mathbf{E}$, there is a *unique* maximal node $(j', u_{j'}, \sigma_{j'})$ with respect to $\leq$ such that either $((j'-1, u_{j'-1}, \sigma_{j'-1}), g = y, (j', u_{j'}, \sigma_{j'})) \in \mathbf{E}$ or $((j'-1, u_{j'-1}, \sigma_{j'-1}), \mathsf{initMT}, (j', u_{j'}, \sigma_{j'})) \in \mathbf{E}$ and $(j', u_{j'}, \sigma_{j'}) \leq (j-1, u_{j-1}, \sigma_{j-1})$. The uniqueness of such a maximal node is, in fact, already ensured by $\rightarrow_{m_g}$, as every write to a global g is immediately succeeded by an $\mathsf{unlock}(m_g)$ operation and every read of a global g is immediately preceded by a $\mathsf{lock}(m_g)$ operation. Then, in case the corresponding action is a write to the global, $\sigma_j\, x = \sigma_{j'-1}\, y$, i.e., the value read for the global is the last value written to it. Alternatively, if the corresponding action is initMT, then $\sigma_j\, x = 0$ as all globals initially have the value 0.

A global trace t is *local* if it has a unique maximal element $\bar{u} = (j, u, \sigma)$ (w.r.t. $\leq$). The functions sink and last as required for a local trace formalism (see Section 2.1) are then defined to return the tuple (u, σ), respectively the action act along the edge from the program order predecessor of $\bar{u}$ (if there is any) and $\bot$ otherwise. For a local trace using our formalism, we denote by $\mathsf{last\_write}_g : \mathcal{T} \rightarrow \mathbf{E}$ a function to extract the maximal write to a global if one appears in the local trace, and the initMT call otherwise as defined above. We also remark that for each node n appearing in a local trace, $\downarrow_n (t)$ is once again a local trace with sink node n.

The partial functions $\mathsf{new}\, u_1\, t$ for program point u_1 and $[\![e]\!]_{\mathcal{T}}$ for control-flow edges e then are defined by extending given local traces appropriately. On top of this intuitive definition, we outline the technical definition, verifying that requirements $(1) - (3)$ from Fig. 2.4 hold in the process.

For **observable** and **local** actions, an argument local trace t with maximal element $\bar{u} = (j, u, \sigma)$ of t, and a control-flow edge (u', act, u'') the set of resulting local traces is empty whenever $u \neq u'$. Otherwise, the set of candidate traces is obtained by adding a node $\bar{u}'' = (j+1, u'', \sigma')$ to t as well as an edge $(\bar{u}, \mathsf{act}, \bar{u}'')$, corresponding to $\bar{u} \rightarrow_p \bar{u}''$ in the program order, where σ' is chosen such that the ego-lane fulfills the requirements (S1)–(S4) on local states. The set of resulting local traces is then given by filtering out those candidates that do not fulfill the requirements on the orders $\rightarrow_c, \rightarrow_j, (\rightarrow_a)_{a \in \mathcal{M}}, (\rightarrow_s)_{s \in \mathcal{S}}$ or the consistency condition on the values of globals read. We remark that in all resulting local traces, $\bar{u}''$ is the maximal element, thus fulfilling requirement (2) from Fig. 2.4.

For **observing** actions, and argument traces (t_0, t_1) where t_0 has the maximal element $\bar{u} = (j, u, \sigma)$, and a control-flow edge (u', act, u''), the set of resulting local traces is empty whenever $u \neq u'$. Similarly, the set of resulting traces is empty if the last

action of t_1 is not in $\mathsf{observes}\,\mathsf{act}$ (requirement (3) from Fig. 2.4). Otherwise, the set of candidate traces is obtained by taking the union of both t_0 and t_1 and adding a new node $\bar{u}'' = (j+1, u'', \sigma')$ as well as an edge $(\bar{u}, \mathsf{act}, \bar{u}'')$, corresponding to $\bar{u} \to_p \bar{u}''$ in the program order, where σ' is chosen such that the ego-lane fulfills the requirements (S1)–(S4) on local states. Additionally, depending on the action, a new element is added to the orders: For an action $\mathsf{lock}(a)$, $\bar{v} \to_a \bar{u}''$ is added, where $\bar{v}$ is the maximal element of t_1. Analogously, for $\mathsf{wait}(s)$, $\bar{v} \to_s \bar{u}''$ and for $x = \mathsf{join}(x')$, $\bar{v} \to_s \bar{u}''$ are added. The set of resulting local traces is then given by filtering out those candidates that do not fulfill the requirements on the orders $\to_c, \to_j, (\to_a)_{a\in\mathcal{M}}, (\to_s)_{s\in\mathcal{S}}$ or the consistency condition on the values of globals read, and then filtering out those global traces that are not local traces. We remark that in all resulting local traces (that have a maximal element by definition), because of the inserted edges, the maximal element is $\bar{u}''$, thus fulfilling requirement (2) from Fig. 2.4.

For a **creating** action and an edge $(u', x = \mathsf{create}(u_1), u'')$ and argument trace t with maximal element $\bar{u} = (j, u, \sigma)$, the set of resulting local traces is empty whenever $u \neq u'$. Otherwise, the set of candidate traces is obtained by adding a node $\bar{u}'' = (j+1, u'', \sigma')$ to t as well as an edge $(\bar{u}, \mathsf{act}, \bar{u}'')$, corresponding to $\bar{u} \to_p \bar{u}''$ in the program order, where σ' is chosen such that the ego-lane fulfills the requirements (S1)–(S4) on local states. The set of resulting local traces is then given by filtering out those candidates that do not fulfill the requirements on the orders $\to_c, \to_j, (\to_a)_{a\in\mathcal{M}}, (\to_s)_{s\in\mathcal{S}}$ or the consistency condition on the values of globals read, and then filtering out those global traces that are not local traces. We remark that in all resulting local traces, $\bar{u}''$ is the maximal element, thus fulfilling requirement (2) from Fig. 2.4.

For the function $\mathsf{new}\,u_1\,t$, where argument trace t has maximal element $\bar{u} = (j, u, \sigma)$, the set of resulting local traces is empty whenever there is no edge $(u, x = \mathsf{create}(u_1), u') \in \mathcal{E}$. Otherwise, a new node $\bar{u}_1 = (0, u_1, \sigma_1)$ is inserted where $\sigma_1 = \sigma \oplus \{\mathsf{self} \mapsto \nu(t)\}$. Additionally, $\bar{u} \to_c \bar{u}_1$ is added, making $\bar{u}_1$ the maximal element. Thus, the requirement Eq. (2.3) and those from Fig. 2.4 are fulfilled.

Proposition 12. *The presented formalism fulfills the requirements* (1)-(3) *outlined in Fig. 2.4, and Theorem 3 thus applies to the considered instance of the local trace semantics.*

2.6 Examples for Local Traces

Consider the program given in Fig. 2.7. Graphical representations of some of its local traces are given in Figs. 2.8 to 2.10.

Fig. 2.8 is a local trace of the thread $t1$ in which it is the first one to acquire the mutex a, then sets its local variable z, and finally releases the mutex again. The sink node of this local trace is highlighted in bold. Additionally, the create order is highlighted in blue, and the mutex order for mutex a is highlighted in red. We remark that the node **1** corresponds to the node $(1, u_1', \sigma_1')$ here. We assume that the σ_i and σ_i' in Fig. 2.8 are consistent mappings from the local variables $\{x, y, z, \mathsf{ret}\}$ to concrete values from $\mathcal{V}$.

```
main:                t1:                  t2:
 initMT;              lock(a);             lock(a);
 x = create(t2);      z = 1;               signal(s);
 y = create(t1);      unlock(a);           unlock(a);
 signal(s);           wait(s);             ret = 12;
 lock(m_g);           lock(a);             return;
 g = 1;               z = join(x);
 unlock(m_g);         unlock(a);
 z = 28;              lock(m_g);
 return;              g = 2;
                      unlock(m_g);
                      x = create(t2);
                      return;
```

Figure 2.7: Example program highlighting the supported features. For this program, execution begins at program point main, and x, y, z are local variables, whereas g is a global variable. To ensure atomicity, every access to the global g is protected by the mutex m_g, which we omit in the further examples. We remark that signaling the condition variable s happens while holding a in t_2, while this is not the case if the signal comes from the main thread.

For the more involved examples, we now omit the internal structure of the nodes and display them as circles. For all figures, we assume them to represent legal calculations.

Consider now the local trace given in Fig. 2.9. The graph highlights the create order, the join order, as well as the locking orders for mutexes m_g and a and the signaling order for condition variable s. The unique sink node of this local trace is highlighted in bold in the figure.

We remark that the order $\rightarrow_{m_g}$, together with the program order of the threads whenever they hold the mutex m_g, totally orders the accesses to the global variable g, in this case denoting that the access of the main thread comes before the access of the thread t_2. This is not the case for the signaling order: The two signal operations occurring in the local trace in Fig. 2.9 are unordered w.r.t. each other, not just when considering $\rightarrow_s$, but also when considering the entire causality order. This is a key difference between the signaling order and the mutex order.

We proceed by giving two example local traces of the program and highlighting how certain local traces can *not* be combined to yield new local traces as one of the consistency requirements is violated.

First, consider the local traces shown in Fig. 2.10. In the local trace (a) on the left, thread t_2 is the first thread to acquire a. After signaling s, t_2 unlocks a again, and thread t_1 acquires it. Finally, thread t_2 unlocks a. The local trace (b) on the right, on the other hand, represents a sub-computation of (a). Let t'_0 denote the local trace (a) and t'_1 denote the local trace (b). Then, when computing $[\![(u, \mathsf{wait}(s), u')]\!]_{\mathcal{LT}}(t'_0, t'_1)$, the *raw* global trace

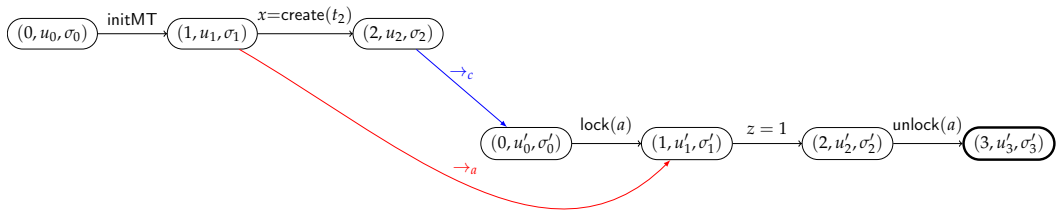


Figure 2.8: Local trace of the program in Fig. 2.7, detailing the node structure. Here, we assume that program points are labeled consecutively and that the first program point of the main thread is called u_0, whereas the first program point of t_1 is called u'_0.

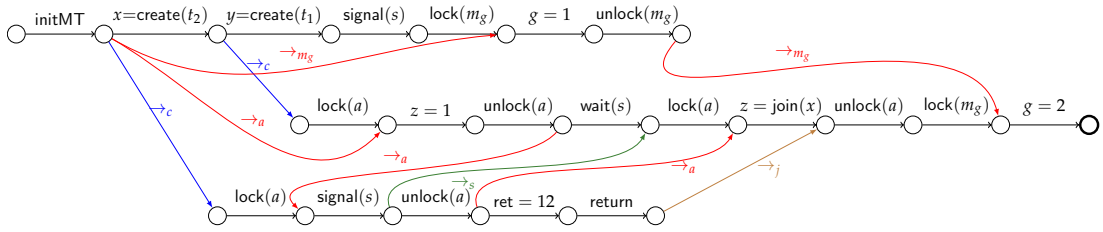
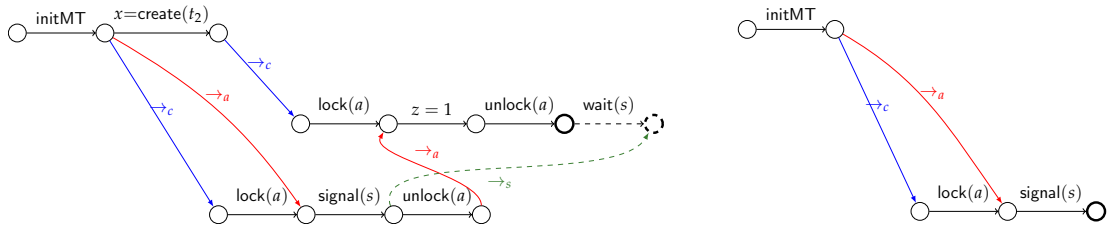


Figure 2.9: Local trace of the program in Fig. 2.7; For this program, execution begins at program point main, and x, y, z are local variables, whereas g is a global variable. To ensure atomicity, every access to the global g is protected by the mutex m_g, which we omit in the further examples.



(a) Local trace ending in unlock(a); Possible raw global trace resulting from combination with local trace on the right indicated in dashed lines.

(b) Local trace ending in the action signal(s).

Figure 2.10: Local traces of the program in Fig. 2.7 highlighting requirements on the signaling order.

shown in (a) when also considering dashed lines is one of the candidate traces. This raw global trace, however, is not a global trace and thus also not a local trace: The operations signal(s) and wait(s) are not unordered w.r.t. each other in the causality order $\leq$. Intuitively, the signal happens too early, as it is not concurrent with the wait in this execution, and can thus not cause thread t_1 to stop waiting.

2.7 Some Digests Abstracting Locking Histories

After having fixed a set of actions considered in our instance of the local trace semantics and having outlined the concrete formalism employed in previous sections, this section now provides some digests tailored to this set of considered actions. The examples in this section focus on digests that — in some sense — compute abstractions of locking histories.

As a first instance, consider digests representing the current lockset of the ego thread. The concept of computing sets of held locks for the ego thread at each program point is *not* novel. To the contrary, it is a standard technique widely employed for the static analysis of multi-threaded software, useful not just for the analysis of globals, but, e.g., also when trying to detect data races. We express this standard technique as a digest here. In this case, $\mathcal{A} = 2^{\mathcal{M}}$, and $\alpha_{\mathcal{A}}\, t$ is defined to consider the history of the ego thread only, and collect all mutexes a for which since the last call of lock(a), unlock(a) has not been called yet. The corresponding right-hand sides for inductively computing the digests are given in Fig. 2.11. We remark that, as a new thread always starts with an empty lockset, $\mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, S$ always returns the set containing the empty lockset. Other than that, the definition of $[\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}(S, S')$ is of particular interest: While it excludes the case that a mutex that is already held is locked again, note that it does not restrict the set of compatible traces, when the intersection $S' \cap S$ is non-empty. Even if the thread who unlocked the mutex a was holding some other mutex b at that time, and the thread now calling lock(a) currently holds b, that does not mean that they are incompatible

$$\mathsf{init}^\sharp_{\mathcal{A}} = \{\emptyset\}$$

$$\mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, S = \{\emptyset\}$$

$$[\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}\,(S, S') = \begin{cases} \emptyset & \text{if } a \in S \\ \{S \cup \{a\}\} & \text{otherwise} \end{cases}$$

$$[\![u, \mathsf{unlock}(a)]\!]^\sharp_{\mathcal{A}}\, S = \begin{cases} \emptyset & \text{if } a \notin S \\ \{S \setminus \{a\}\} & \text{otherwise} \end{cases}$$

$$[\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}\,(S, S') = \{S\} \text{ (other observing)}$$

$$[\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}\, S = \{S\} \text{ (other non-observing)}$$

Figure 2.11: Right-hand sides for expressing locksets as a digest.

— the thread unlocking a might also have unlocked b by now. The right-hand side for $[\![u, \mathsf{unlock}(a)]\!]^\sharp_{\mathcal{A}} S$ here is defined to exclude the case where a mutex that is not held is unlocked.

Refining an analysis according to this instance amounts to computing separate abstract values per lockset and program point, and keeping for every observable action the set of mutexes which were held after the corresponding action occurred. For unlock actions, we will later refer to this set of mutexes as the *background lockset*.

Proposition 13. *The lockset digest from Fig. 2.11 with $\alpha_{\mathcal{A}}(t)$ defined to inductively collect the set of currently held mutexes along the ego-lane of t is an admissible digest.*

Proof. Properties (2.6), (2.8), and (2.10) from Section 2.3 hold by construction. For (2.7), we consider the case of locking a mutex in detail:

$$\forall t_0, t_1 \in \mathcal{T} : \alpha_{\mathcal{A}}([\![(u, \mathsf{lock}(a), v)]\!]_{\mathcal{T}}(t_0, t_1)) \subseteq [\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}(\alpha_{\mathcal{A}}\, t_0, \alpha_{\mathcal{A}}\, t_1)$$

If the ego thread in t_0 already holds the mutex a, i.e., $a \in \alpha_{\mathcal{A}}(t_0)$, $[\![(u, \mathsf{lock}(a), v)]\!]_{\mathcal{T}}(t_0, t_1)$ yields an empty set, and nothing is to be shown. In case $a \notin S = \alpha_{\mathcal{A}}(t_0)$, if there is a resulting trace $t' \in [\![(u, \mathsf{lock}(a), v)]\!]_{\mathcal{T}}(t_0, t_1)$, the ego thread in t' holds mutexes $S \cup \{a\}$, and thus $\alpha_{\mathcal{A}}(t') = S \cup \{a\}$. Thus, for an action corresponding to locking a mutex, the property holds. For unlock, the argument proceeds similarly.

For the other actions, we observe that, in the concrete semantics, the set of mutexes currently held by the ego-thread remains unchanged along any edge where the action does not correspond to locking or unlocking some mutex. The same holds for the digests. Thus, for any edge and action not corresponding to locking or unlocking mutexes and local traces $t_0 \dots t_k$ where $\alpha_{\mathcal{A}}(t_k) = S_k$ and $t' \in [\![e]\!]_{\mathcal{T}}(t_0, \dots, t_{k-1})$, we have $\alpha_{\mathcal{A}}(t') = S_0 \in \{S_0\} = [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(S_0, \dots, S_{k-1})$.

Lastly, properties (2.9) and (2.11) follow from the observation that new threads always start with the empty lockset. □

As such admissibility proofs are fairly mechanical in nature and mostly rely on an appropriately defined abstraction function $\alpha_{\mathcal{A}}$, we will in the subsequent chapters usually

$$\mathsf{init}^\sharp_{\mathcal{A}} = \{\emptyset\}$$
$$\mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, L = \{L\}$$
$$[\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}\,(L, L') = \begin{cases} \emptyset & \text{if } a \in L \wedge a \notin L' \\ \{L \cup L' \cup \{a\}\} & \text{otherwise} \end{cases}$$
$$[\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}\, L = \{L\} \text{ (other non-observing)}$$
$$[\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}\,(L, L') = \{L \cup L'\} \text{ (other observing)}$$

Figure 2.12: Right-hand sides for refining according to encountered lock operations.

not conduct (detailed) admissibility proofs for digests, and instead limit ourselves to providing the key intuition where necessary.

Remark 3. *Alternatively, one could give stronger rules for* $[\![u, \mathsf{wait}(s)]\!]^\sharp_{\mathcal{A}}(S, S')$ *and also for* $[\![u, x = \mathsf{join}(x')]\!]^\sharp_{\mathcal{A}}(S, S')$ *that always return* $\emptyset$ *whenever* $S \cap S' \neq \emptyset$*. For* $\mathsf{wait}(s)$*, this is the case as two operations* $\mathsf{signal}(s)$ *and* $\mathsf{wait}(s)$ *during both of which some common mutex* a *is held are always ordered w.r.t. each other and thus combining two such local traces never yields a valid new local trace. For* $\mathsf{join}(x')$*, this is the case as a mutex that the thread still holds at* return *is never released, and thus the thread calling* $\mathsf{join}(x')$ *cannot have acquired it since the corresponding thread returned.*

We do not make use of this stronger definition in the following to ensure that the lockset digest fits nicely into the framework of ego-lane digests (which will be defined in Section 2.9) and that the predicate $\mathsf{compat}^\sharp_{\mathcal{A}}$ *there need not distinguish between different actions.*

As a further instance, consider tracking which mutexes have been locked at least once in the local trace. In this case, we once again have $\mathcal{A} = 2^{\mathcal{M}}$. Now, at $\mathsf{lock}(a)$, local traces in which the ego thread knows some thread has performed a $\mathsf{lock}(a)$ cannot be combined with local traces that contain no $\mathsf{lock}(a)$. Additionally, a newly created thread knows about the lock events which have occurred in its creator and its initial digest thus comes from its parent. The corresponding right-hand sides are given in Fig. 2.12. This scheme naturally generalizes to counting how often some action (e.g., a write to a global g) occurred, stopping exact bookkeeping at a constant (1 in this case).

2.8 Thread *ID*s as a Digest

While, in the previous section, abstractions of the locking history were tracked as a digest, one can also track an abstraction of the encountered thread create actions. Here, we propose to abstract such creation histories in a way that leads to an analysis of abstract thread *id*s, and their uniqueness. Such *abstract* thread *id*s are a key ingredient when dealing with systems where dynamic thread creation is supported (which leads to a potentially unbounded number of threads), as is the case for programs in the core language considered in this thesis.

To obtain thread *id*s, we identify threads by their thread creation history, i.e., by sequences of *create* edges. As these sequences may grow arbitrarily, we collect all creates occurring after the first repetition into a *set* to obtain finite abstractions.

Example 6. *In the program from Fig. 2.13, the first thread created by* main *receives the abstract thread* id $(\mathit{main} \cdot \langle u_1, t_1\rangle, \emptyset)$. *It creates a thread with abstract thread* id $(\mathit{main} \cdot \langle u_1, t_1\rangle \cdot \langle u_3, t_1\rangle, \emptyset)$. *At program point* u_3, *the latter creates a thread starting at* t_1 *and receiving the abstract thread* id $(\mathit{main} \cdot \langle u_1, t_1\rangle, \{\langle u_3, t_1\rangle\})$ — *as do all threads subsequently created along this edge.* □

Create edges, however, may also be repeatedly encountered within the creating thread, namely in a loop. To deal with this, we track, for each thread, the set C of already encountered *create* edges. When a *create* edge is encountered again, the created thread receives a non-unique thread *id*.

Example 7. *The first time the main thread reaches program point* u_2 *in the program from Fig. 2.13, the created thread is assigned the* unique *abstract thread* id $(\mathit{main} \cdot \langle u_2, t_1\rangle, \emptyset)$. *In subsequent loop iterations, the created threads are no longer kept separate and thus receive the non-unique* id $(\mathit{main}, \{\langle u_2, t_1\rangle\})$. □

Formally, let $\mathcal{N}_C$ and $\mathcal{N}_S$ denote the subsets of program points with an outgoing edge labeled x=create(...), and of first program points of thread templates, respectively. Let $\mathcal{P} \subseteq \mathcal{N}_C \times \mathcal{N}_S$ denote the set of pairs relating thread creation nodes with the starting points of the created threads. The set $\mathcal{V}^{\sharp}_{\mathbf{tid},\mathcal{A}}$ of abstract thread *id*s then consists of all pairs $(i, s) \in (\mathsf{main} \cdot \mathcal{P}^*) \times 2^{\mathcal{P}}$ in which each pair $\langle u, f\rangle$ occurs at most once. The abstract thread *id* of the *main* thread is given by $(\mathsf{main}, \emptyset)$. Elements in $(\mathsf{main} \cdot \mathcal{P}^*) \times \{\emptyset\}$ represent the *unique* thread *id*s, i.e., abstract thread *id*s whose concretizations contain, for each local trace t, at most one concrete thread *id* appearing in t.[4] Elements (i, s) where $s \neq \emptyset$, on the other hand, are *ambiguous*, i.e., may represent multiple concrete thread *id*s appearing in some local trace t.

[4]This definition may seem a little unusual, but is a result of having non-determinism in the concrete semantics (several states in init and allowing assignments x =?), and using local traces as concrete thread *id*s. Uniqueness per local trace and thus per execution is sufficient here, as we do not attempt to relate different executions to each other.

```
main:
 x = g; // PP u1
 y = create(t1);
 for(i = 0; i < 5; i++) {
  // PP u2
  z = create(t1);
 }
```

```
t1:
 g = 42; // PP u3
 y = create(t1);
```

Figure 2.13: Program with multiple thread creations.

We consider not only abstract thread *id*s as part of the digest — but additionally track sets of executed thread creations within the current thread as they are needed to compute the thread *id*s of newly started threads. Accordingly, we set $\mathcal{A} = \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \times 2^{\mathcal{P}}$ and define the right-hand sides as shown in Fig. 2.14, where $\bar{i}$ denotes the set of pairs occurring in the sequence i. The function $\alpha_{\mathcal{A}}$ then is defined inductively in the intuitive manner.

Using $\alpha_{\mathcal{A}}$, it is possible to give a concretization function $\gamma_{\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}} : \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \to 2^{\mathcal{V}_{\mathbf{tid}}}$ from an abstract thread *id* to the set of concrete thread *id*s such that

$$\forall t \in \mathcal{T} : \alpha_{\mathcal{A}}\, t = (i^\sharp, \_) \implies \mathsf{id}\, t \in \gamma_{\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}}\, i^\sharp \tag{2.13}$$

holds. Assume that we have identified the thread *id*s with thread creation histories, i.e., local traces. In this case, the concretization can, e.g., be given by

$$\gamma_{\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}}\, i^\sharp = \{\downarrow_w (t) \mid t \in \mathcal{T}, (i^\sharp, \_) = \alpha_{\mathcal{A}}\, t, \mathsf{last}\, t = \bot, \exists w : w \to_c (0, \mathsf{sink}\, t)\}$$

where, by abuse of notation, we denote by $(0, \mathsf{sink}\, t)$ its flattening into a 3-tuple. That is, the concretization is given by the set of local traces ending in an action creating a thread for which the sequence of create actions *in the created thread* agrees with the one recorded in $\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}$.

We remark that the concretizations of different $\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}$ are thus disjoint, and require that this property holds for the chosen concrete thread *id* domain $\mathcal{V}_{\mathbf{tid}}$ and the concretization function $\gamma_{\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}}$:

$$\forall i^\sharp_0, i^\sharp_1 \in \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} : i^\sharp_0 \neq i^\sharp_1 \implies \gamma_{\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}}\, i^\sharp_0 \cap \gamma_{\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}}\, i^\sharp_1 = \emptyset \tag{2.14}$$

Example 8. *Consider again the program from Fig. 2.13 with right-hand sides from Fig. 2.14, and assume that the right-hand side for observing actions returns the set containing its first argument. The initial thread has the abstract thread* id $i_0 = (\mathsf{main}, \emptyset)$*. At its start point, the digest thus is* $(i_0, \emptyset)$*. At the create edge originating at* u_1*, a new thread with* id $(\mathsf{main} \cdot \langle u_1, t_1\rangle, \emptyset)$ *is created.*

$$\mathsf{init}^\sharp_{\mathcal{A}} = \{((\mathsf{main}, \emptyset), \emptyset)\}$$
$$[\![u, x{=}\mathsf{create}(u_1)]\!]^\sharp_{\mathcal{A}}\,(i, C) = \{(i, C \cup \{\langle u, u_1\rangle\})\}$$
$$[\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}\,(i, C) = \{(i, C)\} \quad \text{(other non-observing)}$$
$$\mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, ((d, s), C) =$$
$$\quad \textbf{let } (d', s') = (d, s) \circ \langle u, u_1\rangle \textbf{ in}$$
$$\quad \textbf{if } s' = \emptyset \wedge \langle u, u_1\rangle \in C \textbf{ then } ((d, \{\langle u, u_1\rangle\}), \emptyset)$$
$$\quad \textbf{else } ((d', s'), \emptyset)$$

$$(d, s) \circ \langle u, u_1\rangle =$$
$$\quad \textbf{if } d = (d_0 \cdot \langle u, u_1\rangle) \cdot d_1 \textbf{ then}$$
$$\quad\quad (d_0, s \cup \bar{d_1} \cup \{\langle u, u_1\rangle\})$$
$$\quad \textbf{else if } s = \emptyset \textbf{ then } (d \cdot \langle u, u_1\rangle, \emptyset)$$
$$\quad \textbf{else } (d, s \cup \{\langle u, u_1\rangle\})$$

Figure 2.14: Right-hand sides for thread *id*s.

The digest for this thread then is $((\mathit{main} \cdot \langle u_1, t_1\rangle, \emptyset), \emptyset)$*. For the main thread, the encountered create edge* $\langle u_1, t_1\rangle$ *is added to the second component of the digest, making it* $(i_0, \{\langle u_1, t_1\rangle\})$*.*

When u_2 *is reached with* $(i_0, \{\langle u_1, t_1\rangle\})$*, a unique thread with* id $(\mathit{main} \cdot \langle u_2, t_1\rangle, \emptyset)$ *is created. The new digest of the creating thread then is* $(i_0, \{\langle u_1, t_1\rangle, \langle u_2, t_1\rangle\})$*. In subsequent iterations of the loop, for which* u_2 *is reached with* $(i_0, \{\langle u_1, t_1\rangle, \langle u_2, t_1\rangle\})$*, a non-unique thread with* id $(\mathit{main}, \{\langle u_2, t_1\rangle\})$ *is created.*

When reaching u_3 *with* id $(\mathit{main}, \{\langle u_2, t_1\rangle\})$*, a thread with* id $(\mathit{main}, \{\langle u_2, t_1\rangle, \langle u_3, t_1\rangle\})$ *is created as the* id *of the creating thread was already not unique. When reaching it with the* id $(\mathit{main} \cdot \langle u_1, t_1\rangle, \emptyset)$*, a new thread with* id $(\mathit{main} \cdot \langle u_1, t_1\rangle \cdot \langle u_3, t_1\rangle, \emptyset)$ *is created. When the newly created thread reaches this program point, the threads created there have the* non-unique id $(\mathit{main} \cdot \langle u_1, t_1\rangle, \{\langle u_3, t_1\rangle\})$*, as* $\langle u_3, t_1\rangle$ *already appears in the* id *of the creating thread.* □

Abstract thread *id*s should provide us with functions

- $\mathsf{unique} : \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \to \mathbf{bool}$ tells whether a thread *id* is unique.
- $\mathsf{lcu\_anc} : \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \to \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \to \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}$ returns the last common *unique* ancestor of two threads.
- $\mathsf{may\_create} : \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \to \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \to \mathbf{bool}$ checks whether a thread *may* (transitively) create another.

For our domain $\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}$, these can be defined as $\mathsf{unique}\,(i,s) = (s = \emptyset)$ and

$$\begin{array}{lcl} \mathsf{lcu\_anc}\,(i,s)\,(i',s') & = & (\text{longest common prefix } i\,i', \emptyset) \\ \mathsf{may\_create}\,(i,s)\,(i',s') & = & (\bar{i} \cup s) \subseteq (\bar{i'} \cup s') \end{array}$$

We use these predicates to enhance the definitions of $[\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}$, $[\![u, x'{=}\mathsf{join}(x)]\!]^\sharp_{\mathcal{A}}$, and $[\![u, \mathsf{wait}(s)]\!]^\sharp_{\mathcal{A}}$ to take into account that the ego thread cannot acquire a mutex from another thread, (successfully) call join for it, or receive a signal from it, if this other thread has definitely not yet been created. For a thread with thread *id* i', it is known that it has not yet been created if

(1) it is directly created by the *unique* ego thread, but the ego thread has not yet reached the program point where i' is created;

(2) its thread *id* indicates that a thread that has not yet been created according to (1), is part of the creation history of i'.

Accordingly, we introduce the predicate $\mathsf{may\_run}\,(i,C)\,(i',C')$ and define it by

$$(\mathsf{lcu\_anc}\,i\,i' = i) \implies \exists \langle u, u'\rangle \in C : (i{\circ}\langle u, u'\rangle = i' \vee \mathsf{may\_create}\,(i{\circ}\langle u, u'\rangle)\,i')$$

It is false whenever thread i' is definitely not yet started. We then set

$$\begin{array}{lcl} [\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}\,(i,C)\,(i',C') & = & [\![u, x'{=}\mathsf{join}(x)]\!]^\sharp_{\mathcal{A}}\,(i,C)\,(i',C') \\ & = & [\![u, \mathsf{wait}(s)]\!]^\sharp_{\mathcal{A}}\,(i,C)\,(i',C') \\ & = & \begin{cases} \{(i,C)\} & \text{if } \mathsf{may\_run}\,(i,C)\,(i',C') \\ \emptyset & \text{otherwise} \end{cases} \end{array} \tag{2.15}$$

This analysis of thread *id*s and uniqueness can be considered as some form of *May-Happen-In-Parallel* (or, more precisely, *Must-Not-Happen-In-Parallel*) analysis.

Remark 4. *MHP information is useful in a variety of scenarios: For example, a thread-modular analysis of* data races *or* deadlocks *that does not consider thread* ids *can be refined with this digest to exclude more data races or deadlocks. To provide a semantic footing for such an analysis, first the notion of accesses only ordered by atomicity mutexes in the local trace semantics would need to be tied to existing notions of data races.*

Additionally, abstract thread ids *may be combined with an allocation-site abstraction [26] and a uniqueness analysis for allocation sites in single-threaded programs to also identify heap locations in multi-threaded programs as unique, which under some circumstances may allow for* strong *updates.*

Another way in which thread ids *can be leveraged is to lift an analysis of single-threaded* use-after-free *bugs to a multi-threaded setting. To this end, all thread* id *information of threads calling* `free` *for a certain heap location is recorded. If, based on this information and the thread* id *information of a thread accessing the memory, it can be excluded that the call to free has already happened, no* use-after-free *bug can occur at this location. This is the approach taken by the* Goblint *analyzer for detecting across-thread* use-after-free *bugs [103].*

In the context of this thesis, we will employ this thread *id* digest in later chapters to improve the precision of thread-modular analyses of globals.

2.9 Ego-Lane Digests

We observe that the thread *id* digest presented in the previous section can be computed just in terms of the history of the ego thread and its parents up to the creation point. This will render it applicable also for analyses that use a constraint system of a slightly different form than the one outlined previously which is used by the concrete semantics and some analyses. More formally, let us call a digest $\mathcal{A}$ *ego-lane-derived* (or more concisely an *ego-lane digest*), if it fulfills the following property

$$\begin{aligned} &\forall A_0 \in \mathcal{A} : \exists A' \in \mathcal{A} : \forall A_1 \in \mathcal{A} : [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0, A_1) \in \{\{A'\}, \emptyset\} \\ &\quad (\text{for } (u, \mathsf{act}, v) \in \mathcal{E}, \mathsf{act} \in \mathcal{A}ct_{\mathsf{observing}}) \end{aligned} \tag{2.16}$$

Intuitively, this property ensures that digest information (up to emptiness) can be computed as a function of only the actions on the ego lane and the ego lanes of parent threads (transitively) creating the ego thread up to the point where the ego thread (or one of its parent) is created. Let us call this linear sequence the *creation-extended ego lane* of this thread. Analyses will later make use of this property to incorporate information at points that are different from the ones where further local traces get incorporated in the concrete semantics.

To ensure that such exclusions can be made soundly, it needs to be guaranteed that, when incorporation is delayed when compared to the concrete semantics, the digests at such a later point do still allow for the incorporation whenever the original digests

did at the original point. Thus, we replace the normal compatibility check performed for binary actions by checks via some function $\mathsf{compat}^\sharp_{\mathcal{A}} : \mathcal{A} \to \mathcal{A} \to \textbf{bool}$ required to satisfy

$$\begin{aligned}
&[\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0, A_1) \neq \emptyset \implies \mathsf{compat}^\sharp_{\mathcal{A}}(A_0, A_1) \\
&\quad (\text{for } (u, \mathsf{act}, v) \in \mathcal{E}, \mathsf{act} \in \mathcal{A}ct_{\mathsf{observing}}) \\
&\mathsf{compat}^\sharp_{\mathcal{A}}(A_0, A_1) \implies \forall A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0) : \mathsf{compat}^\sharp_{\mathcal{A}}(A', A_1) \\
&\quad (\text{for } (u, \mathsf{act}, v) \in \mathcal{E}, \mathsf{act} \in (\mathcal{A}ct_{\mathsf{observable}} \cup \mathcal{A}ct_{\mathsf{creating}} \cup \mathcal{A}ct_{\mathsf{local}})) \\
&\mathsf{compat}^\sharp_{\mathcal{A}}(A_0, A_1) \implies \forall A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0, A_2) : \mathsf{compat}^\sharp_{\mathcal{A}}(A', A_1) \\
&\quad (\text{for } (u, \mathsf{act}, v) \in \mathcal{E}, \mathsf{act} \in \mathcal{A}ct_{\mathsf{observing}}) \\
&\mathsf{compat}^\sharp_{\mathcal{A}}(A_0, A_1) \implies \forall A' \in \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0 : \mathsf{compat}^\sharp_{\mathcal{A}}(A', A_1) \\
&\quad (\text{for } (u, x = \mathsf{create}(u_1), v) \in \mathcal{E})
\end{aligned} \tag{2.17}$$

Furthermore, we require some backwards compatibility property of $\mathsf{compat}^\sharp_{\mathcal{A}}$: If some digest A_1 for another thread is compatible with the current digest A_0, so are any digests of the other thread that are potential predecessor digests to A_1.

$$\begin{aligned}
&\mathsf{compat}^\sharp_{\mathcal{A}}(A_0, A_1) \wedge A_1 \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_2) \implies \mathsf{compat}^\sharp_{\mathcal{A}}(A_0, A_2) \\
&\quad (\text{for } (u, \mathsf{act}, v) \in \mathcal{E}, \mathsf{act} \in (\mathcal{A}ct_{\mathsf{observable}} \cup \mathcal{A}ct_{\mathsf{creating}} \cup \mathcal{A}ct_{\mathsf{local}})) \\
&\mathsf{compat}^\sharp_{\mathcal{A}}(A_0, A_1) \wedge \exists A' \in \mathcal{A} : A_1 \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_2, A') \implies \mathsf{compat}^\sharp_{\mathcal{A}}(A_0, A_2) \\
&\quad (\text{for } (u, \mathsf{act}, v) \in \mathcal{E}, \mathsf{act} \in \mathcal{A}ct_{\mathsf{observing}})
\end{aligned} \tag{2.18}$$

Lastly, for technical reasons, we require that initial digests be compatible with all other digests.

$$\forall A \in \mathcal{A}, \forall A' \in \mathsf{init}^\sharp_{\mathcal{A}} : \mathsf{compat}^\sharp_{\mathcal{A}}(A, A') \tag{2.19}$$

Example 9. *The digest to compute locksets (Fig. 2.11) is ego-lane-derived. The only possible definition for* $\mathsf{compat}^\sharp_{\mathcal{A}}$ *here is the function always returning* True*. On the other hand, the digest for refining according to encountered lock operations (Fig. 2.12) is* not *ego-lane-derived. Intuitively, this is because, at observing actions, parts of the history of the other threads get incorporated, and the resulting digests may differ based on the digest of the other thread. This violates Eq.* (2.16). □

Example 10. *The thread* id *digests with right-hand sides from Fig. 2.14 and Eq.* (2.15) *are ego-lane-derived. Possible definitions for* $\mathsf{compat}^\sharp_{\mathcal{A}}$ *are, e.g., the function always returning* True*, or the more useful function* $\mathsf{compat}^\sharp_{\mathcal{A}}(i, C)(i, C') = \mathsf{may\_run}\,(i, C)(i, C')$*. The first definition will usually not yield any improvement in precision for analyses, whereas the second one will be able to exclude writes of threads that have definitely not been created yet (see, e.g., Example 16 in Section 4.1.2).* □

3 Abstract Domains & 2-Decomposability

In preparation for describing abstract-interpretation-based static analyses in the next chapter, this chapter outlines the requirements on the abstract value domains employed, and introduces the notation that will be used in the rest of this thesis — both for relational and non-relational domains. While many of the definitions in this chapter will seem familiar to the reader, there are two non-routine aspects to the setup considered here that justify giving this content its own chapter: Firstly, we couple together a relational and a non-relational abstract domain and provide functions for translating between both, which is not so common. Secondly, and perhaps more importantly, we also introduce the notion of *2-decomposability* of relational domains, which plays a key role for our relational analyses.

While this chapter re-uses the notation already employed in our two earlier works [107, 108], some definitions deviate from the ones used in either of the earlier works — primarily for technical reasons relating to the novel proofs this thesis provides.

3.1 Non-Relational Domains

Recall that we denote by $\mathcal{V}ars$ the set of all variables. We assume that we are given for each type τ of values, a complete lattice $\mathcal{V}^\sharp_\tau$ abstracting sets of concrete values from $\mathcal{V}_\tau$. We require that the concretization function $\gamma_{\mathcal{V}^\sharp_\tau} : \mathcal{V}^\sharp \to 2^{\mathcal{V}_\tau}$ is monotonic.

For values of type thread *id*, we impose the additional restriction that $\mathcal{V}^\sharp_{\mathbf{tid}}$ is given as the powerset domain over some finite carrier set $S_{\mathcal{V}^\sharp_{\mathbf{tid}}}$ with $\bot = \emptyset$, $\top = S_{\mathcal{V}^\sharp_{\mathbf{tid}}}$, and join and meet given by union and intersection, respectively. This restriction is needed to ensure monotonicity in the side-effects for constraints corresponding to thread returns in the analyses later on, and thus for proving the existence of *least* solutions for the abstract constraint systems corresponding to these analyses. We further demand that the concretization of an abstract value $i^\sharp$ of type thread *id* is given by the union of the concretizations of the constituent singleton sets in $S_{\mathcal{V}^\sharp_{\mathbf{tid}}}$, i.e., $\gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}\, i^\sharp = \bigcup_{i \in i^\sharp}(\gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}\, \{i\})$.

Let $\mathcal{V}^\sharp$ denote the collection of the lattices $\mathcal{V}^\sharp_\tau$. Then, let $\bar{\mathcal{V}}^\sharp = \mathcal{V}ars \to_\bot \mathcal{V}^\sharp$ denote the set of all type-consistent assignments σ from variables to abstract values, where all bindings for *local* variables, i.e., those variables in $\mathcal{X}$ are non-$\bot$, extended with a dedicated least element (also denoted by $\bot$) and equipped with the induced point-wise ordering. We define a monotonic concretization function $\gamma_{\mathcal{V}^\sharp} : \mathcal{V}^\sharp \to 2^{\mathcal{V}}$ by applying the concretization function of the respective type. We then lift $\gamma_{\mathcal{V}^\sharp}$ to obtain a monotonic

concretization function $\gamma_{\bar{\mathcal{V}}^\sharp} : \bar{\mathcal{V}}^\sharp \to 2^{\mathcal{V}ars \Rightarrow \mathcal{V}}$ to a set of *partial* maps:

$$\gamma_{\bar{\mathcal{V}}^\sharp}\, \sigma^\sharp = \{\sigma \mid \forall x \in \mathcal{X} : \sigma\, x \in \gamma_{\mathcal{V}^\sharp}(\sigma^\sharp\, x) \wedge \forall g \in \mathcal{G} : (g \in \mathsf{dom}(\sigma) \Rightarrow \sigma\, g \in \gamma_{\mathcal{V}^\sharp}(\sigma^\sharp\, g))\}$$

with $\gamma_{\bar{\mathcal{V}}^\sharp} \bot = \emptyset$. Alternatively, in some proofs, we will find it convenient to work with an intermediate concretization $\bar{\gamma}_{\bar{\mathcal{V}}^\sharp} : \bar{\mathcal{V}}^\sharp \Rightarrow (\mathcal{V}ars \to 2^{\mathcal{V}})$ to (total) maps from variables to sets of concrete values where

$$\bar{\gamma}_{\bar{\mathcal{V}}^\sharp}\, \sigma^\sharp = \{x \mapsto \gamma_{\mathcal{V}^\sharp}(\sigma^\sharp\, x) \mid x \in \mathcal{V}ars\}$$

with $\bar{\gamma}_{\bar{\mathcal{V}}^\sharp} \bot = \{x \mapsto \emptyset \mid x \in \mathcal{V}ars\}$. We remark that for a non-relational domain (and $\mathcal{X} \neq \emptyset$, which holds here as $\mathsf{self} \in \mathcal{X}$), these two definitions are equivalent.

We further demand that the *meet* operation $\sqcap$ of $\mathcal{V}^\sharp$ safely overapproximates the intersection of the concretizations of the respective arguments, which, e.g., is the case for Galois connections [29].

Furthermore, we require that the domain provide the following operations:

$$\begin{aligned} [\![x \leftarrow e]\!]^\sharp_{\bar{\mathcal{V}}^\sharp} &: \bar{\mathcal{V}}^\sharp \to \bar{\mathcal{V}}^\sharp \text{ (assignment for expression } e) \\ r|_Y &: \bar{\mathcal{V}}^\sharp \to \bar{\mathcal{V}}^\sharp \text{ (restriction to } Y \subseteq \mathcal{V}ars) \\ [\![?e]\!]^\sharp_{\bar{\mathcal{V}}^\sharp} &: \bar{\mathcal{V}}^\sharp \to \bar{\mathcal{V}}^\sharp \text{ (guard for condition } e) \end{aligned}$$

where the assignment operation may be defined as follows

$$[\![x \leftarrow e]\!]^\sharp_{\bar{\mathcal{V}}^\sharp}\, \sigma = \sigma \oplus \{x \mapsto [\![e]\!]^\sharp_{\mathcal{E}xp}\, \sigma\}$$

for an abstract version of the expression evaluation function $[\![\cdot]\!]_{\mathcal{E}xp}$. Restriction is used to express non-deterministic assignments, i.e., assignments where the right-hand side corresponds to the special expression ?. Restricting a relation r to a subset Y of variables amounts to *forgetting* all information about variables not in Y. Thus, we demand $\sigma|_{\mathcal{V}ars} = \sigma$, $\sigma|_\emptyset = \top$ whenever $\sigma \neq \bot$, $\bot|_\emptyset = \bot$, and $\sigma|_{Y_1} \sqsupseteq \sigma|_{Y_2}$ when $Y_1 \subseteq Y_2$, $(\sigma|_{Y_1})\Big|_{Y_2} = \sigma|_{Y_1 \cap Y_2}$, and for $\sigma \neq \bot$,

$$(\sigma|_Y)\, x = \top \quad (x \notin Y) \qquad\qquad (\sigma|_Y)\, x = \sigma\, x \quad (x \in Y) \tag{3.1}$$

Restriction thus is *idempotent*. The further requirements on these operations are outlined in Fig. 3.1, where the expression evaluation function $[\![\cdot]\!]_{\mathcal{E}xp}$ from Section 2.4 is lifted, so it can be applied to mappings that also have bindings for variables not in $\mathcal{X}$, which will by construction not be accessed.

We further require an abstract version of the computation of a new thread *id* $\nu(t)$ for some thread t. This function $\nu^\sharp : \mathcal{N} \to \bar{\mathcal{V}}^\sharp \to \mathcal{N} \to \mathcal{V}^\sharp_{\mathbf{tid}}$ then needs to satisfy the following soundness property:

$$\begin{aligned} &\forall t \in \mathcal{T} : \forall u_1 \in \mathcal{N} : \forall \sigma^\sharp \in \bar{\mathcal{V}}^\sharp : \forall t' \in \mathsf{new}\, u_1\, t : \\ &\quad \mathsf{sink}\, t = (u_0, \sigma) \wedge \sigma \in \gamma_{\bar{\mathcal{V}}^\sharp}(\sigma^\sharp) \implies \nu(t) \in \gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}(\nu^\sharp\, u_0\, \sigma^\sharp\, u_1) \end{aligned} \tag{3.2}$$

where we remark that — by construction of $\gamma_{\bar{\mathcal{V}}^\sharp}\sigma^\sharp$ — if the set is non-empty, it contains maps that are defined for all elements of $\mathcal{X}$, and thus match the type of the maps in σ. This property ensures that the abstract version of the computation of a new thread *id* is sound w.r.t. its concrete counterpart.

$$\gamma_{\bar{\mathcal{V}}^\sharp}\left([\![x \leftarrow e]\!]^\sharp_{\bar{\mathcal{V}}^\sharp}\,\sigma^\sharp\right) \supseteq \{\sigma \oplus \{x \mapsto [\![e]\!]_{\mathcal{E}xp}\,\sigma\} \mid \sigma \in \gamma_{\bar{\mathcal{V}}^\sharp}\,\sigma^\sharp\}$$
$$\gamma_{\bar{\mathcal{V}}^\sharp}\left(\sigma^\sharp|_Y\right) \supseteq \{\sigma \oplus \{x_1 \mapsto v_1, \ldots, x_m \mapsto v_m\} \mid v_i \in \mathcal{V}, x_i \in \mathcal{V}ars \setminus Y, \sigma \in \gamma_{\bar{\mathcal{V}}^\sharp}\,\sigma^\sharp\}$$
$$\gamma_{\bar{\mathcal{V}}^\sharp}\left([\![?e]\!]^\sharp_{\bar{\mathcal{V}}^\sharp}\,\sigma^\sharp\right) \supseteq \{\sigma \mid \sigma \in \gamma_{\bar{\mathcal{V}}^\sharp}\,\sigma^\sharp, [\![e]\!]_{\mathcal{E}xp}\,\sigma \neq 0\}$$

Figure 3.1: Required properties for operations of the non-relational domain.

3.2 Relational Domains

Next, we define the notion of *relational* domain employed in the description of our analysis in Section 4.2. Here, a *relational domain* $\mathcal{R}$ is a complete lattice which once more provides operations

$$\begin{aligned} [\![x \leftarrow e]\!]^\sharp_{\mathcal{R}} &: \mathcal{R} \to \mathcal{R} \text{ (assignment for expression } e\text{)} \\ r|_Y &: \mathcal{R} \to \mathcal{R} \text{ (restriction to } Y \subseteq \mathcal{V}ars\text{)} \\ [\![?e]\!]^\sharp_{\mathcal{R}} &: \mathcal{R} \to \mathcal{R} \text{ (guard for condition } e\text{)} \end{aligned}$$

where we re-use the symbol for restriction, as it will always be clear, from the type of the value it is applied to, whether the relational or the non-relational version is used. Additionally, $\mathcal{R}$ supplies the functions

$$\mathsf{lift} \;:\; \bar{\mathcal{V}}^\sharp \to \mathcal{R} \qquad\qquad \mathsf{unlift} \;:\; \mathcal{R} \to \bar{\mathcal{V}}^\sharp$$

which allow casting from the non-relational domain to the relational domain as well as extracting single-variable information. We assume that $\mathsf{lift}\,\bot = \bot$ and $\mathsf{unlift}\,\bot = \bot$, and require that $\mathsf{unlift} \circ \mathsf{lift} \sqsupseteq \mathsf{id}$ where $\sqsupseteq$ refers to the ordering of $\bar{\mathcal{V}}^\sharp$. We also demand that unlift distributes[1] over $\sqcup$, i.e.,

$$\mathsf{unlift}\,(x \sqcup_{\mathcal{R}} y) = (\mathsf{unlift}\,x) \sqcup_{\bar{\mathcal{V}}^\sharp} (\mathsf{unlift}\,y) \tag{3.3}$$

where we have indexed joins with their domain here to enhance clarity. Usually, it will be clear from the context which operation is meant, and we will omit such indexes. Moreover, we once more require that the *meet* operation $\sqcap$ (of $\mathcal{R}$ in this case) safely approximates the intersection of the concretizations. For restriction, we once again demand that $r|_{\mathcal{V}ars} = r$, $r|_\varnothing = \top$, $r|_{Y_1} \sqsupseteq r|_{Y_2}$ when $Y_1 \subseteq Y_2$, $(r|_{Y_1})\big|_{Y_2} = r|_{Y_1 \cap Y_2}$, and require for lift and unlift and restriction that, for $r \neq \bot$,

$$\mathsf{unlift}\,(r|_Y)\,x = \top \quad (x \notin Y) \qquad\qquad \mathsf{unlift}\,(r|_Y)\,x = (\mathsf{unlift}\,r)\,x \quad (x \in Y) \tag{3.4}$$

As for the non-relational case, restriction thus is *idempotent*. For convenience, we also define a shorthand for assigning abstract values:

$$[\![x \leftarrow^\sharp v]\!]^\sharp_{\mathcal{R}}\,r = \left(r|_{\mathcal{V}ars \setminus \{x\}}\right) \sqcap (\mathsf{lift}\,(\top \oplus \{x \mapsto v\}))$$

[1] This property is not technically distributivity, as the first occurrence of $\sqcup$ refers to $\mathcal{R}$, whereas the second one refers to $\bar{\mathcal{V}}^\sharp$. We nevertheless call it distributivity to convey the intuition.

$$\gamma_{\mathcal{R}}\,\bot = \emptyset \qquad \gamma_{\mathcal{R}}\,(\mathsf{lift}\,\sigma^\sharp) \supseteq \gamma_{\bar{\mathcal{V}}^\sharp}\,\sigma^\sharp \qquad \gamma_{\bar{\mathcal{V}}^\sharp}\,(\mathsf{unlift}\,r) \supseteq \gamma_{\mathcal{R}}\,r$$

$$\forall r, s : r \sqsubseteq s \implies \gamma_{\mathcal{R}}\,r \subseteq \gamma_{\mathcal{R}}\,s$$

$$\begin{aligned}\gamma_{\mathcal{R}}\,([\![x \leftarrow e]\!]^\sharp_{\mathcal{R}}\,r) &\supseteq \{\sigma \oplus \{x \mapsto [\![e]\!]_{\mathcal{E}xp}\,\sigma\} \mid \sigma \in \gamma_{\mathcal{R}}\,r\}\\ \gamma_{\mathcal{R}}\,(r|_Y) &\supseteq \{\sigma \oplus \{x_1 \mapsto v_1, \ldots, x_m \mapsto v_m\} \mid v_i \in \mathcal{V}, x_i \in \mathcal{V}ars \setminus Y, \sigma \in \gamma_{\mathcal{R}}\,r\}\\ \gamma_{\mathcal{R}}\,([\![?e]\!]^\sharp_{\mathcal{R}}\,r) &\supseteq \{\sigma \mid \sigma \in \gamma_{\mathcal{R}}\,r, [\![e]\!]_{\mathcal{E}xp}\,\sigma \neq 0\}\end{aligned}$$

Figure 3.2: Required properties for $\gamma_{\bar{\mathcal{V}}^\sharp} : \bar{\mathcal{V}}^\sharp \to 2^{\mathcal{V}ars \Rightarrow \mathcal{V}}$ and $\gamma_{\mathcal{R}} : \mathcal{R} \to 2^{\mathcal{V}ars \Rightarrow \mathcal{V}}$.

We once more require a monotonic concretization function $\gamma_{\mathcal{R}} : \mathcal{R} \to 2^{\mathcal{V}ars \Rightarrow \mathcal{V}}$ which together with $\gamma_{\bar{\mathcal{V}}^\sharp}$ must satisfy the requirements presented in Fig. 3.2 where $[\![\cdot]\!]_{\mathcal{E}xp}$ is once again lifted as above.

Example 11. *As a value domain $\mathcal{V}^\sharp$, consider the flat lattice over the sets of values of appropriate type τ, and for $\bar{\mathcal{V}}^\sharp$ the set of corresponding maps from variables to abstract values, extended with $\bot$ as least element. A relational domain $\mathcal{R}_1$ is obtained by collecting satisfiable conjunctions of equalities between variables or variables and constants where the ordering is logical implication, extended with* False *as least element. The greatest element in this complete lattice is given by* True*. The operations lift and unlift for non-$\bot$ arguments then can be defined as*

$$\mathit{lift}\,\sigma = \bigwedge\{x = \sigma\,x \mid x \in \mathcal{V}ars, \sigma\,x \neq \top\} \qquad \mathit{unlift}\,r\,x = \begin{cases} c & \text{if } r \implies (x = c) \\ \top & \text{otherwise} \end{cases}$$

The restriction of r to a subset Y of variables is given by the conjunction of all equalities implied by r that only contain variables from Y or constants. □

3.3 2-Decomposability

A variable clustering $\mathcal{S} \subseteq 2^{\mathcal{V}ars}$ is a set of subsets (*clusters*) of variables. For any cluster $Y \subseteq \mathcal{V}ars$, let $\mathcal{R}^Y = \{r \mid r \in \mathcal{R}, r|_Y = r\}$; this set collects all abstract values from $\mathcal{R}$ containing information on variables in Y only. Given an arbitrary clustering $\mathcal{S} \subseteq 2^{\mathcal{V}ars}$, *any* relation $r \in \mathcal{R}$ can be approximated by a meet of relations from $\mathcal{R}^Y$ ($Y \in \mathcal{S}$) since for every $r \in \mathcal{R}$, $r \sqsubseteq \prod\{r|_Y \mid Y \in \mathcal{S}\}$ holds.

Some relational domains, however, can be fully recovered from their restrictions to specific subsets of clusters. We consider for $k \geq 1$, the set $[\mathcal{V}ars]_k = \{Y \mid Y \subseteq \mathcal{V}ars, 1 \leq |Y| \leq k\}$, i.e, the set of all non-empty subsets $Y \subseteq \mathcal{V}ars$ of cardinality at most k. We call a relational domain $\mathcal{R}$ *k-decomposable* if each abstract value from $\mathcal{R}$ can be precisely expressed as the meet of its restrictions to clusters of $[\mathcal{V}ars]_k$ and when all least upper bounds can be recovered by computing with clusters of $[\mathcal{V}ars]_k$ only; that is,

$$r = \prod\left\{r|_Q \mid Q \in [\mathcal{V}ars]_k\right\} \qquad (\bigsqcup R)\,|_Q = \bigsqcup\left\{r|_Q \mid r \in R\right\} \quad (Q \in [\mathcal{V}ars]_k) \tag{3.5}$$

holds for each abstract relation $r \in \mathcal{R}$ and each set of abstract relations $R \subseteq \mathcal{R}$.

Example 12. *The domain $\mathcal{R}_1$ from the previous example is* 2*-decomposable. This also holds for the* octagon *domain [81] and many other weakly relational numeric domains such as pentagons [78], weighted hexagons [51], logahedra [62], TVPI [120], dDBM [100], and AVO [27]. The* affine equalities *or* inequalities *domains [35, 68], however, are not* 2*-decomposable.* □

However, the concept of 2-decomposability does not only apply to numerical domains. For example, Seidl et al. [111] study the 2-decomposability of various non-numerical domains and construct 2-decomposable approximations of non-numerical domains that are not 2-decomposable themselves.

Remark 5. *By setting $\bar{\mathcal{V}}^\sharp = \mathcal{R}$ and letting $\mathsf{lift} = \mathsf{unlift} = \mathsf{id}$, one can obtain from any non-relational domain $\bar{\mathcal{V}}^\sharp$ a domain that fits into the framework of relational domains presented here. Thus, any relational analysis presented in the subsequent chapters can naturally be instantiated with a non-relational domain via this construction.*

4 Static Analysis by Abstract Interpretation

In a series of papers starting in 2011, Antoine Miné and his co-authors developed analyses of the values of globals [83–85, 87, 129, 130] based on abstract interpretation that can be considered the *gold standard* for thread-modular static value analysis. The core analysis [83, 84] consists of a refinement of data flow which takes schedulability into account by propagating values written before unlock operations to corresponding lock operations — provided that appropriate side-conditions are met. Due to these side-conditions, more flows are generally excluded than in other approaches [38, 89]. An alternative approach by Vojdani [133, 134] is realized in the static analyzer Goblint. This analysis is not based on data flows. Instead, for each global g, a set of mutexes that definitely protect accesses to g is determined. Then, *side-effects* during the analysis of the threads' local states are used to accumulate an abstraction of the set of all possibly written values. This approach then is enhanced by means of *privatization* to account for exclusive manipulations by individual threads similar to the thread-local shape analysis of Gotsman et al. [55], which infers lock-invariants [96] by privatizing carved-out sections of the heap owned by a thread. Despite its conceptual simplicity and perhaps to our surprise, it turns out the Vojdani style analysis is *not* subsumed by Miné's approach, but that the approaches are incomparable. Since Miné's analysis is more precise on many examples, we highlight only non-subsumption in the other direction here.

Example 13. *We use sets of integers for abstracting int values. Consider the following concurrent program*[1] *with global variable* `g` *and local variables* `x` *and* `y`*:*

```
main:                    t1:
  initMT;                  lock(a);
  y = create(t1);          lock(b);
  lock(a);                 g = 42;
  lock(b);                 unlock(a);
  x = g;                   g = 17;
  ...                      unlock(b);
```

Program execution starts at program point `main` *where, after creation of another thread* `t1` *and locking of the mutexes* `a` *and* `b`*, the value of the global* `g` *is read. The created thread, on the other hand, also locks the mutexes* `a` *and* `b`*. Then, it writes to* `g` *the two values 42 and 17 where mutex* `a` *is unlocked in-between the two writes, and mutex* `b` *is unlocked only in the very end.*

According to Miné's analysis, the value $\{42\}$ *is merged into the local state at the operation* `lock(a)`*, while* $\{17\}$ *is merged at* `lock(b)`*. Thus, the local* `x` *receives the value* $\{0, 17, 42\}$.

[1] We have omitted mutexes m_g in this example for clarity.

Vojdani's analysis, on the other hand, finds that all accesses to `g` *are protected by the mutex* `b`*. Unlocking of* `a`*, therefore, does not publish the intermediately written value* $\{42\}$*. Only the final value* $\{17\}$ *at* `unlock(b)` *is published. Therefore, the local* `x` *only receives the value* $\{0, 17\}$*.* □

The jumping off point for this part of the work was a desire to understand this intriguing incomparability and the desire to develop precision refinements to improve either flavor of analysis. Here, we focus on the sequentially consistent setting, ignoring effects of weak memory, which are of major concern in some works, e.g., in [6, 48, 129, 130]. Also, we assume that we are in a setting akin to Pthreads, i.e., in a setting with dynamic thread creation and joining — but without priorities as they are, e.g., common in systems in an automotive or avionic setting.

In Section 4.1, we propose non-relational analyses of the values of globals that consider globals in isolation. To this end, we first cast both the analysis proposed by Miné [83, 84] adapted to our setting and the one proposed by Vojdani [133, 134] into a common framework — namely side-effecting constraint systems as also used to formulate the local trace semantics. This lays the groundwork for providing enhancements to improve the precision of either style, as well as an implementation in a common tool which allows experimentation on real-world benchmarks (Chapter 5).

Section 4.2 then goes beyond analyses that consider globals in isolation, which renders them inherently non-relational. Here, we do not track arbitrary relations, but instead focus on those relations within (potentially overlapping) clusters of globals that are mediated by protecting mutexes. The intuition behind this choice is that developers usually rely on *robust* means to ensure mutual exclusion, and intricate invariants are thus not encountered in high-level code too frequently.

For both analyses considering globals in isolation and those considering clusters of globals, we allow refinement according to digests as discussed in Section 2.3. While the analyses abstracting clusters of globals incorporate information at lock operations, and thus allow for arbitrary digests to be considered, the analyses abstracting globals in isolation delay the incorporation of values past the point where lock is called, and thus only allow for ego-lane digests (Section 2.9). One digest of particular interest here is the thread *id* digest from Section 2.8, which allows computing thread *id*s in a setting such as ours, where threads are dynamically created and joined.

We proceed similarly for all analyses: First giving an intuitive description of the analysis and giving its constraint system, proceeding to (a sketch of) a soundness proof, and discussing potential limitations and possible extensions last.

For all analyses, we assume that the refinement from Fig. 2.11 has been applied, i.e., that unknowns for program points take the form $[u, S]$ for $u \in \mathcal{N}$ and $S \subseteq \mathcal{M}$. While the unknowns for program points and locksets do coincide between the analyses and the (refined) concrete semantics, this is not necessarily the case for other unknowns. There, more intricate arguments are required to relate solutions of the abstract constraint system to solutions of the constraint system representing the concrete semantics.

Table 4.1 provides an overview of the different analyses and highlights some advantages and shortcomings. Since, just like the unknowns associated with program points,

Table 4.1: Overview of the families of analyses presented in this thesis.

Name	Relational for $\mathcal{G}$	Requires $\bar{\mathcal{M}}$	Unknowns associated with globals and/or mutexes	Refinable	Proof
Protection-Based (Section 4.1.1)	✗	✓	$[g], [g']$ for $g \in \mathcal{G}$	Ego-Lane Digests	As an abstraction of Write-Centered
Lock-Centered (Section 4.1.4)	✗	✗	$[g, a, S]$ for $g \in \mathcal{G}, a \in \mathcal{M}$, $S \subseteq \mathcal{M}$	Ego-Lane Digests	Directly
Write-Centered (Section 4.1.6)	✗	✗	$[g, a, S, w]$ for $g \in \mathcal{G}, a \in \mathcal{M}$, $S \subseteq \mathcal{M}, w \subseteq \mathcal{M}$	Ego-Lane Digests	Directly
Write- and Lock-Centered (Section 4.1.8)	✗	✗	$[g, a, S, w]$ for $g \in \mathcal{G}, a \in \mathcal{M}$, $S \subseteq \mathcal{M}, w \subseteq \mathcal{M}$	Ego-Lane Digests	Via proofs of Write- and Lock-Centered
Mutex-Meet (Section 4.2.1)	✓	✓	$[a, Q]$ for $a \in \mathcal{M}, Q \in \mathcal{Q}_a$ (where $\mathcal{Q}_a$ are clusters of globals associated with a)	✓	Directly

the unknowns associated with thread returns ($[i]$ for $i \in S_{\mathcal{V}^\sharp_{\mathbf{tid}}}$) and signals ($[s]$ for $s \in \mathcal{S}$) take the same form for all analyses, they are not listed in the table. This table may be helpful to refer back to when reading later sections.

For clarity of presentation, we will always denote the $\bot$ element of the respective domains of each analysis that denotes unreachability by the common symbol $\underline{\bot}$.

The analyses presented in this chapter were first proposed in Schwarz et al. [107] (Section 4.1), and in Schwarz et al. [108] (Section 4.2), respectively. Compared to this earlier work, we enhance all analyses with a handling of signalling and waiting, and enhance the non-relational analyses with a handling of returning and thread joins which were not considered in the earlier work [107]. Additionally, refinement according to ego-lane digests for these analyses is an original contribution of this thesis.

4.1 Analyses Considering Globals in Isolation

In this section, we propose four novel analyses that consider globals in isolation, i.e., are non-relational. The first analysis (*Protection-Based Reading*) is an improved version of Vojdani's analysis [133, 134]. It assumes that for each global g, some set $\bar{\mathcal{M}}[g]$ of mutexes exists that is held at each write operation to g and maintains a private copy of the global as long as one of the mutexes from $\bar{\mathcal{M}}[g]$ is known to be held.

The second analysis is an improved version of the analysis proposed by Miné [83, 84] – when adapted to a setting with dynamic thread creation as recapped and cast into terms of side-effecting constraint systems in Section 4.1.3. It addresses two shortcomings of the original analysis by tracking additional information about the history of the ego thread.

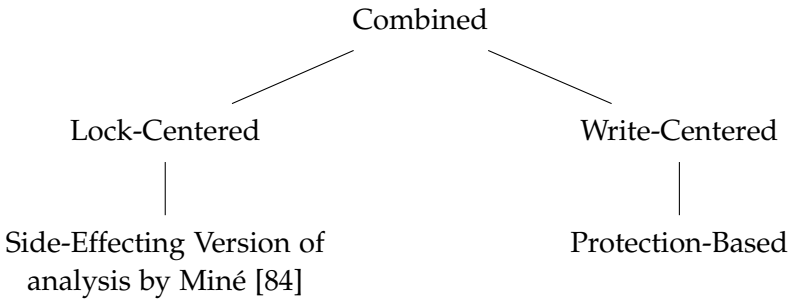


Figure 4.1: Precision relationship between analyses considering globals in isolation.

As this information is organized by mutex, the analysis is called *Lock-Centered Reading*.

To obtain good invariants, *Protection-Based Reading* relies on the set of protecting mutexes for a global being non-empty, i.e., some mutex (other than the atomicity mutex) is held at every write access program-wide. As this may not always be the case, we present a third analysis which lifts this extra assumption and strictly subsumes *Protection-Based Reading*. Once again, this analysis tracks some extra information about the history of the ego thread. This information is organized around individual global variables, which is why the analysis is called *Write-Centered Reading*. Interestingly, *Write-Centered Reading* and *Lock-Centered Reading* are still incomparable. Therefore, we sketch a fourth non-relational analysis that is more precise than either of the three other analyses.

The relationship between the precision of these five analyses is visualized as a Hasse diagram in Fig. 4.1, where the top element corresponds to the most precise analysis.

4.1.1 Protection-Based Reading

The original analysis proposed by Vojdani [133, 134] and implemented in the Goblint system assumes that for each global g, there is a set of mutexes definitely held whenever g is accessed. The best information about the values of g available after acquiring a protecting lock is maintained in a separate unknown $[g]$. The value of the unknown $[g]$ for the global g is eagerly "privatized": That is, it is incorporated into the local state for a program point and currently held lockset whenever g first becomes protected, i.e., a mutex protecting g is acquired while none was held before. As long as one of these protecting mutexes is held, all reads and writes refer to this local copy of the global, and the copy can be destructively updated. It is only when the last mutex protecting g is released that the value of the local copy is potentially visible to other threads, and thus is published to the unknown $[g]$. This base setting is enhanced in three ways:

- Instead of assuming a set of mutexes protecting *both reading and writing* of g, we now consider the mutexes held whenever the global is *written* as protecting. While this does not necessarily lead to an improvement in precision, it allows for analyzing interesting patterns where, e.g., only a subset of mutexes is acquired for reading from a global, while a superset is held when writing to it.

- Besides the unknown $[g]$ for describing the possible values of the global g for protected accesses, another unknown $[g]'$ is introduced for the results of unprotected read accesses to g, i.e., accesses where none of the write-protecting mutexes — other than the atomicity mutex m_g — are held.
- Instead of incorporating the value of the global g stored at $[g]$ into the local state at each lock operation of a mutex from the protecting set, the local state for a program point and currently held lockset only keeps track of the values written by the ego thread. At a read operation $x = g$, the value of global g is assigned to the local variable x. For that, the analysis relies on the value stored at unknown $[g]$ together with the value of g stored in the local state. However, if the ego thread has definitely written to g since acquiring a protecting mutex, and has not yet released all protecting mutexes since then, it is sufficient to consider the local copy as any other value will definitely have been overwritten.

Recall that $\bar{\mathcal{M}} : \mathcal{G} \to 2^{\mathcal{M}}$ maps each global g to the set of mutexes definitely held when g is *written to*. Due to our atomicity assumption, the set $\bar{\mathcal{M}}[g]$ is non-empty, since $m_g \in \bar{\mathcal{M}}[g]$ always holds. We also require that $m_h \in \bar{\mathcal{M}}[g] \implies h = g$, i.e., the special mutexes m_g only appear in the set of write-protecting mutexes for the global g they belong to. In this thesis, we assume this mapping to be given, but remark that this requirement could be lifted as outlined in Remark 9. The unknown $[g]'$ stores an abstraction (from $\mathcal{V}^\sharp$) of *all* values ever written to g, while the unknown $[g]$ stores an abstraction (from $\mathcal{V}^\sharp$) of those values that were written *last* before releasing a protecting mutex of g other than m_g. For each pair (u, S) of program point u and currently held lockset S, on the other hand, the analysis maintains (1) a set P of definitely written globals g since a protecting mutex of g has been acquired and not all protecting mutexes have been released (ordered by $\supseteq$), together with (2) a variable assignment $\sigma : \bar{\mathcal{V}}^\sharp$ of potential descriptions of values for local or global variables.

When a mutex in $\bar{\mathcal{M}}[g]$ is definitely held, after a write to variable g, all modifications to g are performed destructively on the local copy, and further reads also refer to this copy. Immediately after the write to g (at the $\mathsf{unlock}(m_g)$) the updated value of the local copy is merged into $[g]'$ via a side-effect. On the other hand, the value of the copy must be merged into $[g]$ only when it can no longer be guaranteed that all other protecting mutexes $(\bar{\mathcal{M}}[g] \setminus \{m_g\})$ are held, i.e., when the first protecting mutex is released.

After having provided the key intuition, we now describe the analysis in a more formal way: We start by giving the right-hand-side function for the start state at the initial program point of the main thread $u_0 \in \mathcal{N}$ with the empty lockset $\emptyset$, i.e., $[u_0, \emptyset] \sqsupseteq \mathsf{init}^\sharp$:

$$
\begin{aligned}
\mathsf{init}^\sharp\ \_ \quad = \quad & \textbf{let } \sigma = \{x \mapsto \top \mid x \in \mathcal{X}\} \cup \{g \mapsto [\![0]\!]^\sharp_{\mathcal{E}xp}\ \top \mid g \in \mathcal{G}\} \textbf{ in} \\
& \textbf{let } \rho = \{[g] \mapsto [\![0]\!]^\sharp_{\mathcal{E}xp}\ \top \mid g \in \mathcal{G}\} \cup \{[g]' \mapsto [\![0]\!]^\sharp_{\mathcal{E}xp}\ \top \mid g \in \mathcal{G}\} \textbf{ in} \\
& (\rho, (\emptyset, \sigma \oplus \{\mathsf{self} \mapsto [\![i_0]\!]^\sharp_{\mathcal{E}xp}\ \top\}))
\end{aligned}
$$

That is, on top of computing the start state for the analysis, an initial side-effect is triggered to publish the information that globals may have the initial value 0. We remark

that any mapping can be given to the abstract expression evaluation function $[\![\cdot]\!]^\sharp_{\mathcal{E}xp}$ in this case, as both expressions evaluate to constants.

The right-hand side for the action initMT returns the abstract state of the predecessor and causes no side-effects.

Now, consider the right-hand side $[v, S'] \sqsupseteq [\![[u, S], \mathsf{act}]\!]^\sharp$ for the edge $e = (u, \mathsf{act}, v)$ of the control-flow graph and appropriate locksets S and S'. Consider the right-hand side for a thread creation edge. Recall that we have a function $\nu^\sharp\, u\, \sigma\, u_1$ that returns the (abstract) thread *id* of a thread started at an edge originating from u in some local state σ, where the new thread starts execution at program point u_1. The returned argument is an element of $\mathcal{V}^\sharp_{\mathbf{tid}}$ for which the soundness condition from (3.2) is satisfied.

$$
\begin{array}{lcl}
[\![[u, S], x = \mathsf{create}(u_1)]\!]^\sharp \eta & = & \textbf{let } (P, \sigma) = \eta\,[u, S] \textbf{ in} \\
& & \textbf{let } i = \nu^\sharp\, u\, \sigma\, u_1 \textbf{ in} \\
& & \textbf{let } \sigma' = \sigma \oplus (\{\mathsf{self} \mapsto i\} \cup \{g \mapsto [\![0]\!]^\sharp_{\mathcal{E}xp}\, \top \mid g \in \mathcal{G}\}) \textbf{ in} \\
& & \textbf{let } \rho = \{[u_1, \emptyset] \mapsto (\emptyset, \sigma')\} \textbf{ in} \\
& & (\rho, (P, \sigma \oplus \{x \mapsto i\}))
\end{array}
$$

This function has no effect on the local state apart from setting x to the abstract thread *id* of the newly created thread, while providing an appropriate initial state to the start point of the newly created thread. The value for any global g is set to $[\![0]\!]^\sharp_{\mathcal{E}xp}\, \top$ in this initial state, as in the newly created thread, no local writes to g have happened yet.[2] For guards and computations on locals, the right-hand-side functions are defined to operate on σ only, using the assignment $[\![x \leftarrow e]\!]^\sharp_{\hat{\mathcal{V}}^\sharp}$, restriction $r|_{\mathcal{V}ars \setminus \{x\}}$ (for assignments of ? to x), and guard $[\![?e]\!]^\sharp_{\hat{\mathcal{V}}^\sharp}$ operations defined by the domain, leaving P unchanged.

Concerning locking and unlocking of a mutex a, the lock operation does not affect the local state, while at each unlock, all local copies of globals g for which not all protecting mutexes are held anymore, are published via a side-effect to the respective unknowns $[g]$ or $[g]'$. Moreover, globals for which none of the protecting mutexes are held anymore, are removed from P. We thus have:

$$
\begin{array}{lcl}
[\![[u, S], \mathsf{lock}(a)]\!]^\sharp \eta & = & (\emptyset, \eta\,[u, S]) \\
[\![[u, S], \mathsf{lock}(m_g)]\!]^\sharp \eta & = & (\emptyset, \eta\,[u, S]) \\
[\![[u, S], \mathsf{unlock}(m_g)]\!]^\sharp \eta & = & \textbf{let } (P, \sigma) = \eta\,[u, S] \textbf{ in} \\
& & \textbf{let } P' = \{h \in P \mid ((S \setminus \{m_g\}) \cap \bar{\mathcal{M}}[h]) \neq \emptyset\} \textbf{ in} \\
& & \textbf{let } \rho = \{[g]' \mapsto \sigma\, g\} \cup \{[g] \mapsto \sigma\, g \mid \bar{\mathcal{M}}[g] = \{m_g\}\} \textbf{ in} \\
& & (\rho, (P', \sigma)) \\
[\![[u, S], \mathsf{unlock}(a)]\!]^\sharp \eta & = & \textbf{let } (P, \sigma) = \eta\,[u, S] \textbf{ in} \\
& & \textbf{let } P' = \{g \in P \mid ((S \setminus \{a\}) \cap \bar{\mathcal{M}}[g]) \neq \emptyset\} \textbf{ in} \\
& & \textbf{let } \rho = \{[g] \mapsto \sigma\, g \mid a \in \bar{\mathcal{M}}[g]\} \textbf{ in} \\
& & (\rho, (P', \sigma))
\end{array}
$$

[2] The value $[\![0]\!]^\sharp_{\mathcal{E}xp}\, \top$ is always also read via $[g]$, so it would also be sound to set the local value to $\bot$. We do not do this here, as it has no precision impact but causes complications for some of our proofs.

for $a \notin \{m_g \mid g \in \mathcal{G}\}$. Recall that the dedicated mutex m_g for each global g has been introduced for guaranteeing atomicity. It is always acquired immediately before and always released immediately after each access to g. The special treatment of this dedicated mutex ensures that all values written to g are side-effected to the unknown $[g]'$, while values written to g are side-effected to the unknown $[g]$ only when unlock is called for a protecting mutex *different* from m_g.

For global g and local x, we define for writing to and reading from g,

$$\begin{aligned}
[\![[u,S], g = x]\!]^\sharp \eta &= \textbf{let } (P,\sigma) = \eta\,[u,S] \textbf{ in} \\
&\quad (\emptyset, (P \cup \{g\}, \sigma \oplus \{g \mapsto (\sigma\, x)\})) \\
[\![[u,S], x = g]\!]^\sharp \eta &= \textbf{let } (P,\sigma) = \eta\,[u,S] \textbf{ in} \\
&\quad \textbf{if } g \in P \textbf{ then} \\
&\qquad (\emptyset, (P, \sigma \oplus \{x \mapsto (\sigma\, g)\})) \\
&\quad \textbf{else if } S \cap \bar{\mathcal{M}}[g] = \{m_g\} \textbf{ then} \\
&\qquad (\emptyset, (P, \sigma \oplus \{x \mapsto \sigma\, g \sqcup \eta\,[g]'\})) \\
&\quad \textbf{else} \\
&\qquad (\emptyset, (P, \sigma \oplus \{x \mapsto \sigma\, g \sqcup \eta\,[g]\}))
\end{aligned}$$

For realizing returns from threads and thread joins, we rely on side-effects to unknowns corresponding to the abstract thread *id* of the thread calling return, i.e., $\sigma\,\mathsf{self}$ which is from the lattice $\mathcal{V}^\sharp_{\mathbf{tid}}$. To ensure monotonicity of the side-effects, any unknown that receives a side-effect for any smaller value of $\sigma\,\mathsf{self}$, also needs to receive a side-effect for the current value of self. To this end, the unknowns for thread *id*s are not from $\mathcal{V}^\sharp_{\mathbf{tid}}$ but from $S_{\mathcal{V}^\sharp_{\mathbf{tid}}}$, and all unknowns contained in $\sigma\,\mathsf{self}$ receive a side-effect.

Recall that the concrete semantics provides a special local variable ret whose value is the one returned by the thread.

$$\begin{aligned}
[\![[u,S], \mathsf{return}]\!]^\sharp \eta &= \textbf{let } (P,\sigma) = \eta\,[u,S] \textbf{ in} \\
&\quad \textbf{let } I = \sigma\,\mathsf{self} \textbf{ in} \\
&\quad (\{[i] \mapsto \sigma\,\mathsf{ret} \mid i \in I\}, (P,\sigma)) \\
[\![[u,S], x = \mathsf{join}(x')]\!]^\sharp \eta &= \textbf{let } (P,\sigma) = \eta\,[u,S] \textbf{ in} \\
&\quad \textbf{let } v = \textstyle\bigsqcup_{i' \in (\sigma\, x')} (\eta[i']) \textbf{ in} \\
&\quad \textbf{if } v = \bot \textbf{ then} \\
&\qquad (\emptyset, \underline{\bot}) \\
&\quad \textbf{else} \\
&\qquad (\emptyset, (P, \sigma \oplus \{x \mapsto v\}))
\end{aligned}$$

Thus, upon a call of return the value of ret is side-effected to all unknowns possibly corresponding to the abstract thread *id*. All such unknowns are then consulted upon a join operation — provided they are in the set of values stored in x'. If the value of all such unknowns is $\bot$, none of the threads potentially being joined have terminated. In this case, the return value of the entire right-hand side is set to $\underline{\bot}$ to denote unreachability.

For $\mathsf{signal}(s)$ and $\mathsf{wait}(s)$, we define

$$\begin{aligned}
[\![[u,S],\mathsf{signal}(s)]\!]^\sharp\eta &= \textbf{let } (P,\sigma) = \eta\,[u,S] \textbf{ in} \\
&\quad (\{[s] \mapsto (P,\sigma)\}, (P,\sigma)) \\
[\![[u,S],\mathsf{wait}(s)]\!]^\sharp\eta &= \textbf{let } (P,\sigma) = \eta\,[u,S] \textbf{ in} \\
&\quad \textbf{if } \eta\,[s] = \underline{\bot} \textbf{ then} \\
&\qquad (\emptyset, \underline{\bot}) \\
&\quad \textbf{else} \\
&\qquad (\emptyset, (P,\sigma))
\end{aligned}$$

Thus, signal publishes its current local state to the unknown associated with the condition variable, and wait checks whether the corresponding condition variable is ever potentially signaled. If it is, the local state is returned unmodified. Otherwise, the entire right-hand side returns $\underline{\bot}$ as the successor is unreachable.

Remark 6. *We remark that all right-hand sides are monotonic here – both in the returned values and in the side-effects, given that the assignment and guard functions provided by the domain are also monotonic. In this case, the constraint system has a unique least solution, which we denote by $\eta^\sharp$ here. We remark that for this unique least solution $\eta^\sharp$, $\eta^\sharp\,[g] \sqsubseteq \eta^\sharp\,[g]'$ holds.*

Example 14. *Consider, e.g., the following program fragment and assume that $\bar{\mathcal{M}}[g] = \{a, m_g\}$ and that we use value sets for abstracting int values.*

```
initMT;
lock(a);
lock(m_g);  g = 5;     unlock(m_g);
lock(b);
unlock(b);
lock(m_g);  x = g;     unlock(m_g);
lock(m_g);  g = x+1;   unlock(m_g);
unlock(a);
```

Then, after $\mathsf{unlock}(b)$, the state attained by the program (where variables self and ret are omitted for clarity of presentation) is

$$s_1 = (\{g\}, \{g \mapsto \{5\}, x \mapsto \top\})$$

where $[g]'$ has received the contribution $\{5\}$ but no side-effect to $[g]$ has been triggered. The read of g in the subsequent assignment refers to the local copy. Accordingly, the second write to g and the succeeding $unlock(m_g)$ result in the local state

$$s_2 = (\{g\}, \{g \mapsto \{6\}, x \mapsto \{5\}\})$$

with side-effect $\{6\}$ to $[g]'$ and no side-effect to $[g]$. Accordingly, after $\mathsf{unlock}(a)$, the attained state is

$$s_3 = (\emptyset, \{g \mapsto \{6\}, x \mapsto \{5\}\})$$

and the value of $[g]$ is just $\{0,6\}$ — even though g has been written twice. We remark that without separate treatment of m_g, the value of $\{5\}$ would immediately be side-effected to $[g]$. □

While this example highlights the behavior w.r.t. mutexes and global variables, the following example highlights the other concurrency primitives thread creation and joining as well as signalling and waiting.

Example 15. *Assume integer sets are used for abstracting int values, and thread* ids *distinguishing threads by the program point at which they start, i.e., their templates are used. Consider the following concurrent program and assume that* $\bar{\mathcal{M}}[g] = \{m_g\}$.

```
main:                              t1:
  initMT;                            lock(m_g); g = 17; unlock(m_g);
  y = create(t1);                    wait(s);
  lock(m_g); x = g; unlock(m_g);     ret = 8;
                                     return;
  if(x == 0) {
    y = create(t2);                t2:
  }                                  lock(m_g); g = 42; unlock(m_g);
  signal(s);                         wait(t);
  x = join(y);                       ret = 42;
  // ASSERT(x == 8);                 return;
```

First, the main thread creates thread t_1*, which may set the value of the global* g *to 17. Its thread* id *is stored into a variable* y*. Then, the main thread reads the value of* g *into a local variable* x*. If* x *is* 0*, a new thread* t_2 *is started and the thread* id *is stored into the variable* y*, overwriting the stored value. Then, the condition variable* s *is signalled and lastly join is called on the thread* id *stored into* y*. Here, the analysis succeeds in showing the assertion at the end of main. The reasoning is as follows: The analysis for thread* t_2 *consults the unknown* $[t]$ *when it reaches the call to* $\mathsf{wait}(t)$*. As no thread ever calls* $\mathsf{signal}(t)$*, all succeeding program points are found to be dead. In* t_1*, on the other hand, the value of the unknown* $[s]$ *is not* $\bot$ *as the current local state is side-effected by the main thread when it calls* $\mathsf{signal}(s)$*. Thus, thread* t_1 *reaches the return statement with an abstract state* $(\emptyset, \{\mathsf{ret} \mapsto \{8\}, \mathsf{self} \mapsto \{t_1\}, \dots\})$*, and the value* $\{8\}$ *is side-effected to the unknown* $[t_1]$*. In the main thread, the call to* $\mathsf{join}(y)$ *is reached with an abstract state of the form* $(\emptyset, \{y \mapsto \{t_1, t_2\}, \dots\})$*. Thus, the unknowns* $[t_1]$ *and* $[t_2]$ *are consulted. As only* $[t_1]$ *has received any side-effects, for the state after the call to join, we have* $(\emptyset, \{y \mapsto \{t_1, t_2\}, x \mapsto \{8\}\})$ *and the assertion thus holds.* □

Theorem 5. Protection-Based Reading *is sound w.r.t. the local trace semantics.*

Proof. In Section 4.1.2, we show how to obtain this analysis as an instance of the refined analysis presented there. The soundness proof for that analysis is, in turn, deferred to Section 6.1.3. □

Remark 7. *The precision for calls to* $\mathsf{join}(x')$ *could further be improved here by considering all*

possible values of x separately and refining the local state before the join accordingly.

$$\begin{aligned}[[u, S], x = \mathsf{join}(x')]^\sharp \eta \quad = \quad & \textbf{let } (P, \sigma) = \eta\,[u, S] \textbf{ in} \\ & \textbf{let } V\,i' = \textbf{if } \eta\,[i'] = \bot \textbf{ then} \\ & \qquad \bot \\ & \quad \textbf{else} \\ & \qquad (P, \sigma \oplus \{x \mapsto \eta\,[i'], x' \mapsto \{i'\}\}) \\ & \textbf{in} \\ & \left(\emptyset, \bigsqcup_{i' \in (\sigma\,x')} (V\,i')\right)\end{aligned}$$

Then, the value for x' can be refined in case it is known that, for some elements of the abstract thread id *stored in x', the call to return is not reached by any of the threads with a corresponding concrete thread* id*. In this case, the call to $\mathsf{join}(x')$ will not terminate if the value of x' is the thread* id *of one of these threads. Consider again Example 15. In this case, the value of y after the call to join could be refined to $\{t_1\}$. To not overcomplicate the presentation later, we choose to forgo this opportunity but remark that it applies to all the analyses presented in this thesis.*

Remark 8. *As presented, the analysis always maintains some information about each global variable, which may lead to large local states. Alternatively, one may remove information about globals from the local state as soon as all protecting mutexes are released. Then, an implementation may set the start state to not contain any information on the globals, and then may additionally track for each program point and currently held lockset, a set W of all globals which possibly have been written (and not yet published) while holding protecting mutexes. A local copy of a global g may then safely be removed from σ (set to $\bot$) if $g \notin P \cup W$. This is possible because for each $g \notin P \cup W$, $\sigma\,g$ has already been side-effected and hence is already included in $\eta\,[g]$ and $\eta\,[g]'$, and thus the value $\sigma\,g$ would normally have, is already accounted for through $\eta\,[g]$ or $\eta\,[g]'$, one of which is always consulted whenever a global is read that is not in P.*

Remark 9. *This analysis, as discussed here, requires the map $\bar{\mathcal{M}} : \mathcal{G} \to 2^{\mathcal{M}}$ to be given beforehand. This map can, e.g., be provided by some pre-analysis. Alternatively, our analysis can be modified to infer $\bar{\mathcal{M}}$ on the fly. For that, we consider the $\bar{\mathcal{M}}[g]$ to be separate unknowns of the constraint system. They take values in the complete lattice $2^{\mathcal{M}}$ (ordered by superset) and each $\bar{\mathcal{M}}[g]$ is initialized to the full set of all mutexes except special mutexes m_h for $h \neq g$, i.e., initially $\bar{\mathcal{M}}[g] = \mathcal{M} \setminus \{m_h \mid h \in \mathcal{G} \setminus \{g\}\}$. The right-hand-side function for writes to a global g then is extended to provide the current lockset as a contribution to $\bar{\mathcal{M}}[g]$:*

$$\begin{aligned}[[u, S], g = x]^\sharp \eta \quad = \quad & \textbf{let } (P, \sigma) = \eta\,[u, S] \textbf{ in} \\ & (\{\bar{\mathcal{M}}[g] \mapsto S\}, (P \cup \{g\}, \sigma \oplus \{g \mapsto (\sigma\,x)\}))\end{aligned}$$

There is one (minor) obstacle, though: the right-hand-side for control-flow edges with $\mathsf{unlock}(a)$ is no longer monotonic in the unknowns $\bar{\mathcal{M}}[g], g \in \mathcal{G}$: If $\bar{\mathcal{M}}[g]$ shrinks to no longer contain a, $\mathsf{unlock}(a)$ will no longer produce a side-effect to the unknown $[g]$, whereas it previously did.

Remark 10. *To improve efficiency in a implementation, it is also possible to abandon state-splitting according to held locksets — at the cost of losing some precision. To this end, it suffices to additionally track for each program point a set $\bar{S}$ of must-held mutexes as part of the local state from the lattice $2^{\mathcal{M}}$ (ordered by superset) and replace S with $\bar{S}$ in all right-hand sides.*

4.1.2 Protection-Based Reading with Ego-Lane Digests

The analysis as presented thus far does not take thread *id*s into account, and, e.g., also reads values from threads that have definitely not been started yet.

To remedy this, it would be desirable to modify the analysis to take digests into account and then plug in the thread *id* digest from Section 2.8. However, the unknowns used by this analysis differ from the ones used in the concrete semantics: For example, the analysis does not incorporate any information when a mutex is locked, and instead reads the values of global variables only upon access to a global variable by consulting dedicated unknowns. Therefore, refinement of this analysis can only be performed with *ego-lane* digests, which allow taking this deviation from the concrete semantics into account by giving an appropriate definition of $\mathsf{compat}^\sharp_{\mathcal{A}}$. In this way, the refined analysis can be proven sound for any admissible ego-lane digest.

When applying a refinement based on ego-lane digests to the analysis from Section 4.1.1, the resulting analysis has the following set of unknowns:

- $[u, S, A]$ for $u \in \mathcal{N}$, $S \subseteq \mathcal{M}$ and $A \in \mathcal{A}$,
- $[g, A]$ and $[g, A]'$ for $g \in \mathcal{G}$ and $A \in \mathcal{A}$,
- $[i, A]$ for $i \in S_{\mathcal{V}^\sharp_{\mathsf{tid}}}$ and $A \in \mathcal{A}$, and
- $[s, A]$ for $s \in \mathcal{S}$ and $A \in \mathcal{A}$.

The constraint system then takes the following form:

$$
\begin{aligned}
&[u_0, \emptyset, A] && \sqsupseteq \quad \mathsf{init}(A)^\sharp \\
&\quad \text{for } A \in \mathsf{init}^\sharp_{\mathcal{A}} \\
&[u', S, A'] && \sqsupseteq \quad [\![[u, S, A_0], x = \mathsf{create}(u_1)]\!]^\sharp \\
&\quad \text{for } (u, x = \mathsf{create}(u_1), u') \in \mathcal{E}, A' \in [\![u, x = \mathsf{create}(u_1)]\!]^\sharp_{\mathcal{A}}(A_0) \\
&[u', S \cup \{a\}, A'] && \sqsupseteq \quad [\![[u, S, A_0], \mathsf{lock}(a)]\!]^\sharp \\
&\quad \text{for } (u, \mathsf{lock}(a), u') \in \mathcal{E}, A' \in \textstyle\bigcup_{A_1 \in \mathcal{A}} \{[\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}(A_0, A_1)\} \\
&[u', S, A'] && \sqsupseteq \quad [\![[u, S, A_0], \mathsf{act}]\!]^\sharp \\
&\quad \text{for } (u, \mathsf{act}, u') \in \mathcal{E}, \mathsf{act} \in (\mathcal{A}ct_{\mathsf{observing}} \setminus \{\mathsf{lock}(a) \mid a \in \mathcal{M}\}), \\
&\qquad A' \in \textstyle\bigcup_{A_1 \in \mathcal{A}} \{[\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0, A_1)\} \\
&[u', S \setminus \{a\}, A'] && \sqsupseteq \quad [\![[u, S, A_0], \mathsf{unlock}(a), A']\!]^\sharp \\
&\quad \text{for } (u, \mathsf{unlock}(a), u') \in \mathcal{E}, A' \in [\![u, \mathsf{unlock}(a)]\!]^\sharp_{\mathcal{A}}(A_0) \\
&[u', S, A'] && \sqsupseteq \quad [\![[u, S, A_0], \mathsf{act}, A']\!]^\sharp \\
&\quad \text{for } (u, \mathsf{act}, u') \in \mathcal{E}, \mathsf{act} \in (\mathcal{A}ct_{\mathsf{observable}} \setminus \{\mathsf{unlock}(a) \mid a \in \mathcal{M}\}), \\
&\qquad A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0) \\
&[u', S, A'] && \sqsupseteq \quad [\![[u, S, A_0], \mathsf{act}]\!]^\sharp \\
&\quad \text{for } (u, \mathsf{act}, u') \in \mathcal{E}, \mathsf{act} \in \mathcal{A}ct_{\mathsf{local}}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0)
\end{aligned}
\tag{4.1}
$$

We remark that as the digests here are ego-lane-derived, by Eq. (2.16) from Section 2.9, the sets $\bigcup_{A_1 \in \mathcal{A}} \{[\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0, A_1)\}$ for an observable action act and a digest A_0 of the ego thread contain at most one element.
The right-hand sides for the initial constraints and for thread creation are then given by

$$\begin{aligned}
\mathsf{init}(A)^\sharp_ \;=\; & \textbf{let } \sigma = \{x \mapsto \top \mid x \in \mathcal{X}\} \cup \{g \mapsto [\![0]\!]^\sharp_{\mathcal{E}xp}\, \top \mid g \in \mathcal{G}\} \textbf{ in} \\
& \textbf{let } \rho = \{[g, A] \mapsto [\![0]\!]^\sharp_{\mathcal{E}xp}\, \top \mid g \in \mathcal{G}\} \cup \{[g, A]' \mapsto [\![0]\!]^\sharp_{\mathcal{E}xp}\, \top \mid g \in \mathcal{G}\} \textbf{ in} \\
& (\emptyset, (\emptyset, \sigma \oplus \{\mathsf{self} \mapsto [\![i_0]\!]^\sharp_{\mathcal{E}xp}\, \top\}))
\end{aligned}$$

$$\begin{aligned}
& [\![[u, S, A_0], x = \mathsf{create}(u_1)]\!]^\sharp \eta = \\
& \quad \textbf{let } (P, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& \quad \textbf{let } i = v^\sharp\, u\, \sigma\, u_1 \textbf{ in} \\
& \quad \textbf{let } \sigma' = \sigma \oplus (\{\mathsf{self} \mapsto i\} \cup \{g \mapsto [\![0]\!]^\sharp_{\mathcal{E}xp}\, \top \mid g \in \mathcal{G}\}) \textbf{ in} \\
& \quad \textbf{let } \rho = \{[u_1, \emptyset, A'] \mapsto (\emptyset, \sigma') \mid A' \in \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0\} \textbf{ in} \\
& \quad (\rho, (P, \sigma \oplus \{x \mapsto i\}))
\end{aligned}$$

The right-hand sides corresponding to initMT, lock, assignment to a global variable, and guards and computations on locals remain unchanged when compared to their definitions from Section 4.1.1 — except that they now consult the unknown enhanced with the digest.

For unlocking of mutexes, the new right-hand sides are given by

$$\begin{aligned}
& [\![[u, S, A_0], \mathsf{unlock}(m_g), A']\!]^\sharp \eta = \\
& \quad \textbf{let } (P, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& \quad \textbf{let } P' = \{h \in P \mid ((S \setminus \{m_g\}) \cap \bar{\mathcal{M}}[h]) \neq \emptyset\} \textbf{ in} \\
& \quad \textbf{let } \rho = \{[g, A']' \mapsto \sigma\, g\} \cup \{[g, A'] \mapsto \sigma\, g \mid \bar{\mathcal{M}}[g] = \{m_g\}\} \textbf{ in} \\
& \quad (\rho, (P', \sigma))
\end{aligned}$$

$$\begin{aligned}
& [\![[u, S, A_0], \mathsf{unlock}(a), A']\!]^\sharp \eta = \\
& \quad \textbf{let } (P, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& \quad \textbf{let } P' = \{g \in P \mid ((S \setminus \{a\}) \cap \bar{\mathcal{M}}[g]) \neq \emptyset\} \textbf{ in} \\
& \quad \textbf{let } \rho = \{[g, A'] \mapsto \sigma\, g \mid a \in \bar{\mathcal{M}}[g]\} \textbf{ in} \\
& \quad (\rho, (P', \sigma))
\end{aligned}$$

for $a \notin \{m_g \mid g \in \mathcal{G}\}$. For reading from a global, on the other hand, we have

$$\begin{aligned}
[\![[u, S, A_0], x = g]\!]^\sharp \eta \;=\; & \textbf{let } (P, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& \textbf{if } g \in P \textbf{ then} \\
& \qquad (\emptyset, (P, \sigma \oplus \{x \mapsto (\sigma\, g)\})) \\
& \textbf{else if } S \cap \bar{\mathcal{M}}[g] = \{m_g\} \textbf{ then} \\
& \qquad (\emptyset, (P, \sigma \oplus \{x \mapsto \sigma\, g \sqcup \bigsqcup_{A' \in \mathcal{A},\, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'} \eta\,[g, A']'\})) \\
& \textbf{else } (\emptyset, (P, \sigma \oplus \{x \mapsto \sigma\, g \sqcup \bigsqcup_{A' \in \mathcal{A},\, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'} \eta\,[g, A']\}))
\end{aligned}$$

For return, thread join, signal(s), and wait(s), we define

$$
\begin{array}{lcl}
[\![[u,S,A_0],\mathsf{return},A']\!]^\sharp\eta & = & \textbf{let } (P,\sigma) = \eta\,[u,S,A_0] \textbf{ in} \\
& & \textbf{let } I = \sigma\,\mathsf{self} \textbf{ in} \\
& & (\{[i,A'] \mapsto \sigma\,\mathsf{ret} \mid i \in I\}, (P,\sigma)) \\
[\![[u,S,A_0],x = \mathsf{join}(x')]\!]^\sharp\eta & = & \textbf{let } (P,\sigma) = \eta\,[u,S,A_0] \textbf{ in} \\
& & \textbf{let } v = \bigsqcup_{i' \in (\sigma\,x')} \left(\bigsqcup_{A' \in \mathcal{A},\, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A'} (\eta[i',A'])\right) \textbf{ in} \\
& & \textbf{if } v = \bot \textbf{ then} \\
& & \quad (\emptyset, \bot) \\
& & \textbf{else} \\
& & \quad (\emptyset, (P, \sigma \oplus \{x \mapsto v\})) \\
[\![[u,S,A_0],\mathsf{signal}(s),A']\!]^\sharp\eta & = & \textbf{let } (P,\sigma) = \eta\,[u,S,A_0] \textbf{ in} \\
& & (\{[s,A'] \mapsto (P,\sigma)\}, (P,\sigma)) \\
[\![[u,S,A_0],\mathsf{wait}(s)]\!]^\sharp\eta & = & \textbf{let } (P,\sigma) = \eta\,[u,S,A_0] \textbf{ in} \\
& & \textbf{if } \left(\left(\bigsqcup_{A' \in \mathcal{A},\, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A'} \eta\,[s,A']\right) = \bot\right) \textbf{ then} \\
& & \quad (\emptyset, \bot) \\
& & \textbf{else} \\
& & \quad (\emptyset, (P,\sigma))
\end{array}
$$

Thus, side-effects are re-directed to unknowns equipped with digests, and the function $\mathsf{compat}^\sharp_{\mathcal{A}}$ is used to decide when information should be incorporated into the analysis.

Example 16. *Consider the analysis refined with the ego-lane-derived thread* id *digest from Section 2.8, with* $\mathsf{compat}^\sharp_{\mathcal{A}}(i,C)(i,C') = \mathsf{may\_run}\,(i,C)(i,C')$ *as proposed in Example 10. Consider additionally the following program fragment and assume that* $\bar{\mathcal{M}}[g] = \{a, m_g\}$ *and that we use value sets for abstracting int values.*

```
main:                t1:              t2:
  initMT;              lock(a);         lock(a);
  x = create(t1);      lock(m_g);       lock(m_g);
  lock(a);             g = 1;           g = 42;
  lock(m_g);           unlock(m_g);     unlock(m_g);
  z = g;               unlock(a);       unlock(a);
  unlock(m_g);         return;          return;
  unlock(a);
  y = create(t2);
  return;
```

The main thread here receives the thread id $\bar{\mathsf{main}} = (\mathsf{main}, \emptyset)$*, and the newly created threads receive the thread* ids $\bar{t}_1 = (\mathsf{main} \cdot \langle u_1, t_1\rangle, \emptyset)$ *and* $\bar{t}_2 = (\mathsf{main} \cdot \langle u_8, t_2\rangle, \emptyset)$*, respectively. As the read of* g *in the main thread is protected, the values of unknowns* $[g,A]$ *are relevant here. At the* $\mathsf{unlock}(a)$ *in thread* $\bar{t}_1$*, the unknown* $[g,(\bar{t}_1,\emptyset)]$ *receives the contribution* $\{1\}$*, whereas at the* $\mathsf{unlock}(a)$ *in thread* $\bar{t}_2$*, the unknown* $[g,(\bar{t}_2,\emptyset)]$ *receives the contribution* $\{42\}$*. At the read*

in $\bar{\text{main}}$*, the local state (when omitting the values of variables of type thread* id *for readability) is* $(P, \sigma) = (\emptyset, \{g \mapsto \{0\}\})$ *and the digest information associated with the preceding program point is* $(\bar{\text{main}}, \{\langle u_1, t_1 \rangle\})$*. Thus, the value read for* g *is given by*

$$\begin{aligned} d &= \sigma\, g \sqcup \bigsqcup_{A' \in \mathcal{A}, \mathit{compat}^\sharp_{\mathcal{A}}\,(\bar{\text{main}}, \{\langle u_1, t_1 \rangle\})\; A'} \eta\, [g, A'] \\ &= \sigma\, g \sqcup \bigsqcup_{A' \in \mathcal{A}, \mathit{may\_run}\,(\bar{\text{main}}, \{\langle u_1, t_1 \rangle\})\; A'} \eta\, [g, A'] \\ &= \{0\} \sqcup \eta\, [g, (\bar{\text{main}}, \emptyset)] \sqcup \eta\, [g, (\bar{t}_1, \emptyset)] \\ &= \{0\} \sqcup \{0\} \sqcup \{1\} = \{0, 1\} \end{aligned}$$

as the only unknowns for protected accesses to g *that receive side-effects are* $[g, (\bar{\text{main}}, \emptyset)]$ *(initialization),* $[g, (\bar{t}_1, \emptyset)]$*, and* $[g, (\bar{t}_2, \emptyset)]$ *and* $\mathit{may\_run}\,(\bar{\text{main}}, \{\langle u_1, t_1 \rangle\})\; (\bar{t}_1, \emptyset)$ *as well as* $\mathit{may\_run}\,(\bar{\text{main}}, \{\langle u_1, t_1 \rangle\})\; (\bar{\text{main}}, \emptyset)$ *are true, while* $\mathit{may\_run}\,(\bar{\text{main}}, \{\langle u_1, t_1 \rangle\})\; (\bar{t}_2, \emptyset)$ *is not. The analysis without this refinement, on the other hand, would find that the value read for* g *is* $\{0, 1, 42\}$*.* □

Remark 11. *By choosing* $\mathcal{A} = \{\bullet\}$ *and setting all right-hand sides for digests to also return* $\{\bullet\}$ *(which is ego-lane-derived), we obtain an analysis equivalent to the one from Section 4.1.1.*

Theorem 6. Protection-Based Reading *enhanced with an ego-lane digest is sound w.r.t. the local trace semantics.*

Proof. In Section 6.1.3, we show that this analysis computes an abstraction of the result of the analysis presented in Section 4.1.7, which we then prove to be sound w.r.t. the local trace semantics in Section 6.1.1. □

4.1.3 Side-Effecting Formulation of the Analysis by Miné

As the analysis in Section 4.1.4 will attempt to improve upon a variant of the analysis by Miné [83, 84], we will first give a side-effecting formulation of this analysis adapted to our setting here. This analysis, when stripped of pre-determined thread *id*s, which are not readily available in a setting such as the one considered here, and other features specific to real-time systems such as ARINC653 [2] and reformulated by means of side-effecting constraint systems, works as follows:

It maintains for each pair (u, S) of program point u and currently held lockset S, copies of globals g whose values are weakly updated whenever the lock for some mutex a is acquired. In order to restrict the set of possibly read values, the global g is split into unknowns $[g, a, S']$ where S' is a *background* lockset held by another thread immediately after executing $\mathsf{unlock}(a)$. Then only the values of those unknowns $[g, a, S']$ are taken into account where $S \cap S' = \emptyset$.

On top of the mechanism that handles synchronized accesses to variables (*synchronized interferences* in Miné's terminology), there also exist *weak interferences*, i.e., accesses not synchronized via some common mutex in his original setting. After adaption to our setting, such weak interferences do not exist because the atomicity assumption introduced mutexes m_g immediately surrounding each access to a global g. The would-be weak interferences for a global g thus are stored at unknowns $[g, m_g, S]$. To be faithful

to the analysis as proposed, where such weak interferences are only consulted at the read and not eagerly copied into the local state, locking and unlocking some m_g for $g \in \mathcal{G}$ does not affect the local state and the values stored at unknowns $[g, m_g, S]$ are instead taken into account and side-effected to when reading from, respectively when writing to, a global g. We also track a set W of written variables by which we restrict synchronized interferences, as is done with the help of the weak interferences of a thread with a given thread *id* in the original setting. We extend this analysis with treatment for return, join, and signal and wait in the same manner these features are handled in the other analyses and remark that these are not present in the original setting. The right-hand-side functions thus are defined as follows:

$$
\begin{aligned}
&\mathsf{init}^\sharp\ \_ = \\
&\quad \textbf{let } \sigma = \{x \mapsto \top \mid x \in \mathcal{X}\} \cup \{g \mapsto [\![0]\!]^\sharp_{\mathcal{E}xp}\ \top \mid g \in \mathcal{G}\} \textbf{ in} \\
&\quad (\emptyset, (\emptyset, \sigma)) \\
&[\![[u, S], x = \mathsf{create}(u_1)]\!]^\sharp \eta = \\
&\quad \textbf{let } (W, \sigma) = \eta\,[u, S] \textbf{ in} \\
&\quad \textbf{let } i = \nu^\sharp\, u\, \sigma\, u_1 \textbf{ in} \\
&\quad \textbf{let } \sigma' = \sigma \oplus \left(\{\mathsf{self} \mapsto i\} \cup \left\{g \mapsto \left(\sigma\, g \sqcap [\![0]\!]^\sharp_{\mathcal{E}xp}\ \top\right) \mid g \in \mathcal{G}\right\}\right) \textbf{ in} \\
&\quad \textbf{let } \rho = \{[u_1, \emptyset] \mapsto (\emptyset, \sigma')\} \textbf{ in} \\
&\quad (\rho, (W, \sigma \oplus \{x \mapsto i\})) \\
&[\![[u, S], \mathsf{lock}(m_g)]\!]^\sharp \eta = (\emptyset, \eta\,[u, S]) \\
&[\![[u, S], \mathsf{unlock}(m_g)]\!]^\sharp \eta = (\emptyset, \eta\,[u, S]) \\
&[\![[u, S], \mathsf{lock}(a)]\!]^\sharp \eta = \\
&\quad \textbf{let } (W, \sigma) = \eta\,[u, S] \textbf{ in} \\
&\quad \textbf{let } \sigma' = \{g \mapsto \bigsqcup\{\eta[g, a, S'] \mid S' \subseteq \mathcal{M}, S' \cap S = \emptyset\} \mid g \in \mathcal{G}\} \textbf{ in} \\
&\quad (\emptyset, (W, \sigma \sqcup \sigma')) \\
&[\![[u, S], \mathsf{unlock}(a)]\!]^\sharp \eta = \\
&\quad \textbf{let } (W, \sigma) = \eta\,[u, S] \textbf{ in} \\
&\quad (\{[g, a, S \setminus \{a\}] \mapsto \sigma\, g \mid g \in W\}, (W, \sigma)) \\
&[\![[u, S], g = x]\!]^\sharp \eta \\
&\quad \textbf{let } (W, \sigma) = \eta\,[u, S] \textbf{ in} \\
&\quad \textbf{let } \sigma' = \sigma \oplus \{g \mapsto \sigma\, x\} \textbf{ in} \\
&\quad (\{[g, m_g, S \setminus \{m_g\}] \mapsto \sigma'\, g\}, (W \cup \{g\}, \sigma')) \\
&[\![[u, S], x = g]\!]^\sharp \eta \\
&\quad \textbf{let } (W, \sigma) = \eta\,[u, S] \textbf{ in} \\
&\quad \textbf{let } g' = \bigsqcup\{\eta\,[g, m_g, S'] \mid S' \subseteq \mathcal{M}, S' \cap S = \emptyset\} \textbf{ in} \\
&\quad (\emptyset, (W, \sigma \oplus \{x \mapsto \sigma\, g \sqcup g'\}))
\end{aligned}
$$

Once again, the right-hand side for the action initMT returns the abstract state of the predecessor and causes no side-effects. Also, computations on locals and guards are

defined in the intuitive manner, operating on σ only, leaving W unchanged. Return, join, signal, and wait are handled in the same manner as in the Protection-Based analysis (Section 4.1.1), when replacing the unmodified P component there with W.

This analysis is intricate; however, side-effecting constraint systems elegantly capture the core idea in just a few lines. The weak interferences are associated with pseudo-lock m_g, but a weak interference is only propagated from a write with lockset S to a read with lockset S' if these sets have an empty intersection. Similarly, synchronized interferences are only propagated from an unlock to a lock if the background locksets permit it.

Remark 12. *The limiting of synchronized interferences in the extended version [84] happens by checking whether a weak interference of the same thread* id *exists where the given mutex is in the background lockset. The author remarks that, to gain precision, one could track for each critical section which variables were modified during it and then only emit synchronized interferences for such variables. In our setting with dynamic thread creation, however, no thread* ids *are readily available. The set W is thus introduced to locally track sets of globals modified since the start of the thread to somewhat limit synchronized interferences and stay true in spirit to the original analysis. Interestingly, this is in some way more precise than the original setting, as W is tracked flow-sensitively, whereas checking the existence of a weak interference is flow-insensitive. On the other hand, an even more faithful rendering would instead require tracking maps from mutexes to those globals that have been potentially modified since the mutex has been acquired, and would then be in line with the additional remark in [84]. While one can construct examples where tracking this information more precisely improves precision, it does not do so for any of the example programs presented in this thesis, and our implementation does not support it.*

4.1.4 Lock-Centered Reading

We identify two sources of imprecision in the variant of the analysis by Miné [83, 84] adapted to our setting as presented in the previous section. One source is *eager reading*, i.e., reading in values of g at every $\mathsf{lock}(a)$ operation. This may import the values of *too many* unknowns $[g, a, S']$ into the local state. Instead, it suffices for each mutex a, to read values at the *last* $\mathsf{lock}(a)$ before actually accessing the global.

Let $\mathcal{U}_\mathcal{M}$ denote the set of all upward-closed subsets of $\mathcal{M}$, ordered by subset inclusion. For convenience, we represent each non-empty value in $\mathcal{U}_\mathcal{M}$ by the set of its minimal elements. Thus, the *least* element of $\mathcal{U}_\mathcal{M}$ is $\emptyset$, while the *greatest* element is given by the *full* power set of mutexes (represented by $\{\emptyset\}$).

We now maintain a map $L : \mathcal{M} \to \mathcal{U}_\mathcal{M}$ in the local state that tracks for each mutex a all minimal background locksets that were held when a was acquired last. This abstraction of acquisition histories [66, 67] allows us to delay the reading of globals until the point where the program actually accesses their values. We call this behavior *lazy reading*.

The other source of imprecision is that each thread may publish values it has not written itself. In order to address this issue, we modify the map component of the local state to, for each global g, only maintain an abstraction $\sigma\, g$ of values the ego thread itself has written and then publish only those values.

A consequence of *lazy reading* is that values for globals are now read from the global invariant at each read. In case the ego thread has definitely written to a variable and no additional locks have occurred since, only the local copy needs to be read. To achieve that, we introduce an additional map $V : \mathcal{M} \to 2^{\mathcal{G}}$ with the order defined point-wise by superset. For a mutex $a \in \mathcal{M}$, $V\,a \subseteq \mathcal{G}$ is the set of global variables that were *definitely* written since a was last acquired by the *ego* thread. In case a has never been acquired by the ego thread, we set $V\,a$ to the set of all global variables that have definitely been written since the start of the thread.

We start by giving the right-hand-side function for the start state at program point $u_0 \in \mathcal{N}$ with the empty lockset $\emptyset$, i.e., $[u_0, \emptyset] \sqsupseteq \mathsf{init}^\sharp$ where

$$\begin{aligned}\mathsf{init}^\sharp\,\_ \;=\; & \textbf{let } V = \{a \mapsto \emptyset \mid a \in \mathcal{M}\} \textbf{ in} \\ & \textbf{let } L = \{a \mapsto \emptyset \mid a \in \mathcal{M}\} \textbf{ in} \\ & \textbf{let } \sigma = \{x \mapsto \top \mid x \in \mathcal{X}\} \cup \{g \mapsto [\![0]\!]^\sharp_{\mathcal{E}xp}\,\top \mid g \in \mathcal{G}\} \textbf{ in} \\ & (\emptyset, (V, L, \sigma \oplus \{\mathsf{self} \mapsto [\![i_0]\!]^\sharp_{\mathcal{E}xp}\,\top\}))\end{aligned}$$

Next, we sketch the right-hand-side function for a thread creation edge.

$$\begin{aligned}&[\![[u, S], x = \mathsf{create}(u_1)]\!]^\sharp\eta = \\ &\quad \textbf{let } (V, L, \sigma) = \eta\,[u, S] \textbf{ in} \\ &\quad \textbf{let } i = v^\sharp\,u\,\sigma\,u_1 \textbf{ in} \\ &\quad \textbf{let } \sigma' = \sigma \oplus \left(\{\mathsf{self} \mapsto i\} \cup \left\{g \mapsto \left(\sigma\,g \sqcap [\![0]\!]^\sharp_{\mathcal{E}xp}\,\top\right) \mid g \in \mathcal{G}\right\}\right) \textbf{ in} \\ &\quad \textbf{let } \rho = \{[u_1, \emptyset] \mapsto (\{a \mapsto \emptyset \mid a \in \mathcal{M}\}, \{a \mapsto \emptyset \mid a \in \mathcal{M}\}, \sigma')\} \textbf{ in} \\ &\quad (\rho, (V, L, \sigma \oplus \{x \mapsto i\}))\end{aligned}$$

This function has no effect on the local state apart from setting x to the abstract thread *id* of the newly created thread, while providing an appropriate initial state to the start point of the newly created thread.

Remark 13. *We remark that setting g to $\left(\sigma\,g \sqcap [\![0]\!]^\sharp_{\mathcal{E}xp}\,\top\right)$ in the initial state of the new thread is done here only for symmetry reasons with the other analyses. Setting all such globals in σ' to $\bot$ would also be sound.*

The right-hand side for the action initMT once more returns the abstract state of the predecessor and causes no side-effects. For guards and computations on locals, the right-hand-side functions are once more defined in a straightforward manner — they operate on σ only, leaving L and V unchanged.

Locking a mutex a resets $V\,a$ to $\emptyset$ and updates L, whereas unlock side-effects the value of globals to the appropriate unknowns.

$$\begin{aligned}[\![[u, S], \mathsf{lock}(a)]\!]^\sharp\eta \;=\; & \textbf{let } (V, L, \sigma) = \eta\,[u, S] \textbf{ in} \\ & \textbf{let } V' = V \oplus \{a \mapsto \emptyset\} \textbf{ in} \\ & \textbf{let } L' = L \oplus \{a \mapsto \{S\}\} \textbf{ in} \\ & (\emptyset, (V', L', \sigma))\end{aligned}$$

$$\begin{aligned}[[u, S], \mathsf{unlock}(a)]^\sharp \eta &= \textbf{let } (V, L, \sigma) = \eta\,[u, S] \textbf{ in} \\ &\quad \textbf{let } \rho = \{[g, a, S \setminus \{a\}] \mapsto \sigma\, g \mid g \in \mathcal{G}\} \textbf{ in} \\ &\quad (\rho, (V, L, \sigma))\end{aligned}$$

The right-hand-side function for an edge corresponding to a write to a global g then consists of a strong update to the local copy and addition of g to $V\,a$ for all mutexes a. For reading from a global g, those values $[g, a, S']$ need to be taken into account where a is one of the mutexes acquired in the past of the *ego* thread, and the intersection of some set in $L\,a$ with the set of mutexes S' held while publishing is empty.

$$\begin{aligned}[[u, S], g = x]^\sharp \eta &= \textbf{let } (V, L, \sigma) = \eta\,[u, S] \textbf{ in} \\ &\quad \textbf{let } V' = \{a \mapsto (V\,a \cup \{g\}) \mid a \in \mathcal{M}\} \textbf{ in} \\ &\quad (\varnothing, (V', L, \sigma \oplus \{g \mapsto (\sigma\, x)\})) \\ [[u, S], x = g]^\sharp \eta &= \textbf{let } (V, L, \sigma) = \eta\,[u, S] \textbf{ in} \\ &\quad \textbf{let } d = \bigsqcup\{\eta[g, a, S'] \mid a \in \mathcal{M}, g \notin V\,a, \exists B \in L\,a, B \cap S' = \varnothing\} \textbf{ in} \\ &\quad (\varnothing, (V, L, \sigma \oplus \{x \mapsto \sigma\, g \sqcup d\}))\end{aligned}$$

In case that $L\,a = \varnothing$, i.e., if, according to the analysis, no thread reaching u with lockset S has ever locked mutex a, then no values from $[g, a, S']$ will be read.

The handling of returns, joins, signals, and waits remains essentially unchanged compared to the analysis from Section 4.1.1.

$$\begin{aligned}[[u, S], \mathsf{return}]^\sharp \eta &= \textbf{let } (V, L, \sigma) = \eta\,[u, S] \textbf{ in} \\ &\quad \textbf{let } I = \sigma\,\mathsf{self} \textbf{ in} \\ &\quad (\{[i] \mapsto \sigma\,\mathsf{ret} \mid i \in I\}, (V, L, \sigma)) \\ [[u, S], x = \mathsf{join}(x')]^\sharp \eta &= \textbf{let } (V, L, \sigma) = \eta\,[u, S] \textbf{ in} \\ &\quad \textbf{let } v = \textstyle\bigsqcup_{i' \in (\sigma\, x')}(\eta[i']) \textbf{ in} \\ &\quad \textbf{if } v = \bot \textbf{ then} \\ &\qquad (\varnothing, \bot) \\ &\quad \textbf{else} \\ &\qquad (\varnothing, (V, L, \sigma \oplus \{x \mapsto v\})) \\ [[u, S], \mathsf{signal}(s)]^\sharp \eta &= \textbf{let } (V, L, \sigma) = \eta\,[u, S] \textbf{ in} \\ &\quad (\{[s] \mapsto (V, L, \sigma)\}, (V, L, \sigma)) \\ [[u, S], \mathsf{wait}(s)]^\sharp \eta &= \textbf{let } (V, L, \sigma) = \eta\,[u, S] \textbf{ in} \\ &\quad \textbf{if } \eta\,[s] = \bot \textbf{ then} \\ &\qquad (\varnothing, \bot) \\ &\quad \textbf{else} \\ &\qquad (\varnothing, (V, L, \sigma))\end{aligned}$$

The following example highlights core aspects of this analysis, and how *eager reading* which was identified as a source of imprecision for the analysis in the previous section is avoided here.

Example 17. *Consider the following program fragment and assume that sets of integers are used to abstract variables of type int.*

```
main:
    initMT;
    x = create(t1);
    lock(b);
    unlock(b);
    lock(a);
    lock(b);
    lock(m_g); z = g; unlock(m_g);
    // ASSERT(z>=0) (1)
    unlock(b);
    unlock(a);
    return;
```

```
t1:
    lock(a);
    lock(b);
    lock(m_g); g = -1; unlock(m_g);
    unlock(b);
    lock(m_g); z = g; unlock(m_g);
    // ASSERT(z==-1) (2)
    lock(m_g); g = 8; unlock(m_g);
    unlock(a);
    return;
```

With Lock-Centered Reading, *both assertions can be shown, whereas the analysis from Section 4.1.3 based on [83, 84] only succeeds in showing the assertion (2). In this analysis, upon the first* lock(b) *in the main thread, the value* $\{0, -1\}$ *associated with the unknown* $[g, b, \{a\}]$ *is joined into the local state. Thus, the assertion (1) will fail. For* Lock-Centered Reading, *no information is incorporated right away upon the call to* lock(b). *When the global* g *is accessed in the main thread, the set of minimal locksets since acquiring it (*$L\,b$*) is* $\{\{a\}\}$, *and the unknown* $[g, b, \{a\}]$ *is thus not consulted. The unknowns* $[g, a, \emptyset]$ *and* $[g, b, \emptyset]$ *which* are *consulted have the values* $\{0, 8\}$ *and* $\{0\}$, *respectively. The assertion (1) therefore can be established.*

For assertion (2), on the other hand, the data structure V *(which for mutex* a *contains* g *when* g *is read in* t_1*) is crucial to exclude reading other values, e.g., from* $[g, a, \emptyset]$. *The analysis from Section 4.1.3 does not require this extra data structure.* □

Theorem 7. Lock-Centered Reading *is sound w.r.t. the local trace semantics.*

Proof. In Section 4.1.5, we show this analysis is an instance of the refined analysis presented there. The soundness proof for that analysis is deferred to Section 6.1.2. □

4.1.5 Lock-Centered Reading with Ego-Lane Digests

We now turn to enhancing this analysis with digests. When applying a refinement based on ego-lane digests to the analysis from Section 4.1.4, the resulting analysis has the following set of unknowns:

- $[u, S, A]$ for $u \in \mathcal{N}$, $S \subseteq \mathcal{M}$, and $A \in \mathcal{A}$,
- $[g, a, S, A]$ for $g \in \mathcal{G}, a \in \mathcal{M}, S \subseteq \mathcal{M}$, and $A \in \mathcal{A}$,
- $[i, A]$ for $i \in S_{\mathcal{V}^\sharp_{\mathbf{tid}}}$ and $A \in \mathcal{A}$, and
- $[s, A]$ for $s \in \mathcal{S}$ and $A \in \mathcal{A}$.

The resulting constraint system here takes the same form as the constraint system for the protection-based analysis enhanced with digests, as given in Eq. (4.1).

The right-hand sides are then given by:

$$
\begin{aligned}
&\mathsf{init}(A)^\sharp\ \_ = \\
&\quad \textbf{let } V = \{a \mapsto \emptyset \mid a \in \mathcal{M}\} \textbf{ in} \\
&\quad \textbf{let } L = \{a \mapsto \emptyset \mid a \in \mathcal{M}\} \textbf{ in} \\
&\quad \textbf{let } \sigma = \{x \mapsto \top \mid x \in \mathcal{X}\} \cup \{g \mapsto [\![0]\!]^\sharp_{\mathcal{E}xp}\ \top \mid g \in \mathcal{G}\} \textbf{ in} \\
&\quad (\emptyset, (V, L, \sigma \oplus \{\mathsf{self} \mapsto [\![i_0]\!]^\sharp_{\mathcal{E}xp}\ \top\})) \\
&[\![[u, S, A_0], x = \mathsf{create}(u_1)]\!]^\sharp \eta = \\
&\quad \textbf{let } (V, L, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
&\quad \textbf{let } V' = \{a \mapsto \emptyset \mid a \in \mathcal{M}\} \textbf{ in} \\
&\quad \textbf{let } L' = \{a \mapsto \emptyset \mid a \in \mathcal{M}\} \textbf{ in} \\
&\quad \textbf{let } i = v^\sharp\ u\ \sigma\ u_1 \textbf{ in} \\
&\quad \textbf{let } \sigma' = \sigma \oplus \left(\{\mathsf{self} \mapsto i\} \cup \left\{g \mapsto \left(\sigma\, g \sqcap [\![0]\!]^\sharp_{\mathcal{E}xp}\ \top\right) \mid g \in \mathcal{G}\right\}\right) \textbf{ in} \\
&\quad \textbf{let } \rho = \{[u_1, \emptyset, A'] \mapsto (V', L', \sigma') \mid A' \in \mathsf{new}^\sharp_{\mathcal{A}}\ u\ u_1\ A_0\} \textbf{ in} \\
&\quad (\rho, (V, L, \sigma \oplus \{x \mapsto i\}))
\end{aligned}
$$

The right-hand sides for initMT, lock, assignment to a global variable, and guards and computations on locals remain unchanged when compared to their definitions from Section 4.1.4 — except that they now consult the unknown enhanced with the digest. For unlocking of mutexes, the new right-hand side is given by:

$$
\begin{aligned}
[\![[u, S, A_0], \mathsf{unlock}(a), A']\!]^\sharp \eta \quad = \quad & \textbf{let } (V, L, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& \textbf{let } \rho = \{[g, a, S \setminus \{a\}, A'] \mapsto \sigma\, g \mid g \in \mathcal{G}\} \textbf{ in} \\
& (\rho, (V, L, \sigma))
\end{aligned}
$$

For reading from a global, we now have

$$
\begin{aligned}
&[\![[u, S, A_0], x = g]\!]^\sharp \eta = \\
&\quad \textbf{let } (V, L, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
&\quad \textbf{let } d = \bigsqcup\{\eta[g, a, S', A'] \mid a \in \mathcal{M}, \\
&\qquad\qquad g \notin V\,a, \exists B \in L\,a, B \cap S' = \emptyset, A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\ A_0\ A'\} \textbf{ in} \\
&\quad (\emptyset, (V, L, \sigma \oplus \{x \mapsto \sigma\, g \sqcup d\}))
\end{aligned}
$$

The handling of returns, joins, signals, and waits once more remains essentially unchanged compared to the analysis from Section 4.1.2.

$$
\begin{aligned}
[\![[u, S, A_0], \mathsf{return}, A']\!]^\sharp \eta \quad = \quad & \textbf{let } (V, L, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& \textbf{let } I = \sigma\,\mathsf{self} \textbf{ in} \\
& (\{[i, A'] \mapsto \sigma\,\mathsf{ret} \mid i \in I\}, (V, L, \sigma))
\end{aligned}
$$

$$
\begin{array}{lcl}
[\![[u,S,A_0], x = \mathsf{join}(x')]\!]^\sharp\eta & = & \textbf{let } (V,L,\sigma) = \eta\,[u,S,A_0] \textbf{ in} \\
& & \textbf{let } v = \bigsqcup_{i' \in (\sigma\, x')} \left(\bigsqcup_{A' \in \mathcal{A},\, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'} (\eta[i', A'])\right) \textbf{ in} \\
& & \textbf{if } v = \bot \textbf{ then} \\
& & \quad (\varnothing, \bot) \\
& & \textbf{else} \\
& & \quad (\varnothing, (V, L, \sigma \oplus \{x \mapsto v\})) \\
[\![[u,S,A_0], \mathsf{signal}(s), A']\!]^\sharp\eta & = & \textbf{let } (V,L,\sigma) = \eta\,[u,S,A_0] \textbf{ in} \\
& & (\{[s,A'] \mapsto (V,L,\sigma)\}, (V,L,\sigma)) \\
[\![[u,S,A_0], \mathsf{wait}(s)]\!]^\sharp\eta & = & \textbf{let } (V,L,\sigma) = \eta\,[u,S,A_0] \textbf{ in} \\
& & \textbf{if } \left(\left(\bigsqcup_{A' \in \mathcal{A},\, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'} \eta\,[s,A']\right) = \bot\right) \textbf{ then} \\
& & \quad (\varnothing, \bot) \\
& & \textbf{else} \\
& & \quad (\varnothing, (V,L,\sigma))
\end{array}
$$

Thus, again side-effects are re-directed to the unknowns equipped with digests, and the function $\mathsf{compat}^\sharp_{\mathcal{A}}$ is used to decide when information should be incorporated into the analysis.

Example 18. *Consider the analysis refined with the ego-lane-derived thread* id *digest from Section 2.8, with* $\mathsf{compat}^\sharp_{\mathcal{A}}(i,C)(i,C') = \mathsf{may\_run}\,(i,C)(i,C')$ *as proposed in Example 10 and consider again the program from Example 16. The reasoning given there also applies to this analysis, i.e., it will also establish that the value read for* g *is* $\{0,1\}$. □

Example 19. *Alternatively, consider the analysis refined in the same way and the following program:*

```
main:                        t1:                      t2:
  initMT;                      y = create(t2);          lock(a);
  lock(m_g);                   lock(a);                 lock(m_g);
  g = 1;                       lock(m_g);               z = g;
  unlock(m_g);                 g = -8;                  unlock(m_g);
  lock(m_g);                   unlock(m_g);             // ASSERT(z == 1); (2)
  z = g;                       lock(m_g);               unlock(a);
  unlock(m_g);                 g = 1;                   return;
  // ASSERT(z == 1); (1)       unlock(m_g);
  x = create(t1);              unlock(a);
  return;                      return;
```

Without refinement, assertion (1) could not be established as writes from other threads would need to be considered. Conversely, assertion (2) is out of reach for the Protection-Based analysis, even when it is refined with thread ids*, as the global* g *is accessed without holding* a *before any other threads are started and thus* a *is not considered a protecting mutex there. This is a situation that is quite common where, e.g., the initialization of a global happens without holding any mutexes, as no other threads have been started yet.* □

Remark 14. *By choosing $\mathcal{A} = \{\bullet\}$ and setting all right-hand sides for digests to also return $\{\bullet\}$ (which is ego-lane-derived), one obtains an analysis equivalent to the one from Section 4.1.4.*

Theorem 8. Lock-Centered Reading *enhanced with an ego-lane digest is sound w.r.t. the local trace semantics.*

Proof. The detailed proof is deferred to Section 6.1.2. Roughly, it proceeds as follows: First, the constraint system for the concrete semantics is massaged into an equivalent form that can more readily be related to the abstract semantics. Then, the equivalence between both formulations is established (Proposition 21). After showing that restricting the unknowns consulted upon read as presented in the transfer functions will not miss any values (Proposition 22), solutions of the abstract constraint system are shown to give rise to solutions of the concrete constraint system (Theorem 18). □

4.1.6 Write-Centered Reading

In this section, we provide an improvement of *Protection-Based Reading* which lifts the assumption that each global g is write-protected by some fixed set of mutexes $\bar{\mathcal{M}}[g]$. In order to lift the assumption, we locally track some additional information about the history of the ego thread.

Recall that $\mathcal{U}_{\mathcal{M}}$ denotes the set of all upward-closed subsets of $\mathcal{M}$, ordered by subset inclusion and that we represent each non-empty value in $\mathcal{U}_{\mathcal{M}}$ by the set of its minimal elements. Further recall that the *least* element of $\mathcal{U}_{\mathcal{M}}$ is $\emptyset$, while the *greatest* element is the *full* power set (represented by $\{\emptyset\}$).

We then introduce the additional data structures $W, P : \mathcal{G} \to \mathcal{U}_{\mathcal{M}}$ to be maintained by the analysis for each unknown $[u, S]$ for program point u and currently held lockset S. The map W tracks for each global g the set of minimal locksets held when g was last written by the ego thread. At the start of a thread, the ego thread has not written any global yet; hence, we set $W\,g = \emptyset$ for all globals g. The map P, on the other hand, tracks, for each global g, all minimal locksets the ego thread has held since its last write to g. A global g not yet written to by the ego thread is mapped to the *full* power set of mutexes (represented by $\{\emptyset\}$). The unknowns for a global g now are of the form $[g, a, S, w]$ for mutexes a, background locksets S at $\mathsf{unlock}(a)$ and minimal lockset w when g was last written.

We start by giving the right-hand-side function for the start state at program point $u_0 \in \mathcal{N}$ with the empty lockset $\emptyset$, i.e., $[u_0, \emptyset] \sqsupseteq \mathsf{init}^\sharp$ where

$$
\begin{aligned}
\mathsf{init}^\sharp\ \_ \;=\; & \textbf{let } W = \{g \mapsto \emptyset \mid g \in \mathcal{G}\} \textbf{ in} \\
& \textbf{let } P = \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}\} \textbf{ in} \\
& \textbf{let } \sigma = \{x \mapsto \top \mid x \in \mathcal{X}\} \cup \{g \mapsto [\![0]\!]^\sharp_{\mathcal{E}xp}\ \top \mid g \in \mathcal{G}\} \textbf{ in} \\
& (\emptyset, (W, P, \sigma \oplus \{\mathsf{self} \mapsto [\![i_0]\!]^\sharp_{\mathcal{E}xp}\ \top\}))
\end{aligned}
$$

The right-hand side for the action initMT once more returns the abstract state of the predecessor and causes no side-effects. Next comes the right-hand-side function for a

thread creating edge.

$$
\begin{aligned}
&[\![[u, S], x = \mathsf{create}(u_1)]\!]^\sharp \eta = \\
&\quad \textbf{let } (W, P, \sigma) = \eta\,[u, S] \textbf{ in} \\
&\quad \textbf{let } W' = \{g \mapsto \emptyset \mid g \in \mathcal{G}\} \textbf{ in} \\
&\quad \textbf{let } P' = \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}\} \textbf{ in} \\
&\quad \textbf{let } i = \nu^\sharp\, u\, \sigma\, u_1 \textbf{ in} \\
&\quad \textbf{let } \sigma' = \sigma \oplus \left(\{\mathsf{self} \mapsto i\} \cup \left\{g \mapsto \left(\sigma\, g \sqcap [\![0]\!]^\sharp_{\mathcal{E}xp}\, \top\right) \mid g \in \mathcal{G}\right\}\right) \textbf{ in} \\
&\quad \textbf{let } \rho = \{[u_1, \emptyset] \mapsto (W', P', \sigma')\} \textbf{ in} \\
&\quad (\rho, (W, P, \sigma \oplus \{x \mapsto i\}))
\end{aligned}
$$

This function has no effect on the local state apart from setting x to the abstract thread *id* of the newly created thread while providing an appropriate initial state to the start point of the newly created thread.

Remark 15. *We remark that setting g to $\left(\sigma\, g \sqcap [\![0]\!]^\sharp_{\mathcal{E}xp}\, \top\right)$ in the initial state of the new thread is only necessary for technical reasons to ensure that this analysis is more precise than the Protection-Based Reading analysis presented in Section 4.1.1 in all circumstances. This is because this analysis does not side-effect the initial value of the globals right at the program start and thus needs to propagate this initial value to the newly created thread if g has indeed not been written yet. However, propagating all of $\sigma\, g$ could lead to a loss of precision, e.g., when g is only written in one branch.*

For guards and computations on locals, the right-hand-side functions are once more defined straightforwardly — they operate on σ only, leaving W and P unchanged.

While nothing happens at locking, unlocking now updates the data structure P and additionally side-effects the current local values for each global g to the corresponding unknowns. Note that no distinction between m_g and other mutexes is drawn here.

$$
\begin{aligned}
[\![[u, S], \mathsf{lock}(a)]\!]^\sharp \eta &= (\emptyset, \eta\,[u, S]) \\
[\![[u, S], \mathsf{unlock}(a)]\!]^\sharp \eta &= \textbf{let } (W, P, \sigma) = \eta\,[u, S] \textbf{ in} \\
&\quad \textbf{let } P' = \{g \mapsto P\, g \sqcup \{S \setminus \{a\}\} \mid g \in \mathcal{G}\} \textbf{ in} \\
&\quad \textbf{let } \rho = \{[g, a, S \setminus \{a\}, w] \mapsto \sigma\, g \mid g \in \mathcal{G}, w \in W\, g\} \textbf{ in} \\
&\quad (\rho, (W, P', \sigma))
\end{aligned}
$$

When writing to a global g, on top of recording the written value in σ, $W\, g$ and $P\, g$ are set to the set $\{S\}$ for the current lockset S. When reading from a global g, now only values stored at those unknowns $\eta\,[g, a, S', w]$ where the following conditions are met are taken into account:

- $a \in S$, i.e., a is one of the currently held locks;
- $S \cap S' = \emptyset$; i.e., the intersection of the current lockset S with the background lockset at the corresponding operation $\mathsf{unlock}(a)$ after the write producing the value stored at this unknown is empty;

- $w \cap S'' = \emptyset$ for some $S'' \in P\,g$, i.e., the background lockset at the write producing the value stored at this unknown is disjoint with one of the locksets in $P\,g$. This excludes writes where the ego thread has since its last thread-local write always held at least one of the locks in w. In this case, that write can not have happened between the last thread-local write of the reading ego thread and its read;
- $a \notin S'''$ for some $S''' \in P\,g$, i.e., a has not been continuously held by the thread since its last write to g.

Accordingly, we define

$$\begin{aligned}
[\![[u,S], g = x]\!]^\sharp \eta &= \textbf{let } (W,P,\sigma) = \eta\,[u,S] \textbf{ in} \\
&\quad \textbf{let } W' = W \oplus \{g \mapsto \{S\}\} \textbf{ in} \\
&\quad \textbf{let } P' = P \oplus \{g \mapsto \{S\}\} \textbf{ in} \\
&\quad (\emptyset, (W', P', \sigma \oplus \{g \mapsto (\sigma\,x)\})) \\
[\![[u,S], x = g]\!]^\sharp \eta &= \textbf{let } (W,P,\sigma) = \eta\,[u,S] \textbf{ in} \\
&\quad \textbf{let } d = \sigma\,g \sqcup \bigsqcup \{\eta[g,a,S',w] \mid a \in S, S \cap S' = \emptyset, \\
&\qquad \exists S'' \in P\,g : S'' \cap w = \emptyset, \\
&\qquad \exists S''' \in P\,g : a \notin S'''\} \textbf{ in} \\
&\quad (\emptyset, (W, P, \sigma \oplus \{x \mapsto d\}))
\end{aligned}$$

The handling of returns, joins, signals, and waits once more remains essentially unchanged compared to the analysis from Section 4.1.1.

$$\begin{aligned}
[\![[u,S], \mathsf{return}]\!]^\sharp \eta &= \textbf{let } (W,P,\sigma) = \eta\,[u,S] \textbf{ in} \\
&\quad \textbf{let } I = \sigma\,\mathsf{self} \textbf{ in} \\
&\quad (\{[i] \mapsto \sigma\,\mathsf{ret} \mid i \in I\}, (W,P,\sigma)) \\
[\![[u,S], x = \mathsf{join}(x')]\!]^\sharp \eta &= \textbf{let } (W,P,\sigma) = \eta\,[u,S] \textbf{ in} \\
&\quad \textbf{let } v = \bigsqcup_{i' \in (\sigma\,x')} (\eta[i']) \textbf{ in} \\
&\quad \textbf{if } v = \bot \textbf{ then} \\
&\qquad (\emptyset, \bot) \\
&\quad \textbf{else} \\
&\qquad (\emptyset, (W, P, \sigma \oplus \{x \mapsto v\})) \\
[\![[u,S], \mathsf{signal}(s)]\!]^\sharp \eta &= \textbf{let } (W,P,\sigma) = \eta\,[u,S] \textbf{ in} \\
&\quad (\{[s] \mapsto (W,P,\sigma)\}, (W,P,\sigma)) \\
[\![[u,S], \mathsf{wait}(s)]\!]^\sharp \eta &= \textbf{let } (W,P,\sigma) = \eta\,[u,S] \textbf{ in} \\
&\quad \textbf{if } \eta\,[s] = \bot \textbf{ then} \\
&\qquad (\emptyset, \bot) \\
&\quad \textbf{else} \\
&\qquad (\emptyset, (W,P,\sigma))
\end{aligned}$$

Example 20. *Assume integer sets are used for abstracting int values. Consider the following concurrent program with global variable* `g` *and local variables* `x`*,* `y`*, and* `z`*:*

```
main:
  initMT;
  y = create(t1);
  z = create(t2);
  lock(c);
  lock(m_g); g = 31; unlock(m_g);
  lock(a);
  lock(b);
  lock(m_g); x = g; unlock(m_g);
  ...
```

```
t1:
  lock(a);
  lock(b);
  lock(m_g); g = 42; unlock(m_g);
  unlock(a);
  lock(m_g); g = 17; unlock(m_g);
  unlock(b);

t2:
  lock(c);
  lock(m_g); g = 59; unlock(m_g);
  unlock(c);
```

At the read $x = g$, the current lockset is $\{a, b, c, m_g\}$ and in the local state $P\,g = \{\{c\}\}$. The only unknown that receives a side-effect for this program and where all conditions above are fulfilled is the unknown $[g, b, \emptyset, \{b, m_g\}]$ which has value $\{17\}$. Hence, this is the only value read from the unknowns for g and together with the value $\{31\}$ from $\sigma\,g$ the final value for x is $\{17, 31\}$. This is more precise than either of the analyses presented thus far: Protection-Based Reading *cannot exclude any values of x as $g = \{m_g\}$, and thus has $\{0, 17, 31, 42, 59\}$ for x.* Lock-Centered Reading *has $V\,c = \{g\}$ at the read. This excludes the write by $t2$ and thus results in $\{17, 31, 42\}$ for x.* □

Theorem 9. Write-Centered Reading *is sound w.r.t. the local trace semantics.*

Proof. In Section 4.1.7, we show this analysis is an instance of the refined analysis presented there. The soundness proof for that analysis is deferred to Section 6.1.1. □

4.1.7 Write-Centered Reading with Ego-Lane Digests

We now turn to enhancing this analysis with digests. When applying a refinement based on ego-lane digests to the analysis from Section 4.1.6, the resulting analysis has the following set of unknowns:

- $[u, S, A]$ for $u \in \mathcal{N}$, $S \subseteq \mathcal{M}$, and $A \in \mathcal{A}$,
- $[g, a, S, w, A]$ for $g \in \mathcal{G}, a \in \mathcal{M}, S \subseteq \mathcal{M}, w \subseteq \mathcal{M}$, and $A \in \mathcal{A}$,
- $[i, A]$ for $i \in S_{\mathcal{V}^\sharp_{\mathbf{tid}}}$ and $A \in \mathcal{A}$, and
- $[s, A]$ for $s \in \mathcal{S}$ and $A \in \mathcal{A}$.

The constraint system once more takes the same form as the constraint system for the protection-based analysis enhanced with digests, as given in Eq. (4.1).

The right-hand sides are then given by:

$$\begin{aligned}
\mathsf{init}(A)^\sharp\ _ \quad = \quad & \textbf{let } W = \{g \mapsto \emptyset \mid g \in \mathcal{G}\} \textbf{ in} \\
& \textbf{let } P = \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}\} \textbf{ in} \\
& \textbf{let } \sigma = \{x \mapsto \top \mid x \in \mathcal{X}\} \cup \{g \mapsto [\![0]\!]^\sharp_{\mathcal{E}xp}\ \top \mid g \in \mathcal{G}\} \textbf{ in} \\
& (\emptyset, (W, P, \sigma \oplus \{\mathsf{self} \mapsto [\![i_0]\!]^\sharp_{\mathcal{E}xp}\ \top\}))
\end{aligned}$$

$$\begin{aligned}
& [\![[u, S, A_0], x = \mathsf{create}(u_1)]\!]^\sharp \eta = \\
& \quad \textbf{let } (W, P, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& \quad \textbf{let } W' = \{g \mapsto \emptyset \mid g \in \mathcal{G}\} \textbf{ in} \\
& \quad \textbf{let } P' = \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}\} \textbf{ in} \\
& \quad \textbf{let } i = \nu^\sharp\ u\ \sigma\ u_1 \textbf{ in} \\
& \quad \textbf{let } \sigma' = \sigma \oplus \left(\{\mathsf{self} \mapsto i\}) \cup \left\{g \mapsto \left(\sigma\, g \sqcap [\![0]\!]^\sharp_{\mathcal{E}xp}\ \top\right) \mid g \in \mathcal{G}\right\}\right) \textbf{ in} \\
& \quad \textbf{let } \rho = \{[u_1, \emptyset, A'] \mapsto (W', P', \sigma') \mid A' \in \mathsf{new}^\sharp_{\mathcal{A}}\ u\ u_1\ A_0\} \textbf{ in} \\
& \quad (\rho, (W, P, \sigma \oplus \{x \mapsto i\}))
\end{aligned}$$

The right-hand sides for initMT, lock, assignment to a global variable, and guards and computations on locals (once more) remain unchanged when compared to their definitions from Section 4.1.6 — except that they now consult the unknown enhanced with the digest.
For unlocking of mutexes, the new right-hand side is given by:

$$\begin{aligned}
& [\![[u, S, A_0], \mathsf{unlock}(a), A']\!]^\sharp \eta = \\
& \quad \textbf{let } (W, P, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& \quad \textbf{let } P' = \{g \mapsto P\, g \sqcup \{S \setminus \{a\}\} \mid g \in \mathcal{G}\} \textbf{ in} \\
& \quad \textbf{let } \rho = \{[g, a, S \setminus \{a\}, w, A'] \mapsto \sigma\, g \mid g \in \mathcal{G}, w \in W\, g\} \textbf{ in} \\
& \quad (\rho, (W, P', \sigma))
\end{aligned}$$

For reading from a global, we have:

$$\begin{aligned}
[\![[u, S, A_0], x = g]\!]^\sharp \eta \quad = \quad & \textbf{let } (W, P, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& \textbf{let } d = \sigma\, g \sqcup \bigsqcup \{\eta[g, a, S', w, A'] \mid a \in S, S \cap S' = \emptyset, \\
& \qquad \exists S'' \in P\, g : S'' \cap w = \emptyset, \\
& \qquad \exists S''' \in P\, g : a \notin S''', \\
& \qquad A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\ A_0\ A'\} \textbf{ in} \\
& (\emptyset, (W, P, \sigma \oplus \{x \mapsto d\}))
\end{aligned}$$

The handling of returns, joins, signals, and waits once more remains essentially unchanged compared to the analysis from Section 4.1.2.

$$\begin{aligned}
[\![[u, S, A_0], \mathsf{return}, A']\!]^\sharp \eta \quad = \quad & \textbf{let } (W, P, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& \textbf{let } I = \sigma\, \mathsf{self} \textbf{ in} \\
& (\{[i, A'] \mapsto \sigma\, \mathsf{ret} \mid i \in I\}, (W, P, \sigma))
\end{aligned}$$

$$
\begin{array}{lcl}
[\![[u,S,A_0],x=\mathsf{join}(x')]\!]^\sharp\eta & = & \textbf{let } (W,P,\sigma)=\eta\,[u,S,A_0] \textbf{ in} \\
& & \textbf{let } v=\bigsqcup_{i'\in(\sigma\,x')}\left(\bigsqcup_{A'\in\mathcal{A},\,\mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A'}(\eta[i',A'])\right) \textbf{ in} \\
& & \textbf{if } v=\bot \textbf{ then} \\
& & \quad (\emptyset,\bot) \\
& & \textbf{else} \\
& & \quad (\emptyset,(W,P,\sigma\oplus\{x\mapsto v\})) \\
[\![[u,S,A_0],\mathsf{signal}(s),A']\!]^\sharp\eta & = & \textbf{let } (W,P,\sigma)=\eta\,[u,S,A_0] \textbf{ in} \\
& & (\{[s,A']\mapsto(W,P,\sigma)\},(W,P,\sigma)) \\
[\![[u,S,A_0],\mathsf{wait}(s)]\!]^\sharp\eta & = & \textbf{let } (W,P,\sigma)=\eta\,[u,S,A_0] \textbf{ in} \\
& & \textbf{if } \left(\left(\bigsqcup_{A'\in\mathcal{A},\,\mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A'}\eta\,[s,A']\right)=\bot\right) \textbf{ then} \\
& & \quad (\emptyset,\bot) \\
& & \textbf{else} \\
& & \quad (\emptyset,(W,P,\sigma))
\end{array}
$$

Thus, again side-effects are re-directed to the unknowns equipped with digests, and the function $\mathsf{compat}^\sharp_{\mathcal{A}}$ is used to decide when information should be incorporated.

Example 21. *Consider the analysis refined with the ego-lane-derived thread* id *digest from Section 2.8, with* $\mathsf{compat}^\sharp_{\mathcal{A}}(i,C)(i,C') = \mathsf{may\_run}(i,C)(i,C')$ *as proposed in Example 10. Consider the following program fragment which corresponds to modifying Example 20 to have* t_2 *start later and have it write* g *while only holding* m_g*. Once more, assume that we use value sets for abstracting int values.*

```
main:                          t1:
  initMT;                        lock(a);
  y = create(t1);                lock(b);
  lock(c);                       lock(m_g); g = 42; unlock(m_g);
  lock(m_g); g = 31; unlock(m_g); unlock(a);
  lock(a);                       lock(m_g); g = 17; unlock(m_g);
  lock(b);                       unlock(b);
  lock(m_g); x = g; unlock(m_g);
  z = create(t2);              t2:
                                 lock(m_g); g = 59; unlock(m_g);
```

The main thread here receives the thread id $\bar{\mathsf{main}} = (\mathsf{main},\emptyset)$*, and the newly created threads receive the thread* ids $\bar{t}_1 = (\mathsf{main}\cdot\langle u_1,t_1\rangle,\emptyset)$ *and* $\bar{t}_2 = (\mathsf{main}\cdot\langle u_7,t_2\rangle,\emptyset)$*, respectively. Then, the possible values of* x *determined by the refined* Write-Centered analysis *are* $\{17,31\}$*, as the thread with thread* id $\bar{t}_2$ *is definitely not started yet when the read occurs. Without the refinement, the set would be* $\{17,31,59\}$*.* Protection-Based Reading *would find* $\{0,31,42,17,59\}$ *without refinement, and is able to exclude* $\{59\}$ *with refinement.* Lock-Centered Reading *would find* $\{17,31,42,59\}$ *without the refinement, and once more is able to exclude* $\{59\}$ *with refinement. This example thus highlights that, even after being refined with thread* id *digests from Section 2.8, the analyses still achieve different results on at least some programs.* □

Remark 16. *By choosing $\mathcal{A} = \{\bullet\}$ and setting all right-hand sides for digests to also return $\{\bullet\}$ (which is ego-lane-derived), we obtain an analysis that is equivalent to the one from Section 4.1.6.*

Theorem 10. Write-Centered Reading *enhanced with an ego-lane digest is sound w.r.t. the trace semantics.*

Proof. The entirety of Section 6.1.1 is dedicated to a detailed account of the proof, we quickly sketch its flow here: It proceeds by first massaging the concrete semantics into an equivalent form that has unknowns that can more readily be related to the abstract semantics and then proving that the least solution of the original constraint system can be constructed from the modified one and vice versa (see Proposition 16). The central issue is to prove that when reading a global g, the restriction to the values of unknowns $[g, a, S', w]$ as indicated by the right-hand-side function is sound (see Proposition 17). Next, an analysis with adapted side-effects is introduced, and a relationship between solutions of this analysis and the original analysis established (Proposition 18). Finally, to put it all together, solutions of the modified abstract constraint system are shown to give rise to solutions of the modified concrete constraint system (Theorem 17). □

4.1.8 Combining Write-Centered with Lock-Centered Reading

The analyses described in Sections 4.1.4 and 4.1.6 are sound, yet incomparable. This is evidenced by Example 20, in which *Write-Centered* is more precise than *Lock-Centered Reading*, and the following example, where the opposite is the case.

Example 22. *Assume that we use value sets for abstracting the values of integer variables. Consider the following concurrent program with two threads, a global variable* `g` *and local variables* `x` *and* `y`*:*

```
main:                                t1:
  initMT;                              lock(d);
  y = create(t1);                      lock(a);
  lock(d);                             lock(m_g); g = 42; unlock(m_g);
  lock(a);                             unlock(a);
  unlock(d);                           lock(m_g); g = 17; unlock(m_g);
  lock(m_g); x = g; unlock(m_g);       unlock(d);
  ...
```

For Write-Centered Reading, *(on top of the initial value of $\{0\}$) both the value at unknowns $[g, a, \{d\}, \{d, a, m_g\}]$ with value $\{42\}$ and $[g, d, \emptyset, \{d, m_g\}]$ with value $\{17\}$ are read, resulting in a value of $\{0, 17, 42\}$ for x. For* Lock-Centered Reading, *at the read in* `main`*, $L\,a = \{\{d\}\}$, and hence $[g, a, \{d\}]$ with value $\{42\}$ does not fulfill the conditions under which its value is taken into account, resulting in a value of $\{0, 17\}$ for x.* □

To obtain an analysis that is sound and more precise than *Write-Centered* and *Lock-Centered Reading*, both can be combined. For the combination, we do not rely on a

reduced product construction but, instead exploit the information of all simultaneously tracked data-structures V, W, P, L together to improve the set of writes read at a particular read operation.

For completeness, we list all right-hand sides of the combined analysis here, giving the version refined according to some ego-lane digest right away.

$$\begin{aligned}
\mathsf{init}(A)^\sharp\ _ \quad = \quad & \textbf{let } W = \{g \mapsto \emptyset \mid g \in \mathcal{G}\} \textbf{ in} \\
& \textbf{let } P = \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}\} \textbf{ in} \\
& \textbf{let } V = \{a \mapsto \emptyset \mid a \in \mathcal{M}\} \textbf{ in} \\
& \textbf{let } L = \{a \mapsto \emptyset \mid a \in \mathcal{M}\} \textbf{ in} \\
& \textbf{let } \sigma = \{x \mapsto \top \mid x \in \mathcal{X}\} \cup \{g \mapsto [\![0]\!]^\sharp_{\mathcal{E}xp}\ \top \mid g \in \mathcal{G}\} \textbf{ in} \\
& (\emptyset, (W, P, V, L, \sigma \oplus \{\mathsf{self} \mapsto [\![i_0]\!]^\sharp_{\mathcal{E}xp}\ \top\}))
\end{aligned}$$

The right-hand side for initMT once more returns the abstract state of the predecessor and causes no side-effects.

$$\begin{aligned}
& [\![[u, S, A_0], x = \mathsf{create}(u_1)]\!]^\sharp \eta = \\
& \quad \textbf{let } (W, P, V, L, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& \quad \textbf{let } W' = \{g \mapsto \emptyset \mid g \in \mathcal{G}\} \textbf{ in} \\
& \quad \textbf{let } P' = \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}\} \textbf{ in} \\
& \quad \textbf{let } V' = \{a \mapsto \emptyset \mid a \in \mathcal{M}\} \textbf{ in} \\
& \quad \textbf{let } L' = \{a \mapsto \emptyset \mid a \in \mathcal{M}\} \textbf{ in} \\
& \quad \textbf{let } i = v^\sharp\ u\ \sigma\ u_1 \textbf{ in} \\
& \quad \textbf{let } \sigma' = \sigma \oplus \left(\{\mathsf{self} \mapsto i\}\right) \cup \left\{g \mapsto \left(\sigma\, g \sqcap [\![0]\!]^\sharp_{\mathcal{E}xp}\ \top\right) \mid g \in \mathcal{G}\right\}\Big) \textbf{ in} \\
& \quad \textbf{let } \rho = \{[u_1, \emptyset, A'] \mapsto (W', P', V', L', \sigma') \mid A' \in \mathsf{new}^\sharp_{\mathcal{A}}\ u\ u_1\ A_0\} \textbf{ in} \\
& \quad (\rho, (W, P, V, L, \sigma \oplus \{x \mapsto i\}))
\end{aligned}$$

The right-hand sides for guards and computations on locals are once more defined to operate on σ only and return W, P, V, and L unchanged. The right-hand side for locking a mutex a now is given by:

$$\begin{aligned}
[\![[u, S, A_0], \mathsf{lock}(a)]\!]^\sharp \eta \quad = \quad & \textbf{let } (W, P, V, L, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& \textbf{let } V' = V \oplus \{a \mapsto \emptyset\} \textbf{ in} \\
& \textbf{let } L' = L \oplus \{a \mapsto \{S\}\} \textbf{ in} \\
& (\emptyset, (W, P, V', L', \sigma))
\end{aligned}$$

while the right-hand side for unlocking a mutex a is given by:

$$\begin{aligned}
[\![[u, S, A_0], \mathsf{unlock}(a), A']\!]^\sharp \eta \quad = \quad & \textbf{let } (W, P, V, L, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& \textbf{let } P' = \{g \mapsto P\, g \sqcup \{S \setminus \{a\}\} \mid g \in \mathcal{G}\} \textbf{ in} \\
& \textbf{let } \rho = \{[g, a, S \setminus \{a\}, w, A'] \mapsto \sigma\, g \mid \\
& \qquad\qquad g \in \mathcal{G}, w \in W\, g\} \textbf{ in} \\
& (\rho, (W, P', V, L, \sigma))
\end{aligned}$$

The right-hand side for writing to a global variable is given by:

$$\begin{aligned}
[\![[u, S, A_0], g = x]\!]^\sharp \eta \quad = \quad & \textbf{let } (W, P, V, L, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& \textbf{let } W' = W \oplus \{g \mapsto \{S\}\} \textbf{ in} \\
& \textbf{let } P' = P \oplus \{g \mapsto \{S\}\} \textbf{ in} \\
& \textbf{let } V' = \{a \mapsto (V\,a \cup \{g\}) \mid a \in \mathcal{M}\} \textbf{ in} \\
& (\emptyset, (W', P', V, L, \sigma \oplus \{g \mapsto (\sigma\,x)\}))
\end{aligned}$$

The handling of returns, joins, signals, and waits once more remains essentially unchanged compared to the analyses from Sections 4.1.2 and 4.1.5.

$$\begin{aligned}
[\![[u, S, A_0], \mathsf{return}, A']\!]^\sharp \eta \quad = \quad & \textbf{let } (W, P, V, L, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& \textbf{let } I = \sigma\,\mathsf{self} \textbf{ in} \\
& (\{[i, A'] \mapsto \sigma\,\mathsf{ret} \mid i \in I\}, (V, L, \sigma)) \\
[\![[u, S, A_0], \mathsf{signal}(s), A']\!]^\sharp \eta \quad = \quad & \textbf{let } (W, P, V, L, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& (\{[s, A'] \mapsto (W, P, V, L, \sigma)\}, (W, P, V, L, \sigma)) \\
[\![[u, S, A_0], x = \mathsf{join}(x')]\!]^\sharp \eta \quad = \quad & \textbf{let } (W, P, V, L, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& \textbf{let } v = \bigsqcup_{i' \in (\sigma\,x')} \left(\bigsqcup_{A' \in \mathcal{A},\, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'} (\eta[i', A']) \right) \textbf{ in} \\
& \textbf{if } v = \bot \textbf{ then} \\
& \qquad (\emptyset, \underline{\bot}) \\
& \textbf{else} \\
& \qquad (\emptyset, (W, P, V, L, \sigma \oplus \{x \mapsto v\})) \\
[\![[u, S, A_0], \mathsf{wait}(s)]\!]^\sharp \eta \quad = \quad & \textbf{let } (W, P, V, L, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& \textbf{if } \left(\bigsqcup_{A' \in \mathcal{A},\, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'} \eta\,[s, A'] \right) = \underline{\bot} \textbf{ then} \\
& \qquad (\emptyset, \underline{\bot}) \\
& \textbf{else} \\
& \qquad (\emptyset, (W, P, V, L, \sigma))
\end{aligned}$$

The key point then is for reading from a global g. Here, the information in the data-structure P is not only used to restrict the set of reads for *Write-Centered Reading*, but can also be used for restricting the set for *Lock-Centered Reading* (highlighted in teal below). This way, also in the Lock-Centered style such writes can be excluded where the ego thread has since its last local write always maintained a non-empty intersection with the set of mutexes held by the thread that wrote the value, which means that write cannot be the last one. This is what differentiates this analysis from simply applying a reduced

product construction of both styles.

$$
\begin{aligned}
[\![[u, S, A_0], x = g]\!]^\sharp \eta \quad = \quad & \textbf{let } (W, P, V, L, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& \textbf{let } d_{\text{WC}} = \bigsqcup\{\eta[g, a, S', w, A'] \mid a \in S, S \cap S' = \emptyset, \\
& \qquad \exists S'' \in P\,g : S'' \cap w = \emptyset, \\
& \qquad \exists S''' \in P\,g : a \notin S''', \\
& \qquad A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \textbf{ in} \\
& \textbf{let } d_{\text{LC}} = \bigsqcup\{\eta[g, a, S', w, A'] \mid a \in \mathcal{M}, \\
& \qquad g \notin V\,a, \exists B \in L\,a, B \cap S' = \emptyset, \\
& \qquad \exists S'' \in P\,g : S'' \cap w = \emptyset, \\
& \qquad A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \textbf{ in} \\
& (\emptyset, (V, L, \sigma \oplus \{x \mapsto \sigma\,g \sqcup (d_{\text{WC}} \sqcap d_{\text{LC}})\}))
\end{aligned}
$$

The resulting analysis is thus more precise than *Write-Centered Reading* and more precise than *Lock-Centered Reading*. We do not detail the soundness proof here. Intuitively, it follows from the soundness proofs of the two analyses in Sections 6.1.1 and 6.1.2 and the additional observation that condition (W3) from Proposition 17 in Section 6.1.1 also applies to Lock-Centered Reading.

4.1.9 Remark on Refinement with Thread *Id*s

All the novel analyses presented in this chapter can be refined with ego-lane digests, and we have usually exemplified this with the digests for computing thread *id*s (from Section 2.8). This is a natural choice and enhances the precision of the analyses by limiting reading

I1 from other threads that have not yet been created.

However, this covers only part of the refinements one would like to make according to the computed thread *id*s. The other two properties one would like to consider are

I2 not reading older writes by the ego thread if it is known to be unique;

I3 and also not reading the writes of threads that have definitely been joined — provided the value they wrote last has been overwritten.

We will cover both properties in the subsequent section, but quickly remark here that this refinement cannot be done purely based on reasoning on ego-lane digests. The intuitive reason is that the ego-lane digests are an abstraction of trace compatibility, and it is often possible for a thread to re-acquire a mutex that no other thread has held since it was released last. For *ego-lane* digests, this compatibility is preserved in the forward direction, making excluding earlier writes by the same thread infeasible for many programs.

As all analyses presented here keep an abstraction of the most-recent *thread-local* writes to a global g in their $\sigma\,g$ component, a dedicated construction for the analyses

refined with the digests from Section 2.8 is possible. To this end, one can modify the analyses to forgo reading from unknowns associated with the thread *id* of the unique ego thread. We do not provide a soundness argument here beyond this intuition. To also achieve property **I3**, the σ components of the analyses would have to be modified to merge the thread-local writes by threads which are joined into σ. We also only give this informal intuition here.

Both improvements will be fully realized for the analysis in the next section (see Section 4.2.4 for details), where their justification relies — in part — on the (full-fledged) digest framework.

4.2 Analyses Considering Clusters of Globals

What all the analyses presented in the preceding sections have in common is that they abstract the value of each global on its own, i.e., are inherently non-relational when it comes to the values of globals. Nevertheless, one may instantiate these analyses with a relational domain to track relations between locals and those copies of globals tracked in the local state. In this section, we turn to analyses that jointly abstract (clusters of) globals, and can thus be relational for globals when instantiated with an appropriate domain. Inferring relational properties is particularly challenging for multi-threaded programs as all interferences by other threads that may happen in parallel must be taken into account. In such an environment, only relational properties between globals protected by common mutexes are likely to persist throughout program execution. Generally, relations on clusters consisting of fewer variables are less brittle than those on larger clusters. Moreover, monolithic relational analyses employing, e.g., the polyhedral abstract domain, are known to be notoriously expensive [81, 121]. Tracking *smaller* clusters may even be more precise than tracking larger clusters [45].

Example 23. *Consider the following program. All accesses to globals g, h, and i are protected by the mutex a. The operations on mutexes m_g $g \in \mathcal{G}$ are omitted for brevity here.*

```
main:                                   t1:          t2:
 initMT;                                 lock(a);     lock(a);
 x = create(t1); y = create(t2);         x = h;       g = ?; h = ?;
 lock(a);                                i = x;       unlock(a);
 g = ?; h = ?; i = ?;                    unlock(a);   return;
 unlock(a); r = join(y); lock(a);        return;
 z = ?; g = z; h = z; i = z;
 unlock(a); lock(a);
 // ASSERT(h==i); (1)
 // ASSERT(g==h); (2)
 unlock(a);
```

In this program, the main thread creates two new threads, starting at t_1 and t_2, respectively. Then, it locks the mutex a to set all globals non-deterministically to some value and unlocks

a again. After having joined the thread t_2, it locks a again and sets all globals to the same *unknown value and unlocks a again. Thread t_1 sets i to the value of h. Thread t_2 sets g and h to (potentially different) unknown values. Assume we are interested in equalities between globals. In order to succeed in showing assertion* (1)*, it is necessary to detect that the main thread is unique and thus cannot read its past writes since these have been overwritten. Additionally, the analysis needs to certify that thread t_2 also is unique, has been joined before the assertion, and that its writes must also have been overwritten.*

For an analysis to prove assertion (2)*, propagating a joint abstraction of the values of all globals protected by a does not suffice: At the unlock of a in t_1, $g{=}h$ need not hold. If this monolithic relation is propagated to the last lock of a in main,* (2) *cannot be shown — despite t_1 modifying neither g nor h.* □

Here, we show that the loss of precision indicated in the example can be remedied by replacing the monolithic abstraction of all globals protected by a mutex with suitably chosen subclusters. In the example, we propose to separately consider the subclusters $\{g, h\}$ and $\{h, i\}$. As t_1 does not write any values to the cluster $\{g, h\}$, the imprecise relation $\top$ is not propagated to the main thread and assertion (2) can be shown.

Subsequently, we give relational analyses of the values of globals which are generic in the relational domain $\mathcal{R}$, with 2-decomposable domains being particularly well-suited, as the concept of *clusters* is central to the analyses. We focus on relations between globals that are jointly *write*-protected by some mutex.

4.2.1 Mutex Meet

As in Section 4.1.1, we once more assume we are given for each global g, a set $\bar{\mathcal{M}}[g]$ of (write) *protecting* mutexes, i.e., mutexes that are *always* held when g is written. Let $\bar{\mathcal{G}}[a] = \{g \in \mathcal{G} \mid a \in \bar{\mathcal{M}}[g]\}$ denote the set of globals protected by a mutex a. Let $\emptyset \neq \mathcal{Q}_a \subseteq 2^{\bar{\mathcal{G}}[a]}$ the set of clusters of these globals we associate with a. For technical reasons, we require at least one cluster per mutex a. This cluster may however be chosen as the empty cluster $\emptyset$, thus not associating any information with a.

Our basic idea is to store at unknowns $[a, Q]$ (for each mutex a and cluster $Q \in \mathcal{Q}_a$) an abstraction of the relations *only* between globals in Q. By construction, all globals in Q are protected by a. Whenever the mutex a is locked, the relational information stored at all $[a, Q]$ is incorporated into the local state by the lattice operation *meet*, i.e., the local state now maintains relations between locals *as well as* globals which no other thread can modify at this program point. Whenever a is unlocked, the new relation between globals in all corresponding clusters $Q \in \mathcal{Q}_a$ is side-effected to the respective unknowns $[a, Q]$. Simultaneously, all information on globals no longer protected is *forgotten* to obtain the new local state. In this way, the analysis is fully relational in the local state while only keeping relations within clusters of globals jointly protected by some mutex. All non-trivial right-hand sides are detailed in Fig. 4.2 — we briefly sketch those here and give an informal definition of the trivial right-hand sides. For the **start point** of the program and the empty lockset, the right-hand side $\mathsf{init}^\sharp$ returns the $\top$ relation

$$\mathsf{init}^\sharp\,\eta = (\emptyset, [\![\mathsf{self} \leftarrow^\sharp ([\![i_0]\!]^\sharp_{\mathcal{E}xp}\,\top)]\!]^\sharp_{\mathcal{R}}\,\top)$$

$$[\![[u,S], \mathsf{initMT}]\!]^\sharp\eta = \textbf{let } r(Q) = [\![\{g \leftarrow 0 \mid g \in Q\}]\!]^\sharp_{\mathcal{R}}\top \textbf{ in } \textbf{let } \rho = \{[a,Q] \mapsto r(Q) \mid a \in \mathcal{M}, Q \in \mathcal{Q}_a\} \textbf{ in } (\rho, \eta\,[u,S])$$

$$[\![[u,S], x{=}\mathsf{create}(u_1)]\!]^\sharp\eta = \textbf{let } r = \eta\,[u,S] \textbf{ in } \textbf{let } i = v^\sharp\,u\,(\mathsf{unlift}\,r)\,u_1 \textbf{ in } \textbf{let } r' = \left\{[\![\mathsf{self} \leftarrow^\sharp i]\!]^\sharp_{\mathcal{R}}\,r\right\}\Big|_{\mathcal{X}} \textbf{ in } \textbf{let } \rho = \{[u_1,\emptyset] \mapsto r'\} \textbf{ in } (\rho, [\![x \leftarrow^\sharp i]\!]^\sharp_{\mathcal{R}} r)$$

$$[\![[u,S], g = x]\!]^\sharp\eta = (\emptyset, [\![g \leftarrow x]\!]^\sharp_{\mathcal{R}}\,(\eta\,[u,S]))$$

$$[\![[u,S], x = g]\!]^\sharp\eta = (\emptyset, [\![x \leftarrow g]\!]^\sharp_{\mathcal{R}}\,(\eta\,[u,S]))$$

$$[\![[u,S], \mathsf{lock}(a)]\!]^\sharp\eta = \left(\emptyset, \eta\,[u,S] \sqcap \left(\textstyle\bigsqcap_{Q\in\mathcal{Q}_a} \eta\,[a,Q]\right)\right)$$

$$[\![[u,S], \mathsf{unlock}(a)]\!]^\sharp\eta = \textbf{let } r = \eta\,[u,S] \textbf{ in } \textbf{let } \rho = \{[a,Q] \mapsto r|_Q \mid Q \in \mathcal{Q}_a\} \textbf{ in } \left(\rho, r|_{\mathcal{X}\cup\bigcup\{\vec{\mathcal{G}}[a'] \mid a' \in (S\setminus a)\}}\right)$$

$$[\![[u,S], \mathsf{return}]\!]^\sharp\eta = \textbf{let } r = \eta\,[u,S] \textbf{ in } \textbf{let } I^\sharp = (\mathsf{unlift}\,r)\,\mathsf{self} \textbf{ in } \left(\left\{[i^\sharp] \mapsto r|_{\{\mathsf{ret}\}} \mid i^\sharp \in I^\sharp\right\}, r\right)$$

$$[\![[u,S], x'{=}\mathsf{join}(x)]\!]^\sharp\eta = \textbf{let } v = \textstyle\bigsqcup_{i'\in((\mathsf{unlift}\,r)\,x')} \mathsf{unlift}\,(\eta[i'])\,\mathsf{ret} \textbf{ in } \textbf{if } v = \bot \textbf{ then } (\emptyset, \underline{\bot}) \textbf{ else } \left(\emptyset, [\![x' \leftarrow^\sharp v]\!]^\sharp_{\mathcal{R}}(\eta\,[u,S])\right)$$

$$[\![[u,S], \mathsf{signal}(s)]\!]^\sharp\eta = (\{[s] \mapsto \eta\,[u,S]\}, \eta\,[u,S])$$

$$[\![[u,S], \mathsf{wait}(s)]\!]^\sharp\eta = \textbf{if } \eta\,[s] = \underline{\bot} \textbf{ then } (\emptyset, \underline{\bot}) \textbf{ else } (\emptyset, \eta\,[u,S])$$

Figure 4.2: Right-hand sides for the basic analysis. Guards and assignments on local variables are defined in the straightforward way and are omitted here. All functions are strict in $\underline{\bot}$, which describes the empty set of local traces. (As the abstract values tracked by this analysis consist of only one component, $\underline{\bot}$ and $\bot$ coincide here.) We only display definitions for non-$\underline{\bot}$ abstract values in this figure. $[\![\{g \leftarrow 0 \mid g \in Q\}]\!]^\sharp_{\mathcal{R}}$ is shorthand for the abstract transformer corresponding to the assignment of 0 to all variables in Q one-by-one.

updated such that the variable self holds the abstract thread *id* i_0 of the *main* thread. The right-hand side for the action initMT returns the abstract state of the predecessor and produces a side-effect for each mutex a and cluster Q that initializes all globals from the cluster with the value 0.

For a **thread creating** edge starting in program point u with lockset S, the right-hand side $[\![[u, S], x{=}\mathsf{create}(u_1)]\!]^\sharp$ first generates a new abstract thread *id* which then once more is assigned to the variable x in the local state of the current thread. Additionally, the start state r' for the newly created thread is constructed and side-effected to the thread's start point with empty lockset $[u_1, \emptyset]$. Since threads start with empty lockset, the state r' is obtained by removing all information about globals from the local state of the creator and assigning the new abstract thread *id* to the variable self.

When **locking** a mutex a, the states stored at unknowns $[a, Q]$ with $Q \in \mathcal{Q}_a$ are combined with the local state by *meet*. This is sound because the value stored at any $[a, Q]$ only maintains relationships between variables write-protected by a, and these values soundly account for the program state at every $\mathsf{unlock}(a)$ and at program start. When **unlocking** a, on the other hand, the local state restricted to the appropriate clusters $Q \in \mathcal{Q}_a$ is side-effected to the respective unknowns $[a, Q]$, so that the changes made to variables in the cluster become visible to other threads. Also, the local state is restricted to the local variables and only those globals for which at least one protecting mutex is still held.

As special mutexes m_g immediately surrounding accesses to g are used to ensure atomicity, and information about g is associated with them, all **reads** and **writes** refer to the local copy of g. **Guards** and **assignments** (which may only involve local variables) are defined analogously.

For a **return** edge, the local state — restricted to the variable ret whose value is to be returned — is side-effected to all members of the abstract thread *id* (recall that the thread *id* domain is given as the powerset domain over some set $S_{\mathcal{V}^\sharp_{\mathbf{tid}}}$) of the current thread, i.e., all members of $(\mathsf{unlift}\, r)\,\mathsf{self}$.

Remark 17. *The handling of* return *as given in Fig. 4.2 could be further improved to maintain some relationship between the thread* id *stored in the variable* self *and the value of* ret *by not side-effecting the same value* $r|_{\{ret\}}$ *to all unknowns but instead potentially improving the value of* r *before restricting to* ret*. In this case, the value side-effected to some unknown* $[i^\sharp]$ *would be given by* $(r \sqcap \mathit{lift}\,(\top \oplus \{\mathit{self} \mapsto \{i^\sharp\}\}))\big|_{\{ret\}}$*. This is similar in spirit to the way additional precision can be obtained for calls to* join *that is outlined in Remark 7. As this construction once more is quite complicated, and we will later use digests to track thread* ids *anyway, we do not realize this optimization.*

For **join**, the least upper bound of all *return* values of all possibly joined threads is computed. This value then is assigned to the variable on the left-hand side of the *join* statement in the local state. The handling of **signal** and **wait** is akin to how these were handled for the analyses in the previous chapter.

Example 24. *Consider the program*[3] *where* $\bar{\mathcal{M}}[g] = \{a, b, m_g\}$, $\bar{\mathcal{M}}[h] = \{a, b, m_h\}$, $\mathcal{Q}_a = \{\{g, h\}\}$, $\mathcal{Q}_b = \{\{g, h\}\}$.

```
main:                          t1:                        t2:
 x = create(t1); y = ?;         lock(b);                   lock(b);
 lock(a); lock(b);              unlock(b);                 lock(a);
 g = y; h = y+9;                lock(a);                   // ASSERT(g==h); (4)
 unlock(b); lock(b);            lock(b);                   unlock(a);
 h = y;                         // ASSERT(g==h); (3)       unlock(b);
 // ASSERT(g==y); (1)           y = ?; g = y; h = y;
 // ASSERT(h==y); (2)           unlock(b);
 unlock(b); unlock(a);          unlock(a);
 x = create(t2);
```

Our analysis succeeds in proving all assertions here. Thread t_2 *is of particular interest: When locking* b *only* $g \leq h$ *is known to hold, and locking the additional mutex* a *means that the better information* $g = h$ *becomes available. The analysis by Mukherjee et al. [89], on the other hand, only succeeds in proving assertion (2) — even when all globals are put in the same region. It cannot establish (1) because all correlations between locals and globals are forgotten when the* mix *operation is applied at the second* $\mathsf{lock}(b)$ *in the main thread. (3) cannot be established because, at* $\mathsf{lock}(b)$ *in* t_1*, the mix operation also incorporates the state after the first* $\mathsf{unlock}(b)$ *in the main thread, where* $g = h$ *does not hold. Similarly, for (4). The same applies for assertion (3) and the analysis using* lock invariants *proposed by Miné [85]. This analysis also falls short of showing (1), as at the* $\mathsf{lock}(b)$ *in the main thread, the lock invariant associated with* b *is joined into the local state. (4) is similarly out of reach. The same reasoning also applies to [85, 89] and the non-relational analyses presented in the preceding sections after equipping the analyses with thread* ids. □

Theorem 11. Mutex-Meet *is sound w.r.t. the local trace semantics.*

Proof. In Section 4.2.2, we show this analysis is an instance of the refined analysis presented there. The soundness proof for that analysis is deferred to Section 6.2.1. □

Remark 18. *Incorporating some information associated with some mutex* a *into the local state upon locking this mutex, and forgetting this information again once the mutex* a *is unlocked, is closely related to the rules for locking and unlocking in* concurrent separation logic *proposed by O'Hearn [96] (extending earlier work of Owicki and Gries [98]). There, the proof specifies a* resource invariant *associated with a which is added to the local state when the mutex* a *is locked, and needs to hold again once the mutex* a *is unlocked, at which point it is removed again.*

[3] In all examples in this section, g, h, and i are globals, whereas x, y, and z are locals, and the clusters at special mutexes m_g contain only g: $\mathcal{Q}_{m_g} = \{\{g\}\}$. Also, the lock and unlock operations for m_g, $g \in \mathcal{G}$ are omitted for brevity. Unless explicitly stated otherwise, domain $\mathcal{R}_1$ from Example 11, enhanced with variable inequalities is used.

4.2.2 Mutex Meet with Digests

To improve the precision of this analysis, we take additional abstractions of local traces in the form of digests as defined in Section 2.3 into account. The resulting analysis then has the following set of unknowns

- $[u, S, A]$ for $u \in \mathcal{N}$, $S \subseteq \mathcal{M}$ and $A \in \mathcal{A}$,
- $[a, Q, A]$ for $a \in \mathcal{M}$, $Q \in \mathcal{Q}_a$ and $A \in \mathcal{A}$,
- $[i, A]$ for $i \in \mathcal{V}^\sharp_{\mathbf{tid}}$ and $A \in \mathcal{A}$, and
- $[s, A]$ for $s \in \mathcal{S}$ and $A \in \mathcal{A}$.

with the constraint system taking the following form:

$$
\begin{array}{lll}
[u_0, \emptyset, A] & \sqsupseteq & \mathsf{init}(A)^\sharp \\
\quad \text{for } A \in \mathsf{init}^\sharp_{\mathcal{A}} & & \\
[u', S, A'] & \sqsupseteq & [\![[u, S, A_0], x = \mathsf{create}(u_1)]\!]^\sharp \\
\quad \text{for } (u, x = \mathsf{create}(u_1), u') \in \mathcal{E}, A' \in [\![u, x = \mathsf{create}(u_1)]\!]^\sharp_{\mathcal{A}}(A_0) & & \\
[u', S \cup \{a\}, A'] & \sqsupseteq & [\![[u, S, A_0], \mathsf{lock}(a), A_1]\!]^\sharp \\
\quad \text{for } (u, \mathsf{lock}(a), u') \in \mathcal{E}, A' \in [\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}(A_0, A_1) & & \\
[u', S, A'] & \sqsupseteq & [\![[u, S, A_0], \mathsf{act}, A_1]\!]^\sharp \\
\quad \text{for } (u, \mathsf{act}, u') \in \mathcal{E}, \mathsf{act} \in (\mathcal{A}ct_{\mathsf{observing}} \setminus \{\mathsf{lock}(a) \mid a \in \mathcal{M}\}), & & \\
\qquad A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0, A_1) & & \\
[u', S \setminus \{a\}, A'] & \sqsupseteq & [\![[u, S, A_0], \mathsf{unlock}(a), A']\!]^\sharp \\
\quad \text{for } (u, \mathsf{unlock}(a), u') \in \mathcal{E}, A' \in [\![u, \mathsf{unlock}(a)]\!]^\sharp_{\mathcal{A}}(A_0) & & \\
[u', S, A'] & \sqsupseteq & [\![[u, S, A_0], \mathsf{act}, A']\!]^\sharp \\
\quad \text{for } (u, \mathsf{act}, u') \in \mathcal{E}, \mathsf{act} \in (\mathcal{A}ct_{\mathsf{observable}} \setminus \{\mathsf{unlock}(a) \mid a \in \mathcal{M}\}), & & \\
\qquad A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0) & & \\
[u', S, A'] & \sqsupseteq & [\![[u, S, A_0], \mathsf{act}]\!]^\sharp \\
\quad \text{for } (u, \mathsf{act}, u') \in \mathcal{E}, \mathsf{act} \in \mathcal{A}ct_{\mathsf{local}}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0) & &
\end{array}
\tag{4.2}
$$

The new right-hand sides then consult appropriate refined unknowns and also cause side-effects to appropriate unknowns. We exemplify this for some actions here, with other right-hand sides defined analogously. All right-hand sides are detailed in the

proof in Section 6.2.1.

$$
\begin{aligned}
[\![[u,S,A_0], x = \mathsf{create}(u_1)]\!]^\sharp\eta &= \textbf{let } r = \eta\,[u,S,A_0] \textbf{ in}\\
&\quad \textbf{let } i = v^\sharp\, u\,(\mathsf{unlift}\, r)\, u_1 \textbf{ in}\\
&\quad \textbf{let } r' = \left\{ [\![\mathsf{self} \leftarrow^\sharp i]\!]^\sharp_{\mathcal{R}}\, r \right\}\Big|_{\mathcal{X}} \textbf{ in}\\
&\quad \textbf{let } \rho = \{[u_1,\varnothing,A'] \mapsto r' \mid \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0\} \textbf{ in}\\
&\quad (\rho, [\![x \leftarrow^\sharp i]\!]^\sharp_{\mathcal{R}} r)\\
[\![[u,S,A_0], \mathsf{lock}(a), A_1]\!]^\sharp\eta &= \left(\varnothing, \eta\,[u,S,A_0] \sqcap \left(\textstyle\bigsqcap_{Q\in\mathcal{Q}_a} \eta\,[a,Q,A_1]\right)\right)\\
[\![[u,S,A_0], \mathsf{unlock}(a), A']\!]^\sharp\eta &= \textbf{let } r = \eta\,[u,S,A_0] \textbf{ in}\\
&\quad \textbf{let } \rho = \{[a,Q,A'] \mapsto r|_Q \mid Q \in \mathcal{Q}_a\} \textbf{ in}\\
&\quad \left(\rho, r|_{\mathcal{X}\cup\bigcup\{\mathcal{G}[a'] \mid a'\in(S\setminus a)\}}\right)
\end{aligned}
$$

$$
\begin{aligned}
[\![[u,S,A_0], \mathsf{initMT}, A']\!]^\sharp\eta &= \textbf{let } r(Q) = [\![\{g \leftarrow 0 \mid g \in Q\}]\!]^\sharp_{\mathcal{R}} \top \textbf{ in}\\
&\quad \textbf{let } \rho = \{[a,Q,A'] \mapsto r(Q) \mid a \in \mathcal{M}, Q \in \mathcal{Q}_a\} \textbf{ in}\\
&\quad (\rho, \eta\,[u,S,A_0])
\end{aligned}
$$

Remark 19. *The new right-hand sides can alternatively be defined in terms of the right-hand sides of the analysis from the previous section, which are used as black boxes. The right-hand sides then act as wrappers, mapping unknowns consulted or side-effected to by the original analysis to the appropriate unknown of the refined system. Our earlier work [108] details this approach for some of the actions considered here. Here, we go for a direct definition in the interest of clarity.*

Example 25. *Consider the following program fragment and assume that the mutex a protects both g and h.*

```
main:                t1:                t2:
 initMT;              x = create(t2);     lock(a);
 lock(a);             lock(a);            // ASSERT(h<g);
 h = 9; g = 10;       h = 11; g = 12;     unlock(a);
 unlock(a);           unlock(a);
 x = create(t1);
```

When refining the analysis from the previous section according to the digests from Fig. 2.12, tracking which mutexes have been locked at least once in the local trace, it succeeds in proving the assert in this program as the initial values of 0 *for g and h can be excluded.* □

Remark 20. *By choosing* $\mathcal{A} = \{\bullet\}$ *and setting all right-hand sides for digests to also return* $\{\bullet\}$*, one obtains an analysis equivalent to the one from Section 4.2.1.*

Theorem 12. Mutex-Meet *enhanced with digest is sound w.r.t. the local trace semantics.*

Proof. The detailed proof is deferred to Section 6.2.1. Roughly, it proceeds as follows: First, the constraint system for the concrete semantics is massaged into an equivalent

form that can more readily be related to the abstract semantics. Then, the equivalence between both formulations is established (Proposition 28). Then, solutions of the abstract constraint system are shown to give rise to solutions of the modified concrete constraint system (Theorem 20). □

4.2.3 Exploiting Thread *ID*s to Improve Relational Analyses

Recall from Section 4.1.9 and Example 23 the ways in which one may want to exploit abstract thread *id*s and their uniqueness to improve the analysis to not read

I1 from other threads that have not yet been created.

I2 the ego thread's past writes, if its thread *id* is unique.

I3 past writes from threads that have already been joined where these values have been overwritten.

Improvements **I1** and **I3** have, e.g., been realized in a setting where thread *id*s and which thread is joined at what program point can be read off from control-flow graphs [72]. Here, however, this information is computed *during* analysis.

In our framework, **I1** is already achieved by instantiating the refined analysis from Section 4.2.2 with the thread *id* digests from Section 2.8.

Example 26. *Consider the program below where* $\bar{\mathcal{M}}[g] = \{a, b, m_g\}$, $\bar{\mathcal{M}}[h] = \{a, b, m_h\}$, $\bar{\mathcal{M}}[i] = \{m_i\}$ *and assume* $\mathcal{Q}_a = \{\{g, h\}\}$.

```
main:                              t1:
 initMT;                            lock(a);
 x = create(t1);                    r = ?; g = r; h = r;
 lock(a);                           unlock(a);
 // ASSERT(g==h);  (1)
 unlock(a);                        t2:
 y = create(t2);                    lock(a); v = g; unlock(a);
 lock(a);
 // ASSERT(g==h);  (2)             t3:
 g = 42; h = 42;                    lock(a); g = 19; unlock(a);
 unlock(a);
 z = create(t3);
 i = 3;  i = 2;
 // ASSERT(i==2); (3)
 i = 8;
```

The analysis succeeds in proving (1), as the thread (starting at) t_3 *that breaks the invariant g=h has definitely not been started yet at this program point. Without refinement, the analysis from Section 4.2.1 could not prove (1). However, this refinement does not suffice to prove (2). At this program point,* t_2 *may already be started. At the* lock(a) *in* t_2, t_3 *may also be started;*

thus, the violation of the invariant g=h by t_3 is incorporated into the local state of t_2 at lock. Upon unlock(a), despite t_2 only reading *g, the imprecise abstract relation violating g=h, is side-effected to $[a, \{g, h\}, t_2]$ and is incorporated at the second lock(a) of the main thread. The final shortcoming is that each thread reads all its own past (and future!) writes — even when it is known to be unique. This means that (3) cannot be proven.* □

To achieve **I2** some effort is required as our analysis forgets the values of globals when they become unprotected. This is in contrast to, e.g., the analyses proposed by Miné [85] or Mukherjee et al. [89] where no extra effort is required to achieve **I2**.

In the following section, we consider the analyses from Section 4.2.2 instantiated with the thread *id* digests from Section 2.8 and then further improve this analysis step-by-step to achieve **I2** and **I3**. We first turn to **I2** and restrict side-effecting to unknowns associated with mutexes to cases where the ego thread has possibly written a protected global since acquiring it. This is in contrast to Section 4.2.1, where a side-effect is triggered unconditionally at every unlock, i.e., everything a thread reads is treated as if that thread potentially wrote it.

To simplify the presentation, we first tie together the (abstract) thread *id*s $\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}$ computed by the digests from Section 2.8, and the (non-relational) thread *id* domain $\mathcal{V}^\sharp_{\mathbf{tid}}$ for which we, thus far, have only given soundness criteria and required that it be defined as a powerset domain over some set $S_{\mathcal{V}^\sharp_{\mathbf{tid}}}$. To tie these together, we further require a function $\mathsf{single} : \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \to \mathcal{V}^\sharp_{\mathbf{tid}}$ to translate from thread *id*s computed by digest to abstract thread *id*s as employed by the value domain. We demand that $\gamma_{\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}}\, i^\sharp \subseteq \gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}\, (\mathsf{single}\, i^\sharp)$ hold. There are several possible choices for $S_{\mathcal{V}^\sharp_{\mathbf{tid}}}$ and single here: One option is setting $S_{\mathcal{V}^\sharp_{\mathbf{tid}}} = \{\bullet\}$ and $\mathsf{single}\, x = \{\bullet\}$ for all $x \in \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}$. This amounts to not tracking any thread *id*s in the value domain at all.

The choice we make subsequently is setting $S_{\mathcal{V}^\sharp_{\mathbf{tid}}} = \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}$ and setting $\mathsf{single}\, x = \{x\}$, which amounts to tracking sets of thread *id*s as computed by the digest for variables of type thread *id*. As a consequence of the structure of the domain and the disjointness of concretizations of values in $\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}$ as required in (2.14) in Section 2.8, we obtain

$$(\gamma_{\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}}\, i^\sharp) \cap (\gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}\, I^\sharp) \neq \emptyset \implies i^\sharp \in I^\sharp \tag{4.3}$$

which is later exploited in the right-hand sides corresponding to returning from a thread. Given these definitions, at any program point corresponding to a call to return, the abstract value tracked for self given by $i' = (\mathsf{unlift}\, r)\, \mathsf{self}$ will be at most as precise as the thread *id* component i of the digest (i, C) associated with this unknown. We thus replace unknowns $[i', (i, C)]$ with unknowns $[(i, C)]$ to avoid the hassle of dealing with two separate types of thread *id*s forming part of some unknowns — without getting any precision advantage.

The new analysis now locally tracks an abstraction of the last thread-local writes via a map $L : (\mathcal{M} \times \mathcal{Q}) \to \mathcal{R}$, where $L\,(a, Q)$ maintains for a mutex a, an abstract relation between the globals in cluster $Q \in \mathcal{Q}_a$. More specifically, the abstract relation on the

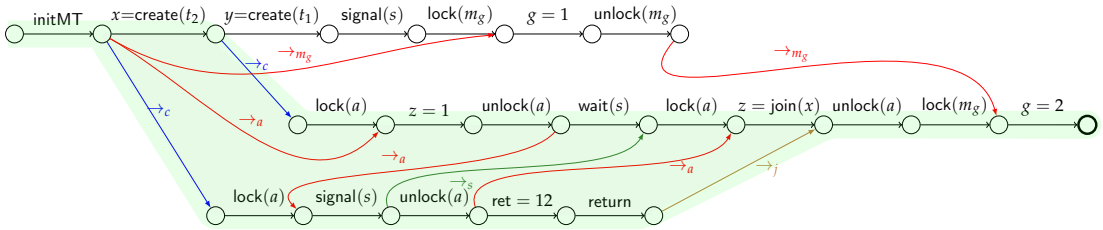


Figure 4.3: Illustration highlighting the *join-local* part of a local trace of the program from Fig. 2.7, and which writes are thus accounted for by L.

globals from Q recorded in $L\,(a, Q)$ is the one that held when a was unlocked *join-locally* for the *first* time after the *last join-local* write to a global in $\bar{\mathcal{G}}[a]$. If there is no such $\mathsf{unlock}(a)$, the relation at program start is recorded. We call an operation in a local trace *join-local* to the ego thread, if it is (a) *thread-local*, i.e., performed by the ego thread, or (b) is executed by a thread that is (transitively) joined into the ego thread, or (c) is *join-local* to the parent thread at the node at which the ego thread is created. This notion will also be crucial for realizing **I3**. *Join-locality* is illustrated in Fig. 4.3, where the *join-local* part of a local trace is highlighted. For *join-local* contributions, it suffices to consult $L\,a$ instead of unknowns $[a, Q, (i, C)]$. Such contributions are called *accounted* for. To check whether a contribution from some thread *id* is *accounted* for, we introduce a function $\mathsf{acc} : (\mathcal{A} \times \mathcal{D}_S) \to \mathcal{A} \to \mathbf{bool}$ (see definition (4.4) below), where $\mathcal{D}_S$ denotes the information tracked by the analysis in the local states, i.e., for unknowns of the form $[u, S, A]$. Besides an abstract value from $\mathcal{R}$, the local state $\mathcal{D}_S$ now contains two additional components:

- A map $L : (\mathcal{M} \times \mathcal{Q}) \to \mathcal{R}$ as outlined above with the join given component-wise;
- A set $W : 2^{\mathcal{G}}$ (ordered by $\subseteq$) of globals that may have been written since one of their protecting mutexes has been locked, and not all protecting mutexes have been unlocked since.

L and W are also abstractions of reaching local traces. The values tracked for unknowns $[(i, C)]$ for $(i, C) \in \mathcal{A}$ and $[s, A]$ for $s \in \mathcal{S}$ are also enhanced with an L component, while the information associated with unknowns for mutexes remains unmodified.

We sketch the right-hand sides here; definitions are given in Figs. 4.4 and 4.5. For the **initialization action** initMT, in contrast to the analysis from Section 4.2.2, there is no initial side-effect to the unknowns for mutexes. The initial values of globals are *join-local*, and thus accounted for in the L component produced by $\mathsf{init}^\sharp$ that is also passed to any subsequently created thread.

The right-hand sides for **thread creation** and **return** differ from the analysis from Section 4.2.2 enhanced with thread *id*s only in the handling of additional data structures L and W. As remarked before, the information $(i, C) \in \mathcal{A}$ is now directly used for thread *id*s and thus $i^\sharp \in \mathcal{V}^\sharp_{\mathbf{tid}}$ are no longer used for unknowns.

$$
\begin{array}{l}
\mathsf{init}((i,C))^\sharp\,\eta = \\
\quad \textbf{let } L\,(a,Q) = [\![\{g \leftarrow 0 \mid g \in Q\}]\!]^\sharp_{\mathcal{R}} \top \textbf{ in} \\
\quad \textbf{let } r = [\![\mathsf{self} \leftarrow^\sharp \mathsf{single}\, i]\!]^\sharp_{\mathcal{R}} \top \textbf{ in} \\
\quad (\emptyset, (\{(a,Q) \mapsto L\,(a,Q) \mid a{\in}\mathcal{M}, Q{\in}\mathcal{Q}_a\}, \emptyset, r)) \\
\\
[\![[u,S,(i,C)], \mathsf{initMT}, (i,C)]\!]^\sharp\eta = (\emptyset, \eta\,[u,S,(i,C)]) \\
\\
[\![[u,S,(i,C)], x{=}\mathsf{create}(u_1)]\!]^\sharp\eta = \\
\quad \textbf{let } (L,W,r) = \eta\,[u,S,(i,C)] \textbf{ in} \\
\quad \textbf{let } (i',C') = \mathsf{new}^\sharp_{\mathcal{A}}\ u\ u_1\,(i,C) \textbf{ in} \\
\quad \textbf{let } r' = ([\![\mathsf{self} \leftarrow^\sharp (\mathsf{single}\, i')]\!]^\sharp_{\mathcal{R}} r)\Big|_{\mathcal{X}} \textbf{ in} \\
\quad \textbf{let } \rho{=}\{[u_1,(\emptyset,(i',C'))]{\mapsto}(L,\emptyset,r')\} \textbf{ in} \\
\quad (\rho, (L, W, [\![x \leftarrow^\sharp \mathsf{single}\, i']\!]^\sharp_{\mathcal{R}} r)) \\
\\
[\![[u,S,(i,C)], \mathsf{return}, (i,C)]\!]^\sharp\eta = \\
\quad \textbf{let } (L,W,r) = \eta\,[u,S,(i,C)] \textbf{ in} \\
\quad \textbf{let } v = r|_{\{\mathsf{ret}\}} \textbf{ in} \\
\quad \textbf{let } \rho = \{[(i,C)] \mapsto (L,v)\} \textbf{ in} \\
\quad (\rho, (L,W,r)) \\
\\
[\![[u,S,(i,C)], x' = \mathsf{join}(x), (i',C')]\!]^\sharp\eta = \\
\quad \textbf{let } (L,W,r) = \eta\,[u,S,(i,C)] \textbf{ in} \\
\quad \textbf{if } i' \notin ((\mathsf{unlift}\, r)\, x) \textbf{ then} \\
\qquad (\emptyset, \underline{\bot}) \\
\quad \textbf{elseif } \mathsf{acc}\,((i,C),(L,W,r))\,(i',C') \textbf{ then} \\
\qquad (\emptyset, \underline{\bot}) \\
\quad \textbf{else} \\
\qquad \textbf{let } (L',v) = \eta[(i',C')] \textbf{ in} \\
\qquad \textbf{let } r' = [\![x' \leftarrow^\sharp (\mathsf{unlift}\, v)\ \mathsf{ret}]\!]^\sharp_{\mathcal{R}} r \textbf{ in} \\
\qquad (\emptyset, (L \sqcup L', W, r'))
\end{array}
$$

Figure 4.4: Right-hand sides for improved (**I1**, **I2**) analysis using thread *id*s.

For **join**, if the return value of the thread is not accounted for, it is assigned to the variable on the left-hand side and the L information from the thread for which join is called is joined into the information maintained by the ego thread. If, on the other hand, it is accounted for, the thread cannot be joined here in the concrete, and the value $\underline{\bot}$ is returned to denote unreachability. There is a separate constraint for each (i', C'), ensuring all threads that could be joined are considered.

For **locking** of mutexes, upon lock, if (i', C') for the corresponding call to unlock is not accounted for, its information on the globals protected by a is joined with the *join-local* information for a maintained in $L\,(a, Q)$, $Q \in \mathcal{Q}_a$. This information about the

$$
\begin{array}{l}
[\![[u,S,(i,C)],\mathsf{lock}(a),(i',C')]\!]^\sharp\eta = \\
\quad \textbf{let } (L,W,r) = \eta\,[u,S,(i,C)] \textbf{ in} \\
\quad \textbf{let } r' = \textbf{if } \mathsf{acc}\,((i,C),(L,W,r))\,(i',C') \textbf{ then } \bot \textbf{ else } \bigsqcap_{Q\in\mathcal{Q}_a} \eta\,[a,Q,(i',C')] \textbf{ in} \\
\quad \left(\emptyset,\left(L,W,r\sqcap\left(\left(\bigsqcap_{Q\in\mathcal{Q}_a} L\,(a,Q)\right)\sqcup r'\right)\right)\right) \\
[\![[u,S,(i,C)],\mathsf{unlock}(a),(i,C)]\!]^\sharp\eta = \\
\quad \textbf{let } (L,W,r) = \eta\,[u,S,(i,C)] \textbf{ in} \\
\quad \textbf{let } (\rho,L') = \textbf{if } \bar{\mathcal{G}}[a]\cap W{=}\emptyset \textbf{ then} \\
\qquad\quad (\emptyset,L) \\
\qquad \textbf{else} \\
\qquad\quad (\{[a,Q,(i,C))] \mapsto r|_Q \mid Q\in\mathcal{Q}_a\}, L\oplus\{(a,Q)\mapsto r|_Q \mid Q\in\mathcal{Q}_a\}) \\
\quad \textbf{in} \\
\quad \textbf{let } r' = r|_{\mathcal{X}\cup\bigcup\{\bar{\mathcal{G}}[a']\mid a'\in(S\setminus a)\}} \textbf{ in} \\
\quad \textbf{let } W'{=}\{g \mid g{\in}W, \bar{\mathcal{M}}[g]\cap S\setminus\{a\}{\neq}\emptyset\} \\
\quad \textbf{in } (\rho,(L',W',r')) \\
[\![[u,S,(i,C)],g = x]\!]^\sharp\eta = \\
\quad \textbf{let } (L,W,r) = \eta\,[u,S,(i,C)] \textbf{ in} \\
\quad (\emptyset,(L,W\cup\{g\},[\![g\leftarrow x]\!]^\sharp_{\mathcal{R}}\,r)) \\
[\![[u,S,(i,C)],x = g]\!]^\sharp\eta = \\
\quad \textbf{let } (L,W,r) = \eta\,[u,S,(i,C)] \textbf{ in} \\
\quad (\emptyset,(L,W,[\![x\leftarrow g]\!]^\sharp_{\mathcal{R}}\,r)) \\
[\![[u,S,(i,C)],\mathsf{signal}(s),(i,C)]\!]^\sharp\eta = \\
\quad \textbf{let } (L,W,r) = \eta\,[u,S,(i,C)] \textbf{ in} \\
\quad (\{[s,(i,C)]\mapsto(L,r)\},(L,W,R)) \\
[\![[u,S,(i,C)],\mathsf{wait}(s),(i',C')]\!]^\sharp\eta = \\
\quad \textbf{let } (L,W,r) = \eta\,[u,S,(i,C)] \textbf{ in} \\
\quad \textbf{if } \eta\,[s,(i',C')] = \bot \textbf{ then} \\
\qquad\quad (\emptyset,\bot) \\
\quad \textbf{elseif } \mathsf{acc}\,((i,C),(L,W,r))\,(i',C') \textbf{ then} \\
\qquad\quad (\emptyset,\bot) \\
\quad \textbf{else} \\
\qquad\quad (\emptyset,(L,W,r))
\end{array}
$$

Figure 4.5: Right-hand sides for improved (**I1**, **I2**) analysis using thread *id*s (continued).

globals protected by a is then incorporated into the local state by $\sqcap$.

For **unlocking** of mutexes, if there may have been a write to a protected global since the mutex was locked (according to W), the *join-local* information L is updated and the

local state restricted to Q is side-effected to the appropriate unknown $[a, Q, (i, C)]$ for $Q \in \mathcal{Q}_a$. Just like in Section 4.2.1, r is then restricted to only maintain relationships between locals and those globals for which at least one protecting mutex is still held. **Reading** from and **writing** to globals once more are purely local operations, where all writes are recorded into W. **Wait** and **signal** remain essentially unmodified, with the difference that, in case (i', C') is accounted for, the successor state is flagged a dead code.

To exclude self writes, we set

$$\mathsf{acc}\,((i, C), \_)\,(i', C') = \mathsf{unique}\, i \wedge i = i' \tag{4.4}$$

The resulting analysis thus takes **I1** (via $[\![\cdot]\!]^\sharp_{\mathcal{A}}$ defined in Eq. (2.15)), as well as **I2** (via acc) into account. This analysis can now show all assertions in Example 26.

Theorem 13. *This analysis is sound w.r.t. the local trace semantics.*

Proof. The proof relies on the following observations:

- When $\bar{\mathcal{G}}[a] \cap W = \emptyset$, no side-effect is required. Any thread locking a following this unlock, will either consult unknowns that the written values were side-effected to at the unlock a immediately succeeding the write, or will have accounted for those values via $L\,a$.
- Exclusions based on acc are sound, i.e., it only excludes *join-local* writes.

The detailed proof is a simplification of the proof for the enhanced analysis also considering joins from Section 4.2.4 which we outline in Section 6.2.2. □

The analysis, as presented thus far, does not make use of components C forming part of unknowns $[a, Q, (i, C)]$ for mutexes $a \in \mathcal{M}, Q \in \mathcal{Q}_a$, $i \in \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}$ and unknowns $[i, C]$ for $i \in \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}$ associated with thread returns. This information can be exploited to exclude a further class of writes, namely, those that are performed by a thread involved in the creation of the ego thread or its ancestor — before the ego thread or this ancestor was created. Any writes that the ego thread may read from the creating thread before the ego thread is created are already accounted for in the start state of the ego thread, so only those writes happening after the creation of the ego thread need to be read via the respective unknowns. To this end, one sets

$$\begin{aligned}\mathsf{acc}\,((i, C), (L, W, r))\,(i', C') = (\mathsf{unique}\, i \wedge i = i') \vee \\ (\mathsf{lcu\_anc}\, i'\, i = i' \wedge\ \nexists \langle u, u' \rangle \in C' : ((i' \circ \langle u, u' \rangle) = i) \\ \vee \mathsf{may\_create}\,(i' \circ \langle u, u' \rangle)\, i)\end{aligned} \tag{4.5}$$

Example 27. *Consider the following program where $\bar{\mathcal{M}}[g] = \{a, m_g\}$ and $\bar{\mathcal{M}}[h] = \{a, m_h\}$ and assume $\mathcal{Q}_a = \{\{g, h\}\}$ and that the domain $\mathcal{R}$ from Example 11 is used.*

```
main:                          t1:
 initMT;                        lock(a);
 lock(a);                       // ASSERT(g==h); (1)
 g = 5; h = 8;                  unlock(a);
 unlock(a);                     return;
 lock(a);
 g = 10; h = 10;
 unlock(a);
 x = create(t1);
 lock(a);
 g = 20; h = 20;
 unlock(a);
 y = join(x);
 lock(a);
 // ASSERT(g==20); (2)
 unlock(a);
```

Here, if we make use of the additional information on which create edges have been encountered, the analysis can determine that assertion (1) in thread t1 holds. □

Remark 21. *If the additional information contained in C components is not exploited and the original definition of* acc *(Eq.* (4.4)*) instead of the improved one (Eq.* (4.5)*) is employed, an implementation may selectively abandon control-point splitting according to C at mutexes, thread* ids*, and signals. This amounts to replacing unknowns* $[a, Q, (i, C)]$*,* $[i, C]$*, and* $[s, (i, C)]$ *with* $[a, Q, i]$*,* $[i]$*, and* $[s, i]$*, respectively. This change may yield an improved runtime and will not impact precision beyond, e.g., the effects of widening, which we consider out of scope for this thesis.*

The analysis still may incur an unnecessary loss of precision when past writes of a thread are propagated to a thread it creates and then back to the creating thread upon a call to join. In particular, this may happen when the created thread has not written to some globals, but the original thread has overwritten them locally in the meantime.

Example 28. *Consider again Example 27. Here, the assertion (2) cannot be proven, as the stale information in* $L\,(a, \{g, h\})$ *in* t_1 *(which includes the fact that g may be* 10*) is incorporated into the main thread upon join (as well as equivalent information from* $L\,(m_g, \{g\})$*). This can be prevented by tracking, for each thread, the set* $\bar{W}$ *of global variables that* may *have been written by it or any thread it has joined, and then only joining in the L information of the joined thread if at least one protected global has been written.* □

We do not detail this improvement here, but use it in our implementation.

Remark 22. *Further useful abstractions to maintain in the ego thread may, e.g., track for each created thread* t'*, the set* $W_C\,t'$ *of globals that have been potentially written to by the* join-local *part of the ego thread since the creation of* t'*. Then, upon joining* t'*, for mutexes a and clusters* $Q \in \mathcal{Q}_a$ *where* $Q \cap W_C\,t' = \emptyset$*, the* $L\,(a, Q)$ *information of the joined thread definitely contains the most up-to-date information, and* $L\,(a, Q)$ *of the ego thread can be discarded.*

4.2.4 Exploiting Thread *ID*s and Joins to Improve Relational Analyses

Thus far, improvement **I3** from Section 4.2.3 remains unrealized. The critical insights needed to realize it are that, after a thread has been joined, it can no longer perform any writes and that all writes of joined threads are *join-local* to the ego thread. It thus is not necessary to read from the global unknowns associated with a thread with a unique thread *id* i, if the thread with thread *id* i has definitely been joined.

We, therefore, enhance the analysis to also track in local states and return states of threads a set J of thread *ids* of threads for which join has definitely been called in the *join-local* part of the local trace. For unique thread *id*s in J, the associated unknowns are not consulted when determining which value to read. To achieve this, we set

$$\mathsf{acc}\left((i,C),(J,L,W,r)\right)(i',C') = \mathsf{unique}\, i' \wedge (i = i' \vee i' \in J)$$

Technically, we track a set $J \in 2^{\mathcal{V}^\sharp_{\mathsf{tid},\mathcal{A}}}$ with the order given by *super*set inclusion that is initially empty. We proceed to give the right-hand sides corresponding to return and join; all other right-hand sides propagate this value locally and pass it as an additional argument to acc.

$$\begin{array}{rcl}
[\![[u,S,(i,C)],\mathsf{return},(i,C)]\!]^\sharp\eta & = & \textbf{let } (J,L,W,r) = \eta\,[u,S,(i,C)] \textbf{ in} \\
& & \textbf{let } v = r|_{\{\mathsf{ret}\}} \textbf{ in} \\
& & \textbf{let } \rho = \{[(i,C)] \mapsto (J,L,v)\} \textbf{ in} \\
& & (\rho,(J,L,W,r))
\end{array}$$

$$\begin{array}{rcl}
[\![[u,S,(i,C)],x' = \mathsf{join}(x),(i',C')]\!]^\sharp\eta & = & \textbf{let } (J,L,W,r) = \eta\,[u,S,(i,C)] \textbf{ in} \\
& & \textbf{if } i' \notin ((\mathsf{unlift}\, r)\, x) \textbf{ then} \\
& & \quad (\varnothing,\bot) \\
& & \textbf{elseif } \mathsf{acc}\left((i,C),(J,L,W,r)\right)(i',C') \textbf{ then} \\
& & \quad (\varnothing,\bot) \\
& & \textbf{else} \\
& & \quad \textbf{let } (J',L',v) = \eta[(i',C')] \textbf{ in} \\
& & \quad \textbf{let } r' = [\![x' \leftarrow^\sharp (\mathsf{unlift}\, v)\ \mathsf{ret}]\!]^\sharp_{\mathcal{R}} r \textbf{ in} \\
& & \quad (\varnothing,(J \cup J' \cup \{i'\}, L \sqcup L', W, r'))
\end{array}$$

We remark that, in the analysis, when performing a thread join, if there are different thread *id*s for which join might be called, there is one constraint for each, and the resulting values are joined to obtain the abstract state for the control flow successor. As the lattice *join* for J is intersection, this naturally handles the case where x can take values of thread *id*s of different threads, in that their J' components are effectively intersected.

Example 29. *Consider the following program where* $\bar{\mathcal{M}}[g] = \{a, m_g\}$ *and* $\bar{\mathcal{M}}[h] = \{a, m_h\}$ *and assume* $\mathcal{Q}_a = \{\{g,h\}\}$ *and that the relational domain* $\mathcal{R}$ *from Example 11 is used.*

```
main:                          t1:
 initMT;                        lock(a);
 x = create(t1);                g = 4; h = 8;
 lock(a);                       unlock(a);
 g = 20; h = 20;                x = ?;
 unlock(a);                     lock(a);
 y = join(x);                   g = x; h = x;
 lock(a);                       unlock(a);
 // ASSERT(g==h); (1)           return;
 g = 5; h = 5;
 unlock(a);
 lock(a);
 // ASSERT(g==5); (1)
 unlock(a);
```

Here, both assertions can be proven. At (1), the thread t_1, is must-joined. Its last write is accounted for in $L\,(a, \{g, h\})$, thus the unknown $[a, \{g, h\}, t_1]$ where the abstract relationship $g = h$ does not hold is not consulted. As the updates in L are destructive, after the main *thread writes* 5 *to g, this is also the only value it reads for g, meaning (2) is proven as well.* □

Remark 23. *To further improve the precision, it would suffice to join in L from joined threads when they have a unique thread* id*, as in other cases, the corresponding unknowns will be consulted anyway. Additionally, the set J of must-joined threads could be published together with the protected globals at an unlock. In this way, a thread need not read from another thread that does all its writes after the first thread has already been* must*-joined. In the interest of brevity, we do not further outline either of these ideas here.*

Remark 24. *The J component and* acc *could also be used in the right-hand side for* wait *to exclude those signals that can not happen in parallel to the thread calling wait. This would correspond to checking the same two conditions as for* join*. We do not elaborate on this here.*

Theorem 14. Mutex-Meet *enhanced with thread* ids *and tracking joined threads is sound w.r.t. the local trace semantics.*

Proof. The proof idea is the same as outlined in the preceding section for Theorem 13. The detailed proof is outlined in Section 6.2.2. □

Remark 25. *When during a call to join* $\mathsf{acc}\,((i, C), (J, L, W, r))\,(i', C')$ *holds, this is an indication that — provided the corresponding program point is reached —* join *is called for a thread that has definitely already been joined before. An analysis may produce a warning for this type of bug. To detect all cases where join* may *be called more than once for the same thread is more difficult and may be an interesting future topic.*

4.2.5 Exploiting Clustered Relational Domains

Naïvely, one might assume that tracking relations among a larger set of globals is necessarily more precise than between smaller sets. Interestingly, this is no longer true

for our analyses, e.g., in the presence of thread *id*s. A similar effect where relating more globals can deteriorate precision has also been observed in the context of an analysis using a data-flow graph to model interferences [45].

To simplify the presentation, we in this section consider again the analysis not exploiting thread *joins* from Section 4.2.3 — but use all improvements in our implementation.

Example 30. *Consider again Example 23 with* $\mathcal{Q}_a = \{\{g, h, i\}\}$. *We reproduce the corresponding source code here for convenience:*

```
main:                                    t1:          t2:
 initMT;                                  lock(a);     lock(a);
 x = create(t1); y = create(t2);          x = h;       g = ?; h = ?;
 lock(a);                                 i = x;       unlock(a);
 g = ?; h = ?; i = ?;                     unlock(a);   return;
 unlock(a); r = join(y); lock(a);         return;
 z = ?; g = z; h = z; i = z;
 unlock(a); lock(a);
 // ASSERT(h==i); (1)
 // ASSERT(g==h); (2)
 unlock(a);
```

For this program, the constraint system of the analysis presented thus far has a unique least solution, as all right-hand sides are monotonic. It verifies that assertion (1) *holds. It assures for* $[a, \{g, h, i\}, t_1]$ *that* $h{=}i$ *holds, while for the* main *thread and the program point before each assertion,* $L\,(a, \{g, h, i\}) = \{g{=}h, h{=}i\}$ *holds, while for* $[a, \{g, h, i\}, \mathsf{main}]$ *and* $[a, \{g, h, i\}, t_2]$ *only* $\top$ *is recorded, as is for any relation associated with* m_g, m_h, *or* m_i.

Assertion (2), however, will not succeed, as the side-effect from t_1 *causes the older values from the first write in the main thread to be propagated to the assertions as well, implying that while* $h{=}i$ *is proven,* $g{=}h$ *is not.* □

Intuitively, the analysis loses precision because, at a program point where a mutex a is unlocked, the current relationships between *all* clusters protected by a are side-effected. As soon as *one* global is written to, the analysis behaves as if *all* protected globals had been written. By only publishing values to affected clusters, i.e., clusters that contain variables that may have been written, more precise information may remain for unaffected clusters.

Accordingly, in the improved analysis, the side-effects at unlocks and the unknowns consulted at lock are modified as indicated in Fig. 4.6. When **unlocking** a mutex a, side-effects are only produced to clusters $Q \in \mathcal{Q}_a$ containing at least one global that was written to since the last $\mathsf{lock}(a)$. For **locking** the mutex a, the abstract value to be incorporated into the local state is assembled from the contributions of different threads to the clusters. For that, the separate constraints for each relevant digest from Eq. (4.2) are combined into one for the set $\mathbf{I} = \{(i', C') \mid (i, C) \in [\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}((i, C), (i', C'))\}$ of *all* relevant digests. This is necessary as side-effects to unaffected clusters at $\mathsf{unlock}(a)$

$$
\begin{aligned}
&[\![[u, S, (i, C)], \mathsf{unlock}(a), (i, C)]\!]^\sharp \eta = \\
&\quad \textbf{let } (L, W, r) = \eta\,[u, S, (i, C)] \textbf{ in} \\
&\quad \textbf{let } \mathcal{Q}' = \{Q \mid Q \in \mathcal{Q}_a, Q \cap W \neq \emptyset\} \textbf{ in} \\
&\quad \textbf{let } L' = L \oplus \{(a, Q) \mapsto r|_Q \mid Q \in \mathcal{Q}'\} \textbf{ in} \\
&\quad \textbf{let } \rho = \{[a, Q, (i, C)] \mapsto r|_Q \mid Q \in \mathcal{Q}'\} \textbf{ in} \\
&\quad \textbf{let } r' = r|_{\mathcal{X} \cup \bigcup \{\vec{\mathcal{G}}[a'] \mid a' \in (S \setminus a)\}} \textbf{ in} \\
&\quad \textbf{let } W' = \{W \mid g \in W, \bar{\mathcal{M}}[g] \cap S \setminus \{a\} \neq \emptyset\} \textbf{ in} \\
&\quad (\rho, (L', W', r'')) \\
\\
&[\![[u, S, (i, C)], \mathsf{lock}(a), \mathbf{I}]\!]^\sharp \eta = \\
&\quad \textbf{let } (L, W, r) = \eta\,[u, S, (i, C)] \textbf{ in} \\
&\quad \textbf{let } J(Q) = \bigsqcup \{\eta\,[a, Q, (i', C')] \mid (i', C') \in \mathbf{I}, \neg \mathsf{acc}\ (i, C)\ (L, W, r)\ (i', C')\} \textbf{ in} \\
&\quad \textbf{let } r' = \bigsqcap_{Q \in \mathcal{Q}_a} (J(Q) \sqcup L\,(a, Q)) \textbf{ in} \\
&\quad (\emptyset, (L, W, r \sqcap r'))
\end{aligned}
$$

Figure 4.6: Right-hand sides for unlocking and locking when limiting side-effecting to potentially written clusters.

have been abandoned and thus the meet with the values for clusters of one thread at a time is unsound. For each cluster Q, the join-local information $L\,(a, Q)$ is joined with all contributions to Q by threads that are not yet *accounted* for, but admitted for Q by the digests. Here, the contributions of threads that do *not* write Q are $\bot$, and thus do not affect the value for Q. Finally, the resulting value is used to improve the local state by *meet*. The right-hand side for $\mathsf{lock}(a)$ thus exploits the fine-grained, per-cluster MHP information provided by the digests and the predicate acc.

Remark 26. *For an intuition on why performing a cluster-wise join of the contributions of different threads before meeting across clusters is necessary here, consider the case where some thread has only published to* some *clusters associated with a mutex, not all. As side-effecting here is limited to modified clusters, clusters that are not written to have value $\bot$. In this case, first meeting would erroneously disregard* all *contributions of that thread.*

We obtain:

Theorem 15. *Given domains $\mathcal{R}$ and $\mathcal{V}^\sharp$ fulfilling the requirements from Fig. 3.2, any solution of the constraint system is sound w.r.t. the local trace semantics. Maximum precision is obtained with $\mathcal{Q}_a = 2^{\vec{\mathcal{G}}[a]}$.*

Proof. For an intuition-driven proof, consider modifying the program by introducing auxiliary mutexes a_Q for each considered cluster $Q \in \mathcal{Q}_a$ and mutex a that are locked and unlocked whenever a is and setting $\bar{\mathcal{M}}[a_Q] = Q$. Provided these mutexes are always locked and unlocked in the same order, the programs are semantically equivalent

for some intuitive notion of equivalence. Performing the analysis from the previous section on this modified program then simulates the behavior of the analysis described in this section on the original program and is sound according to Theorem 13 from Section 4.2.3.

In Section 6.2.3, we sketch a more principled proof that does not rely on this insight and also considers joins as in the previous section. □

For Example 23, with $\mathcal{Q}_a = 2^{\vec{\mathcal{G}}[a]}$, both assertions are verified. Performing the analysis with all subclusters simultaneously can be expensive when sets $\vec{\mathcal{G}}[a]$ are large. The choice of subclustering, thus, generally involves a trade-off between precision and runtime. This is different for k-decomposable relational domains:

Theorem 16. *Provided the relational domain is k-decomposable (Equation (3.5)), the clustered analysis using all subclusters of sizes at most k only, is equally precise as the clustered analysis using all subclusters $\mathcal{Q}_a = 2^{\vec{\mathcal{G}}[a]}$ at mutexes a.*

Proof. Consider a solution η of the constraint system with $\mathcal{Q}_a = 2^{\vec{\mathcal{G}}[a]}$. Then, for unknowns $[a, Q, (i, C)]$ and $[a, Q', (i, C)]$ with $Q \subseteq Q'$ and $|Q| \leq k$, and values $r = \eta\,[a, Q, (i, C)]$, $r' = \eta\,[a, Q', (i, C)]$, we have that $r \sqsubseteq r'|_Q$ (whenever the smaller cluster receives a side-effect, so does the larger one). Thus, by k-decomposability, the additional larger clusters Q', do not improve the *meet* over the clusters of size at most k for individual thread *id*s as well as the *meet* of their *join*s over all thread *id*s. The same also applies to the clustered information stored in L components. □

Example 31. *Consider again Example 23. If the analysis is performed with clusters $\mathcal{Q}_a = \{\{h, i\}, \{g, h\}, \{g, i\}, \{g\}, \{i\}, \{h\}\}$ both assertions can be proven.* □

The one-element clusters, on the other hand, cannot be abandoned, as indicated by the following example:

Example 32. *Consider the following program, assume that a protects both g and h and that the domain $\mathcal{R}$ from Example 11 (enhanced with two-variable inequalities) is used.*

```
main:                          t1:
 initMT;                        lock(a);
 x = create(t1);                g =- 1;
 y = create(t2);                //ASSERT(g<=h);  (3)
 lock(a);                       unlock(a);
 h = 31;                        return;
 unlock(a);
 lock(a);                      t2:
 h = 12;                        lock(a);
 unlock(a);                     h = ?;
 lock(a);                       h = 12;
 // ASSERT(g<=h);  (1)          unlock(a);
 // ASSERT(h==12); (2)          return;
 unlock(a);
```

When running the clustered analysis with the cluster $\mathcal{Q}_a = \{\{g, h\}\}$ alone, the side-effect at the $\mathsf{unlock}(a)$ *in t_1 preserves the relationship $g \leq h$, implying that the assertions (1) and (3) succeed. No precise information on the value of h is preserved at the unknown $[a, \{g, h\}, t_1]$. Consequently, when the* main *thread performs a* $\mathsf{lock}(a)$ *for the third time, the assertion (2) cannot be verified. A clustered analysis, though, that additionally tracks the cluster $\{h\}$, will record $h = 12$ at $[a, \{h\}, t_2]$ and have $\bot$ for $[a, \{h\}, t_1]$. Therefore, assertion (2) can successfully be verified — provided the one element clusters are considered as well.* □

5 Experimental Evaluation

For an experimental evaluation of the analyses, we pose the following research questions:

(RQ1) How do the different approaches in Section 4.1 compare to each other w.r.t. their runtime? What is the runtime impact of enabling/disabling thread *id*s? What impact does the considered domain have?

(RQ2) Can the thread *id* digest successfully identify threads? How many thread *id*s are computed? How many of those are unique?

(RQ3) How do the different approaches in Section 4.1 compare to each other w.r.t. their precision? What is the precision impact of enabling/disabling thread *id*s? What impact does the considered domain have?

(RQ4) How do the different analyses in Section 4.2 compare to each other w.r.t. their runtimes? What is the runtime impact of switching from intervals to octagons? What is that of additionally enabling thread *id*s? What is the additional overhead of considering clusters? How do these runtimes compare to the ones for the analysis in Section 4.1?

(RQ5) How do the different analyses in Section 4.2 compare to each other w.r.t. their precision? What is the precision impact of switching from intervals to octagons? What is that of additionally enabling thread *id*s? What is the precision impact of considering clusters? How does the precision compare to the ones for the analysis in Section 4.1?

To answer these research questions, we implemented all analyses presented in this thesis in the Goblint[1] static analyzer targeting multi-threaded C programs that employ the Pthreads library for concurrency. Goblint supports function pointers, dynamic creation of threads, dynamic allocation of memory, and many other features of the C programming language — meaning it can be used to analyze real-world programs. Goblint performs a context-sensitive interprocedural analysis, where the degree of context-sensitivity can be configured to tune the precision-performance trade-off. The system's modular architecture allows different analyses to be activated and deactivated at runtime and provides a separation of concerns between specifying analyses and other aspects, such as the solver used to solve the side-effecting constraint system [8]. All activated analyses are run at the same time and can communicate with each other through a system of queries and events.

[1] `https://goblint.in.tum.de/` and `https://github.com/goblint/analyzer`

Goblint is implemented in OCaml and makes use of an updated fork[2] of Cil [93] for parsing and simplification of the input program. Goblint comes both with its own custom implementations of some relational domains and an interface to the Apron library [64]. For the experiments using relational domains, we relied on the implementation of the Octagon domain supplied by Apron.

To allow for the analysis of real-world programs, the analyses as presented in this thesis were extended in a variety of ways. Thus, the implementations sometimes slightly deviate from the presentation in the previous chapter, e.g., in the following ways:

- The implementation supports arbitrary data types, including pointers. Thus, it is not possible to actually add mutexes m_g in a pre-processing step, as a pointer dereference at some point in the code may access no, one, or multiple different globals, depending on the runtime value of the pointer variable. Therefore, we consider m_g mutexes conceptually only and do not insert them into the code. For compound datatypes, CIL decomposes accesses into accesses to individual members, meaning no special handling is required.

- The implementation uses machine integers as mandated by the C standard, as opposed to mathematical integers used in this thesis. The handling of signed overflows is configurable: Goblint offers options to flag overflows. Additionally, it allows the user to specify how execution is supposed to continue once a potential overflow has occurred — either by assuming all overflows were due to imprecision or by assuming overflows cause involved variables to have the value $\top$.

- The set of variables that need to be handled as globals in C programs consists not just of the syntactically global variables of the program, but also includes local variables that escape their thread. Goblint comes with support for detecting such escaping variables, and we have extended all approaches to handle the dynamic discovery of additional global variables during analysis, which is also needed to handle heap-allocated data. Additional globals being discovered during analysis also required us to implement support for heterogeneous relational domains in the sense of Journault et al. [65]. A *strengthening* as proposed by them is also implemented for our relational domains in Goblint, though we usually do not enable it, as it is expensive and seems to offer only a small precision boost.

- We added support for function calls: Here, we perform a context-sensitive abstract interpretation using partial tabulation, a variant of the functional approach as proposed by Sharir and Pnueli [118]. For our experiments, we use the complete local abstract state of the caller as the context. As the framework allows for partial contexts [9, 43] as well, it may be interesting to study the impact of only considering *parts* of the local states for the context, e.g., only the part of the state abstracting values of variables.

[2] `https://github.com/goblint/goblint-cil`

- In C, locking and unlocking of mutexes happens through *pointers* to mutexes. Goblint splits by the different mutexes in the points-to set of the pointer arguments to functions locking and unlocking mutexes, and considers each of the resulting paths separately, akin to how the analysis splits unknowns according to lockset. When mutexes are unlocked that are not currently held, a warning is emitted.
- Pthreads offers different types of mutexes (e.g., recursive vs. non-recursive mutexes) [24]. We adapt the analyses to remain sound for recursive mutexes, by, among other changes, considering lock operations of potentially recursive mutexes as no-ops, provided the mutex is already held.
- Thread creation also works through a pointer in C. Thus, at a thread creation point, threads executing different procedures may be started.
- The system already computes the set of mutexes that are always held when a given global is accessed. This information is computed during fixpoint iteration by starting with the full set of mutexes and then intersecting the currently held set with it upon every access (akin to what is outlined in Remark 9). Deviating from the presentation in the previous chapter, we took care to implement all analyses in a way that is able to deal with the set of protecting mutexes shrinking dynamically during the analysis. It is conceivable that the set of protecting mutexes may grow again during analysis when certain code previously considered reachable is discovered to be, in fact, unreachable. The framework does currently not benefit from protecting sets growing again. However, we do not expect the precision gain from supporting this feature to be high.
- Goblint in its base configuration already computes thread *id*s as explained in Section 2.8, so they are always computed for unknowns corresponding to program points but are only used for other unknowns when the respective analysis is configured to exploit them.
- While implemented in Goblint, the handling of condition variables was not activated for this evaluation, as the semantics of C allows for spurious wakeups. Therefore, the analyses would only be able to produce warnings about spuriously live fragments of code, i.e., fragments that can only be live because of spurious wakeups. Such warnings would, however, likely be the same for all approaches.

A preliminary version of this evaluation was already reported on in [107, 108]. In this thesis, we conduct the experiments with a more recent version of Goblint, consider additional analyses, and an extended set of benchmarks.

The benchmarks were conducted on a machine with two Intel Xeon Platinum 8260 @ 2.40 GHz processors with 24 cores each and 256 GB of RAM running Ubuntu 18.04.6 LTS. Goblint was compiled with OCaml 4.14.0 with flambda activated. We remark that Goblint itself is single-threaded, and so the runtime on consumer-grade hardware is expected to be in the same range as on the server used for the evaluation. The version of

Table 5.1: Larger benchmarks from the Goblint suite and from SV-COMP.

Goblint benchmark suite (Posix)

Name	Lines	LLoC	Description
`pfscan`	1295	562	Parallel file scanner
`aget`	1280	587	Multi-threaded HTTP download accelerator
`ctrace`	1407	657	C Tracing library sample program
`knot`	2255	981	Multi-threaded webserver
`ypbind`	6588	992	Linux NIS binding process
`smptrc`	5787	3037	SMTP Open Relay Checker

SV-COMP (Tasks generated from Linux device drivers by LDV toolchain [135])

Name	Lines	LLoC	Description (Driver for)
`iowarrior`	7687	1345	IOWarrior devices from Code Mercenaries
`adutux`	8114	1520	ADU devices from Ontrak Control Systems
`w83977af`	10071	1501	Winbond W83977AF Super I/O chip
`tegra20`	7111	1547	Nvidia's Tegra20/Tegra30 SLINK Controller
`nsc`	12778	2379	Driver for the NSC PC'108 and PC'338 IrDA chipsets
`marvell1`	12246	2465	CMOS camera controller in Marvell 88ALP01 chip
`marvell2`	12256	2465	CMOS camera controller in Marvell 88ALP01 chip

Goblint used for the evaluation[3], as well as the repository containing the benchmarks[4] are available on GitHub. Both repositories and the scripts and intermediate data needed to produce the results in this chapter, are available as an artifact on Zenodo [104].

5.1 Description of the Benchmark Sets

This section describes two sets of benchmarks that play at least some part in answering all research questions outlined above. Both sets have already been used in the literature and are comprised of not-too-small real-world multi-threaded programs.

Goblint benchmarks. This set consists of six multi-threaded Posix programs from the Goblint benchmark suite[5] and seven large benchmarks used in the international competition on software verification (SV-COMP) 2021 [14]. The latter tasks are generated from Linux device drivers by the LDV toolchain [135] and form part of the `c/ldv-linux-3.14-races/` folder from the ConcurrencySafety-Main category[6]. This set was already used in [107, 108].

[3] `https://github.com/goblint/analyzer/tree/michael-schwarz-dissertation`
[4] `https://github.com/goblint/bench/tree/michael-schwarz-dissertation`
[5] `https://github.com/goblint/bench`
[6] `https://github.com/sosy-lab/sv-benchmarks`

Table 5.2: Larger benchmarks from the CONCRAT [61] suite.

Name	Lines	LLoC	Description
AirConnect	17954	7512	Bridge to use AirPlay with UPnP devices
axel	6004	2716	CLI download accelerator
brubeck	5879	2240	Statistics aggregator
C-Thread-Pool	749	241	Minimal threadpool implementation
cava	4858	2011	Cross-platform Audio Visualizer
clib	25773	11090	Package manager for C
dnspod-sr-fixed	9473	4698	DNSPod Security Recursive DNS Server
dump1090	4777	2079	Decoder for Software Defined Radio
EasyLogger	2140	839	High-performance C log library
fzy	2765	1077	Fuzzy finder for the terminal
klib	736	293	A standalone and lightweight C library
level-ip	5699	2452	A userspace TCP/IP stack
libaco	1302	667	C asymmetric coroutine library
libfaketime	528	143	Modifies the system time for a single application
libfreenect	646	245	Drivers and libraries for the Xbox Kinect device
lmdb	11021	5748	Memory-Mapped Database
minimap2	17596	9081	Aligner for DNA or mRNA sequences [77]
Mirai-Source-Code-fixed	1876	820	Source Code of the Mirai malware
nnn	12293	6712	Terminal file manager
phpspy	19695	9551	Sampling PHP profiler
pianobar	11663	4382	Console-based music streaming player
pigz-fixed	9232	5014	A parallel implementation of gzip
pingfs	2403	913	Filesystem storing information in ICMP ping packets
ProcDump-for-Linux	4220	2157	Linux version of ProcDump
Remotery	7562	3531	CPU/GPU profiler with Remote Web View
shairport	8902	3791	An AirPlay audio player for Linux
siege	19880	9239	Load tester for HTTP servers
snoopy	3638	1938	Library to log program executions
sshfs	7451	3258	Network filesystem client
streem	20803	9185	Prototype stream-based programming language
sysbench-fixed	16340	3575	Database and system performance benchmark
the_silver_searcher	7396	3615	ack-like code search
uthash	822	476	Utilities for working with hashtables in C
vanitygen	11163	5160	Vanity address generator for Bitcoin
wrk	8883	3747	Load tester for HTTP servers
zmap	17908	7183	Network scanner [40]

Table 5.3: Tasks from the Concrat suite for which none of the approaches terminated normally within the 15 min time limit. The symbol ☰ denotes termination due to a stack overflow, whereas 🐞 denotes termination due to an internal error of the analyzer or the compiler frontend, and ⧗ is used to indicate timeout.

Name	(Protection)	(Miné)	(Lock)	(Write)	(Combined)	(Protection-I)	(Miné-I)	(Lock-I)	(Write-I)	(Combined-I)	(Protection-TID)	(Lock-TID)	(Write-TID)	(Combined-TID)	(Protection-I-TID)	(Lock-I-TID)	(Write-I-TID)	(Combined-I-TID)
`AirConnect`	☰	☰	☰	☰	☰	☰	☰	☰	⧗	⧗	☰	☰	☰	☰	☰	⧗	⧗	⧗
`axel`	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗
`clib`	🐞	🐞	🐞	🐞	🐞	🐞	🐞	🐞	🐞	🐞	🐞	🐞	🐞	🐞	🐞	🐞	🐞	🐞
`minimap2`	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗
`phpspy`	⧗	⧗	⧗	⧗	⧗	☰	☰	☰	☰	☰	⧗	⧗	⧗	⧗	☰	☰	☰	☰
`Remotery`	☰	☰	☰	☰	☰	☰	☰	☰	☰	☰	☰	☰	☰	☰	☰	☰	☰	☰
`sshfs`	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗	⧗
`streem`	☰	☰	☰	☰	☰	⧗	⧗	⧗	⧗	⧗	☰	☰	☰	☰	⧗	⧗	⧗	⧗
`the_silver_searcher`	☰	⧗	⧗	⧗	⧗	☰	☰	☰	☰	☰	☰	⧗	⧗	⧗	☰	☰	☰	☰

Concrat benchmarks. On top of this, we used the benchmarks assembled by Hong and Ryu [61] for evaluating an automatic C-to-Rust translator for concurrent programs. These benchmarks were assembled by considering public GitHub repositories with more than 1,000 stars that contain C code, use the Pthreads lock API, have less than 500 kB of C code, and can be translated by the C2Rust tool — where programs corresponding to didactic material were excluded. From the 46 programs in this suite, we exclude 5 programs that do not contain a `main` function as our approach does not target libraries, and 5 further programs that do not contain any calls to `pthread_create`, and are thus statically known to be single-threaded. For `dnspod-sr`, `Mirai-Source-Code`, `sysbench`, and `pigz` we use a modified version that fixes some issues with how the original programs were merged to obtain a single C file. We denote this by appending `-fixed` to the program name. This brings the total number of benchmarks in this set to 36.

Tables 5.1 and 5.2 provide a short description of each benchmark program, alongside the number of physical lines of code, as well as the number of logical lines of code (LLoC) counting only lines with executable code — thus excluding, e.g., struct and extern function declarations.

5.2 Evaluation of the Analyses Considering Globals in Isolation

For the evaluation of the scalability and precision of these non-relational analyses (**(RQ1–3)**), we used the Goblint and Concrat benchmarks described above.

We experimented with the following non-relational analyses:

(**Protection**): Protection-Based Analysis (Section 4.1.1)

(**Miné**): Side-Effecting Formulation of the Analysis by Miné [83, 84] (Section 4.1.3)

(**Lock**): Lock-Centered Reading Analysis (Section 4.1.4)

(**Write**): Write-Centered Reading Analysis (Section 4.1.6)

(**Combined**): Combined Lock- and Write-Centered Analysis (Section 4.1.8)

The analyses are performed context-sensitively with a standard points-to analysis for addresses. For the enabled integer domains, we experiment with two different setups: In the first setup, we use only exclusion/inclusion sets, and in the second setup, we additionally activate an interval domain. For the latter configuration, we add the suffix **-I** to the analysis name.

For all analyses for which we have presented refinement based on ego-lane digests, i.e., all analyses except (**Miné**), we additionally experiment with a configuration where refinement according to thread *id*s as presented in Section 2.8 with $\mathsf{compat}^\sharp_{\mathcal{A}}$ defined in terms of $\mathsf{may\_run}$ as described in Example 10 is enabled. For these configurations, we add the suffix **-TID** to the analysis name. For the configurations using this refinement and the interval domain for numeric values, this results in the combined suffix **-I-TID**.

We used a timeout of 15 min for each run and did not bound the memory usage for the experiment. Peak memory usage did not exceed 14 GB for any of the configurations.

For 9 tasks from the Concrat suite, none of the configurations terminated normally, i.e., without raising an exception and within the time limit. Table 5.3 lists these results. Three cases can be distinguished there:

- For 5 of programs, at least one of the analyses encountered a stack overflow in the solver. In Goblint, this occurs for recursive programs whenever the analysis descends deeply into a recursion, always encountering new contexts. Goblint supports techniques [43] to limit the encountered number of contexts. However, these were not enabled for the experiments in this section, to ensure all programs are analyzed in the same way.
- For `clib`, due to a bug in the compiler frontend, the analysis encounters a situation where abstract values of types `int` and `unsigned long` are to be joined. As such a situation does not arise for well-typed programs, the analysis does not support this operation and crashes.
- For the cases where the timeout was reached, it is unclear whether the analysis would terminate normally given more time or encounter issues such as stack overflows. Coming up with more lightweight configurations for Goblint that allow at least some analyses presented in this thesis to terminate within the time limit is an interesting question, which, however, is orthogonal to comparing the precision and performance of the different approaches presented here.

We omit these 9 tasks from further discussion.

5.2.1 Runtime Comparison

Tables 5.4 and 5.5 show the runtimes of the non-relational analyses in the cases where at least one of the configurations terminated normally within the time limit (**RQ1**). While for most small programs, only a small difference in runtime is observable, this changes for larger programs where a bigger difference is observed.

For the configurations without refinement according to thread *id*s (Table 5.4), the protection-based approaches are the fastest across the board, regardless of whether intervals are additionally enabled or not. Among the analyses that admit refinement according to thread *id*s (Table 5.5), the same also holds true.

Fig. 5.1 shows the runtimes for the various approaches, both without and with intervals plotted against the number of logical lines of code per benchmark. In these graphs, data points belonging to the same benchmark are connected by a dashed line. While one can clearly observe that the protection-based approaches tend to be the fastest among the approaches, with the difference between the other approaches mostly negligible, no clear relationship between program size and runtime becomes apparent.

Fig. 5.2 shows the runtimes of terminating runs plotted against the number of encountered unknowns. Here, a clear trend is observable: there seems to be a linear relationship between the number of encountered unknowns and the runtimes of the respective approaches. This conclusion is also supported by the linear regression for the (**Protection**) configuration, which seems to fit the data well. This relationship hints at the analysis time scaling linearly not with the program size, but with the number of program points in contexts to be analyzed, as the number of flow-insensitive unknowns does not grow as fast as the number of program points in contexts. This also supports the hypothesis that for those larger programs where the analyses fail to terminate within the time limit, the root cause is more likely to be the number of encountered contexts to be analyzed — which is also an issue for single-threaded analyses — and less likely a blowup due to the effects of concurrency.

The approaches enhanced with thread *id*s behave similarly to the ones without it, and we thus forgo giving a detailed description and plots here.

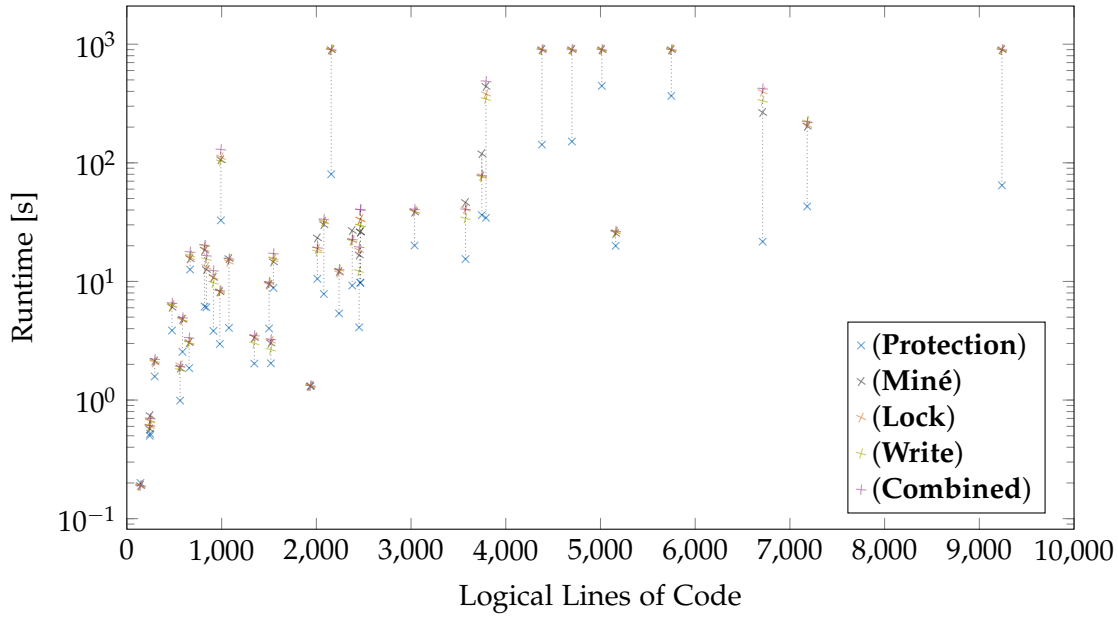
Fig. 5.3 shows the aggregated runtimes for the non-relational analyses for those programs where all approaches terminated. One can see that for all configurations, the protection-based approach offers the shortest runtimes, while the other four (three) approaches are significantly slower on aggregate, but do not exhibit large differences in runtime between each other. This general observation also holds when considering the runtimes on individual tasks (Tables 5.4 and 5.5). This graph also allows shows the overhead of enabling intervals and/or performing a refinement according to thread *id*s.

Table 5.4: Runtimes on the non-relational benchmarks in seconds. Some benchmark names were abbreviated for space (as indicated by ellipses). The fastest runtime for each domain is marked in bold.

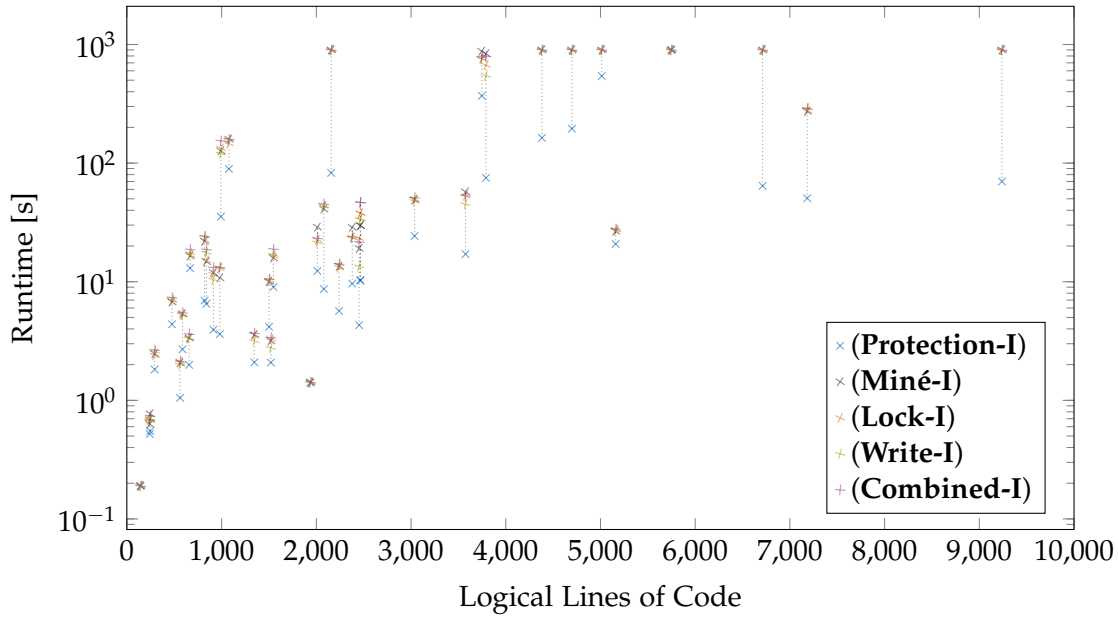
Name	LLoC	(Protection)	(Miné)	(Lock)	(Write)	(Combined)	(Protection-I)	(Miné-I)	(Lock-I)	(Write-I)	(Combined-I)
pfscan	562	**1.0**	1.9	2.0	1.8	1.9	**1.1**	2.1	2.1	2.0	2.1
aget	587	**2.6**	4.8	4.8	4.8	5.0	**2.7**	5.4	5.3	5.3	5.5
ctrace	657	**1.9**	3.1	3.1	3.1	3.4	**2.0**	3.4	3.3	3.3	3.6
knot	981	**3.0**	8.2	8.2	8.3	8.4	**3.6**	10.8	13.0	13.2	13.2
ypbind	992	**32.7**	106.1	111.4	103.1	129.5	**35.4**	127.0	133.3	120.1	154.2
smtprc	3037	**20.1**	38.4	39.2	38.8	40.6	**24.3**	49.0	49.7	48.0	51.1
iowarrior	1345	**2.0**	3.4	3.4	3.0	3.5	**2.1**	3.6	3.6	3.2	3.7
adutux	1520	**2.0**	3.0	3.2	2.7	3.3	**2.1**	3.2	3.3	2.8	3.4
w83977af	1501	**4.0**	9.5	9.4	9.7	9.8	**4.2**	10.1	10.0	10.3	10.5
tegra20	1547	**8.8**	14.6	15.3	15.7	17.2	**9.0**	15.9	16.5	16.9	18.9
nsc	2379	**9.2**	26.9	22.2	22.0	22.6	**9.6**	28.7	23.6	23.4	23.9
marvell1	2465	**9.7**	26.0	33.4	29.5	40.3	**10.2**	29.7	37.8	33.6	46.7
marvell2	2465	**9.8**	26.2	33.3	29.7	40.3	**10.4**	29.8	37.9	33.7	46.6
brubeck	2240	**5.4**	12.1	11.8	12.5	12.6	**5.7**	13.6	13.2	13.9	14.1
C-Thread-Pool	241	**0.5**	0.6	0.6	0.6	0.6	**0.5**	0.6	0.7	0.7	0.7
cava	2011	**10.5**	23.3	18.8	17.9	19.3	**12.3**	28.8	22.6	21.5	23.2
dnspod-sr-fixed	4698	**151.2**	⧖	⧖	⧖	⧖	**194.9**	⧖	⧖	⧖	⧖
dump1090	2079	**7.8**	30.1	31.9	31.8	33.4	**8.7**	40.9	43.5	43.3	45.3
EasyLogger	839	**6.1**	12.6	13.3	15.4	16.5	**6.5**	14.8	15.1	17.6	18.7
fzy	1077	**4.1**	15.6	14.6	15.4	15.6	**89.3**	158.8	146.5	158.9	158.7
klib	293	**1.6**	2.1	2.1	2.1	2.2	**1.8**	2.5	2.4	2.6	2.7
level-ip	2452	**4.1**	16.8	18.3	12.2	19.5	**4.3**	19.1	23.3	13.5	21.6
libaco	667	**12.6**	15.5	15.9	16.5	17.7	**13.0**	16.5	16.8	17.4	18.7
libfaketime	143	0.2	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**
libfreenect	245	**0.5**	0.7	0.7	0.7	0.7	**0.6**	0.8	0.7	0.7	0.7
lmdb	5748	**366.5**	⧖	⧖	⧖	⧖	≡	≡	≡	≡	≡
Mirai-...-fixed	820	**6.1**	18.3	18.7	20.1	20.2	**6.9**	21.9	22.4	24.0	24.2
nnn	6712	**21.6**	265.6	397.4	333.3	422.1	**64.3**	⧖	⧖	⧖	⧖
pianobar	4382	**142.3**	⧖	⧖	⧖	⧖	**163.0**	⧖	⧖	⧖	⧖
pigz-fixed	5014	**445.1**	⧖	⧖	⧖	⧖	**541.8**	⧖	⧖	⧖	⧖
pingfs	913	**3.8**	10.8	10.9	9.8	12.3	**3.9**	11.9	11.6	10.4	13.2
ProcDump-...	2157	**80.2**	⧖	890.5	⧖	⧖	**82.6**	⧖	⧖	⧖	⧖
shairport	3791	**34.4**	440.9	380.6	344.9	486.9	**75.0**	835.1	669.2	546.2	798.6
siege	9239	**64.5**	⧖	⧖	⧖	⧖	**69.9**	⧖	⧖	⧖	⧖
snoopy	1938	**1.3**	**1.3**	1.3	1.3	1.3	**1.4**	1.4	1.4	**1.4**	1.4
sysbench-fixed	3575	**15.4**	46.3	40.9	34.1	40.5	**17.1**	57.2	53.0	45.2	53.2
uthash	476	**3.9**	6.1	6.3	6.3	6.6	**4.4**	6.8	7.0	7.0	7.4
vanitygen	5160	**20.0**	25.4	26.2	26.1	26.5	**20.8**	26.7	27.5	27.3	27.8
wrk	3747	**36.3**	119.3	77.6	76.1	78.3	**368.8**	878.7	759.4	754.7	785.8
zmap	7183	**42.9**	203.4	207.5	223.6	221.3	**50.5**	273.3	276.8	288.5	288.6

Table 5.5: Runtimes on the non-relational benchmarks enhanced with thread *id*s in seconds. Some benchmark names were abbreviated for space (as indicated by ellipses). The fastest runtime for each domain is indicated in bold.

Name	LLoC	(Protection-TID)	(Lock-TID)	(Write-TID)	(Combined-TID)	(Protection-I-TID)	(Lock-I-TID)	(Write-I-TID)	(Combined-I-TID)
pfscan	562	**1.1**	2.6	2.0	2.3	**1.2**	2.8	2.6	3.0
aget	587	**3.2**	5.9	5.7	6.1	**3.4**	6.6	6.4	6.9
ctrace	657	**2.7**	6.2	5.3	6.7	**2.9**	7.2	5.9	7.7
knot	981	**3.2**	8.5	8.6	8.7	**3.9**	11.2	11.3	11.4
ypbind	992	**120.8**	⧖	494.3	⧖	**146.4**	⧖	637.9	⧖
smtprc	3037	**22.6**	52.2	48.7	54.4	**27.1**	64.8	59.5	66.7
iowarrior	1345	**2.1**	3.4	3.1	3.5	**2.1**	3.6	3.2	3.7
adutux	1520	**2.1**	3.2	2.7	3.3	**2.1**	3.3	2.8	3.4
w83977af	1501	**5.5**	11.5	12.0	12.1	**5.8**	12.3	12.7	13.0
tegra20	1547	**11.6**	20.6	21.1	23.8	**12.2**	23.0	23.9	26.9
nsc	2379	**12.8**	25.5	24.4	25.2	**13.4**	27.1	26.1	26.9
marvell1	2465	**12.4**	53.8	33.3	48.9	**13.5**	64.4	38.3	57.9
marvell2	2465	**12.5**	54.5	33.4	49.1	**13.7**	65.2	38.7	57.7
brubeck	2240	**6.2**	13.3	14.1	14.2	**6.5**	14.8	15.7	15.7
C-Thread-Pool	241	**0.6**	0.7	0.7	0.7	**0.6**	0.8	0.8	0.8
cava	2011	**10.9**	19.5	18.3	20.2	**12.8**	23.8	22.3	24.5
dnspod-sr-fixed	4698	**163.6**	⧖	⧖	⧖	**204.3**	⧖	⧖	⧖
dump1090	2079	**12.3**	39.8	38.6	40.3	**14.1**	53.6	52.1	54.2
EasyLogger	839	**6.4**	15.9	16.7	18.1	**6.9**	18.2	19.2	20.8
fzy	1077	**4.3**	14.9	15.8	15.9	**115.5**	153.9	170.9	171.4
klib	293	**1.8**	2.4	2.4	2.6	**2.1**	2.8	2.8	3.0
level-ip	2452	**4.5**	21.7	12.8	22.6	**4.6**	27.5	14.3	25.0
libaco	667	**29.7**	136.4	43.9	144.4	**33.1**	162.4	50.7	171.4
libfaketime	143	**0.2**	**0.2**	**0.2**	**0.2**	0.2	**0.2**	**0.2**	**0.2**
libfreenect	245	**0.7**	0.9	0.9	1.0	**0.7**	1.0	1.0	1.1
lmdb	5748	**372.3**	⧖	⧖	⧖	≡	≡	≡	≡
Mirai-...-fixed	820	**6.9**	19.8	21.2	21.4	**8.2**	25.2	26.7	27.1
nnn	6712	**23.6**	454.0	343.3	438.5	**81.9**	⧖	⧖	⧖
pianobar	4382	**209.3**	⧖	⧖	⧖	**239.5**	⧖	⧖	⧖
pigz-fixed	5014	**814.7**	⧖	⧖	⧖	⧖	⧖	⧖	⧖
pingfs	913	**6.0**	43.1	24.4	55.9	**6.4**	49.0	27.0	62.9
ProcDump-...	2157	**849.1**	⧖	⧖	⧖	⧖	⧖	⧖	⧖
shairport	3791	**46.9**	⧖	821.9	⧖	**100.6**	⧖	⧖	⧖
siege	9239	**106.2**	⧖	⧖	⧖	**116.3**	⧖	⧖	⧖
snoopy	1938	1.3	1.4	**1.3**	1.4	**1.4**	1.5	1.4	1.4
sysbench-fixed	3575	**19.0**	145.8	47.9	81.5	**22.1**	203.6	57.9	100.9
uthash	476	**4.2**	9.0	7.7	8.9	**4.8**	10.2	8.7	10.2
vanitygen	5160	**19.9**	26.7	26.5	27.1	**20.9**	28.1	27.7	28.5
wrk	3747	**37.0**	78.2	76.3	78.7	**383.3**	785.2	741.3	782.3
zmap	7183	**47.2**	213.3	229.6	228.4	**55.5**	287.9	297.1	296.6



(a) Base configurations.



(b) Interval configurations.

Figure 5.1: Runtimes per benchmark program, where a runtime of 900s corresponds to a timeout or other non-normal termination and the runtimes for each benchmark are connected by a dotted vertical line.

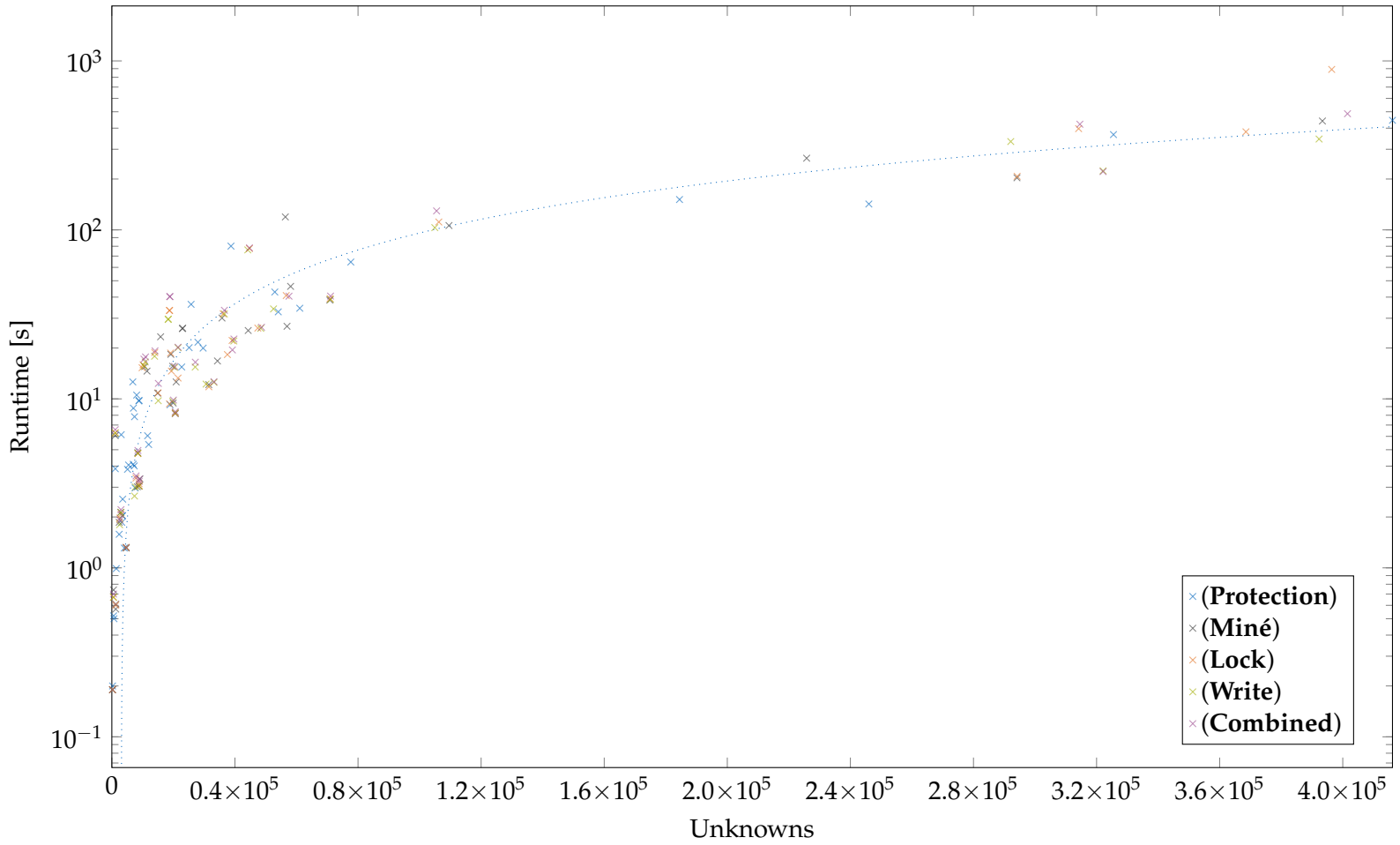


Figure 5.2: Runtimes of the basic approaches plotted against the number of encountered unknowns. The dotted line represents a linear regression for the (**Protection**) configuration.

5.2.2 Identified Thread *Id*s

Before studying differences in precision between the approaches, we first turn to the question whether the digests introduced in Section 2.8 succeed at identifying thread *id*s in the benchmark programs and how many of these thread *id*s can be shown to be unique (**(RQ2)**). While, in principle, the encountered thread *id*s depend on the analysis considered, as, e.g., a more precise analysis may find sections of code where threads are created to be unreachable, or a weaker points-to analysis may lead to more threads being identified, no such effects were observed for the benchmarks considered here. The number of thread *id*s and how many of those are known to be unique per benchmark program is given in Table 5.6.

The number of identified thread *id*s ranges between 2 for `EasyLogger` and 39 for `wrk`, with an outlier of 107 for `ypbind`. We remark that the very high number of threads for this program seems to stem from a call to `pthread_create` where only very imprecise points-to information for the pointer argument is computed, leading to many threads potentially being created. For 5 benchmark programs, all identified thread *id*s are unique as indicated by the bold numbers in Table 5.6. For these benchmarks, the number of encountered thread *id*s is an upper bound to the number of *concrete* threads created. In general, the high fraction of thread *id*s identified as unique here is encouraging.

5.2.3 Precision Comparison

To answer **(RQ3)**, the results computed by the different analyses need to be compared. However, the analyses have different unknowns and track different auxiliary information, making a direct comparison based on the computed solutions to the respective constraint systems impossible. Instead, we record and compare the observable behavior in the form of which abstract values are read for what global variables at which locations. We exclude reads that happen via pointers where the points-to-set the analysis reads from does contain the $\top$ element, which represents a pointer for which a catastrophic loss of precision has occurred.

Base Configurations. We first consider the precision differences in the base setting without intervals and thread *id*s as also presented in [107], only on the extended set of benchmarks here. For 5 of the programs, only (**Protection**) terminated. For `ProcDump-for-Linux`, only (**Protection**) and (**Lock**) terminated. For this program, both terminating approaches yielded the same precision. For the remaining 34 programs, all configurations terminated. For these, the results mostly echo the ones in the earlier evaluation:

- For 24 of the programs, all approaches were equally precise.
- For 6 of the remaining programs, all analyses were equally precise except (**Miné**), which was less precise for between 9.6% and 0.7% of values read.

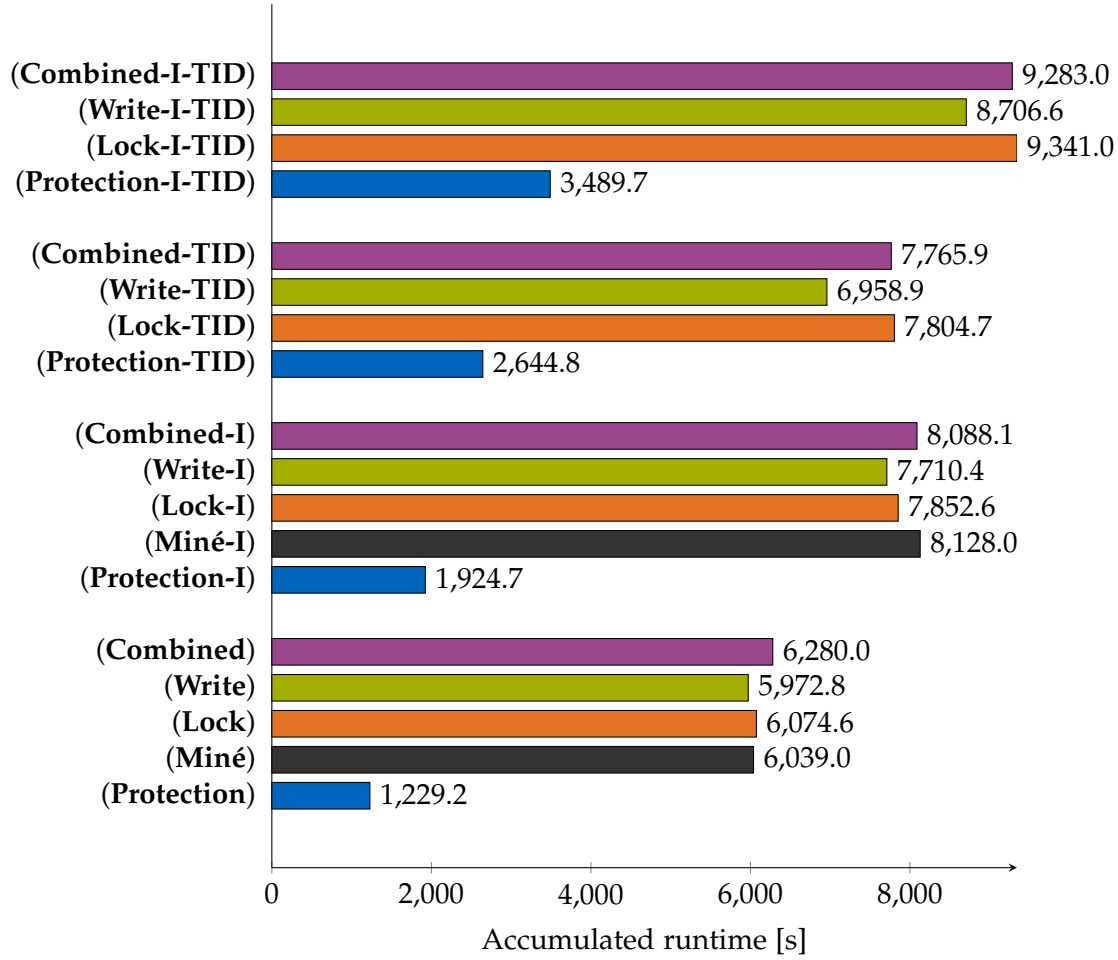


Figure 5.3: Accumulated runtime for the non-relational analyses where all approaches terminated. This subset of the complete suite comprises around 106k LLoC.

Table 5.6: Number of (unique) thread *id*s identified in the benchmark programs. Some benchmark names were abbreviated for space (as indicated by ellipses). Whenever all identified thread *id*s are unique, the numbers are given in bold.

Name	LLoC	#TIDs	#Unique	Name	LLoC	#TIDs	#Unique
`pfscan`	562	3	2	`klib`	293	6	3
`aget`	587	6	4	`level-ip`	2452	9	7
`ctrace`	657	**3**	**3**	`libaco`	667	35	13
`knot`	981	9	5	`libfaketime`	143	4	3
`ypbind`	992	107	41	`libfreenect`	245	14	8
`smtprc`	3037	4	3	`lmdb`	5748	3	2
`iowarrior`	1345	**4**	**4**	`Mirai-...-fixed`	820	6	4
`adutux`	1520	**4**	**4**	`nnn`	6712	23	12
`w83977af`	1501	6	4	`pianobar`	4382	26	13
`tegra20`	1547	7	5	`pigz-fixed`	5014	14	7
`nsc`	2379	11	7	`pingfs`	913	35	18
`marvell1`	2465	6	5	`ProcDump-...`	2157	24	16
`marvell2`	2465	6	5	`shairport`	3791	21	10
`brubeck`	2240	32	16	`siege`	9239	5	4
`C-Thread-Pool`	241	6	3	`snoopy`	1938	7	4
`cava`	2011	20	11	`sysbench-fixed`	3575	14	9
`dnspod-sr-fixed`	4698	9	6	`uthash`	476	**5**	**5**
`dump1090`	2079	9	5	`vanitygen`	5160	33	17
`EasyLogger`	839	**2**	**2**	`wrk`	3747	39	18
`fzy`	1077	8	4	`zmap`	7183	11	7

Table 5.7: Precision comparison for the programs `shairport` and `zmap`. The table shows the comparison between the results from the analysis corresponding to the row and the one corresponding to the column. Incomparable results are denoted by $\not\lesseqgtr$. The same relationship was observed for both programs.

	(Protection)	**(Miné)**	**(Lock)**	**(Write)**	**(Combined)**
(Protection)		$\not\lesseqgtr$	$\sqsupset$	$\sqsupset$	$\sqsupset$
(Miné)	$\not\lesseqgtr$		$\sqsupset$	$\sqsupset$	$\sqsupset$
(Lock)	$\sqsubset$	$\sqsubset$		$=$	$=$
(Write)	$\sqsubset$	$\sqsubset$	$=$		$=$
(Combined)	$\sqsubset$	$\sqsubset$	$=$	$=$	

- For 2 programs, all analyses were equally precise except (**Protection**), which was less precise for 54.4%, respectively 4.3% of values read.

- For the remaining 2 programs `shairport` and `zmap`, the relationship between the precision of the different approaches is depicted in Table 5.7. Here, approaches (**Lock**), (**Write**), and (**Combined**) are all equally precise, and more precise than either (**Protection**) (for 1.2%, resp. 0.2% of values read), or (**Miné**) (for 0.1%, resp. 0.4% of values read). The precisions of (**Protection**) and (**Miné**) are incomparable.

Table 5.8 provides an overview of the results of the precision comparison for the base configuration. It lists, for each approach, for how many programs each analysis yielded the *most* precise results. Considering the drastic differences in runtime (and that only (**Protection**) terminated for 5 of the programs), this seems to hint at (**Protection**) being the most reasonable compromise for the analysis of the programs considered here, despite it being less precise than (**Write**), (**Lock**), and (**Combined**) for 4 of the benchmarks from the suite.

TID Configurations. For these analyses, additionally enabling refinement according to thread *id*s seems to not increase precision as measured by the read values at program locations much. The analyses read the same values as in their base configuration on all but three programs.

For `dnspod-sr-fixed`, only the approaches based on (**Protection**) did terminate within the resource limit. In this case, thread *id*s yielded a precision increase for 2.2% of values read. For `zmap`, the **-TID** configurations are all better than their base configurations: (**Write-TID**), (**Lock-TID**), and (**Combined-TID**) achieve the same precision, which is better than the precision of (**Write**), (**Lock**), and (**Combined**) for 0.2% of values read. (**Protection-TID**) is also more precise than (**Protection**) for 0.2% of values read. Comparing within the **-TID** variants, (**Protection-TID**) is less precise than the other approaches once more for 0.2% of values read.

Given the extra cost incurred by refining according to thread *id*s, it seems that, for these benchmarks, the modest precision increase obtained is not worth the cost.

Table 5.8: Number of programs for which each approach yielded the most precise result for the 34 programs on which all approaches terminated. The (+1) for (**Lock**) indicates the program, where only (**Lock**) and (**Protection**) terminated. Analogously for the (+1+5) for (**Protection**), where (+5) denotes the additional 5 programs where only it terminated.

(**Protection**)	(**Miné**)	(**Lock**)	(**Write**)	(**Combined**)
30 (+1+5)	26	**34** (+1)	**34**	**34**

```c
#include <pthread.h>
int occupied;
pthread_mutex_t m;

void* thread(void* arg) {
  pthread_mutex_lock(&m);
  if(occupied < 1) {
    occupied++;
    pthread_mutex_unlock(&m); // (1)
  } else {
    pthread_mutex_unlock(&m);
  }
}

int main() {
  pthread_t t;

  pthread_create(&t, 0, &thread, 0);

  pthread_mutex_lock(&m);
  occupied = 0;
  pthread_mutex_unlock(&m); // (2)

  pthread_mutex_lock(&m);
  __goblint_check(occupied >= 0);
  pthread_mutex_lock(&m);
  return 0;
}
```

Figure 5.4: Example of a pthread program highlighting the effects of widening extracted from pfscan. The assertion succeeds with the Protection-Based Analysis but fails with Write-Centered Reading. As this is an example extracted from C code, we do not employ our toy language, but turn to C to describe this program here.

Interval Configurations. When considering the analysis with the interval domain additionally enabled, it is important to note that the domain used for globals requires widening now. This may lead to the analyses behaving differently w.r.t. precision than in the base configuration, where the relationships between the individual results mostly correspond to the results one would obtain for *least* solutions.

Example 33. *Consider the program in Figure 5.4, which is an extracted snippet from the pfscan benchmark. Notice that the variable `occupied` is protected by the mutex `m` and ranges between `0` and `1` in the concrete.*
When additionally using intervals and the usual widening on them, the assertion succeeds with the Protection-Based Analysis but fails with the Write-Centered Reading analysis. This effect is due to widening in combination with the enabled domains: Instead of considering the combination of inclusion/exclusion sets and intervals, we illustrate the problem with an easier-to-follow combination of constant propagation and intervals. Let us denote abstract values in this domain as a pair where the first element of the pair is the constant (or $\top$), and the second is the interval.

- Goblint *applies widening for all contributions to a global unknown as soon as more than one increasing contribution has happened from the same program point.*
- *The enabled solver will, in this example, after performing any side-effects corresponding to thread creation, attempt to compute a fixpoint for the unknown corresponding to the return of the thread whose start point just received a side-effect.*

- *For Write-Centered Reading, the contributions to* $[occupied, m, \emptyset, \{m\}]$ *from unlock of* m *in the true branch in* `thread` *(annotated with (1) in the program) are* $(1, [1,1])$ *and in a later iteration* $(\top, [1,1])$ *(as no reduced product between the domains is applied, and the constants cannot exploit the guard to refine the read value* $(\top, [0,1])$*) which leads to this unknown becoming a widening point with the value* $(\top, [1,1])$*. When the contribution from (2), i.e.,* $(0, [0,0])$*, is considered, widening is applied, and the value of the unknown becomes* $(\top, [1,1]) \nabla (0, [0,0]) = (\top, [minint, 1])$*.*
- *For Protection-Based Reading, the unknown* $[occupied]$ *receives an initial contribution of* $(0, [0,0])$ *when* Goblint *first enters multi-threaded mode, and then later the contribution* $(1, [1,1])$*. These are joined and the value* $(\top, [0,1])$ *is attained. While the unknown may later also become a widening point, all the contributions are accounted for by the result of this join, and no widening takes place.*

For this example, there exist various ways to obtain the desired precision also with Write-Centered Reading, including disabling the inclusion/exclusion sets, enabling reduced product, or choosing a different widening strategy. However, all of these also have drawbacks. □

Example 33 thus illustrates the complex interplay between different options and widening and highlights that it is not unexpected that the relationships between least solutions, on the one hand, and the empirically observed precision, on the other hand, may not always align.

Out of the 40 programs for which at least one configuration terminated without intervals, this is still the case for 39 programs when enabling intervals: For `lmdb`, none of the approaches additionally using intervals terminated. For 6 programs, only (**Protection-I**) terminated.

For the remaining 33 programs, all configurations terminated. The breakdown of the precisions is as follows:

- For 16 of the programs, all approaches were equally precise.
- For 4 programs, all approaches were equally precise except for (**Protection-I**), which was *more* precise for between 0.7% (`cava`) and 28.3% (`klib`) of values read.
- For 3 programs, all approaches were equally precise except for (**Miné-I**), which was *less* precise for between 0.7% (`vanitygen`) and 28.6% (`snoopy`) of values read.
- For `fzy`, all approaches were equally precise except for (**Protection-I**), which was *less* precise for 54.4% of values read.
- For `dump1090`, all approaches were equally precise except for (**Protection-I**) which yielded precision incomparable to the others.

For the remaining 8 programs, the relationships between the precisions are more involved and are highlighted in Table 5.9.

Table 5.10 provides an overview of the results of the precision comparison for the interval configurations. It lists, for each approach, for how many programs the analysis

Table 5.9: Precision comparison for the 8 programs where the relationship between precisions is not straightforward. The table shows the comparison between the results from the analysis corresponding to the row and the one corresponding to the column. Incomparable results are denoted by ⋦.

	(Protection-I)	(Miné-I)	(Lock-I)	(Write-I)	(Combined-I)	(Protection-I)	(Miné-I)	(Lock-I)	(Write-I)	(Combined-I)	(Protection-I)	(Miné-I)	(Lock-I)	(Write-I)	(Combined-I)
	`pfscan`					`knot`					`ypbind`				
(Protection-I)		⊏	⊏	⊏	⊏		⊏	⊏	⊏	⊏		⊏	⊏	=	=
(Miné-I)	⊐		⊐	⋦	⋦	⊐		⋦	⋦	⋦	⊐		⋦	⊐	⊐
(Lock-I)	⊐	⊏		⊏	⊏	⊐	⋦		=	=	⊐	⋦		⊐	⊐
(Write-I)	⊐	⋦	⊐		=	⊐	⋦	=		=	=	⊏	⊏		=
(Combined-I)	⊐	⋦	⊐	=		⊐	⋦	=	=		=	⊏	⊏	=	
	`shairport`					`sysbench-fixed`					`zmap`				
(Protection-I)		⋦	⋦	⋦	⋦		⊏	⊏	⊏	⊏		⋦	⊐	⊐	⊐
(Miné-I)	⋦		⊐	⋦	⋦	⊐		⋦	⊐	⊐	⋦		⊐	⊐	⊐
(Lock-I)	⋦	⊏		⊏	⊏	⊐	⋦		⊐	⊐	⊏	⊏		=	=
(Write-I)	⋦	⋦	⊐		=	⊐	⊏	⊏		=	⊏	⊏	=		=
(Combined-I)	⋦	⋦	⊐	=		⊐	⊏	⊏	=		⊏	⊏	=	=	
	`wrk`					`level-ip`									
(Protection-I)		⊏	⊏	=	=		⊏	⋦	⊏	⊏					
(Miné-I)	⊐		=	⊐	⊐	⊐		⊐	=	=					
(Lock-I)	⊐	=		⊐	⊐	⋦	⊏		⊏	⊏					
(Write-I)	=	⊏	⊏		=	⊐	=	⊐		=					
(Combined-I)	=	⊏	⊏	=		⊐	=	⊐	=						

yielded the *most* precise results, as well as (after the slash) the number of results where no other approaches were better, but some yielded incomparable results.

Considering the drastic differences in runtime (and that only (**Protection-I**) terminated for 6 of the programs), this seems to hint at (**Protection-I**) being the most reasonable choice for the analysis of the programs considered here when analyzed with intervals enabled.

To evaluate the impact of additionally enabling the interval domain, we compare the results between the base configurations and the configurations with intervals by projecting the results of the base configuration to the interval domain while using the range information encoded in the exclusion set domain. The observed results were relatively uniform across the benchmarks: For 39 benchmarks, at least one base configuration and one configuration with intervals terminated. For 30 of these benchmarks, all interval configurations were more precise than their respective base configuration. The percentage of values that became more precise ranged between 1.6% to 87.0% with on average around 30% of values read becoming more precise. The median improvement was between 24.3% and 30.7%. For the remaining 9 programs, none of the analyses got more precise by additionally enabling the interval domain. All in all, this confirms the conventional wisdom that an analysis using intervals also gives precision advantages in practice.

Interval-TID Configurations. As previously for the configurations without intervals, it is of interest to study the impact of additionally enabling the refinement according to thread *id*s. Out of a total of 37 programs, for which at least one configuration with intervals and thread *id*s terminated, for 27 programs, the precision for all analyses did not change when additionally enabling thread *id*s. For the remaining 10 programs, the results were more varied:

- For `knot`, all configurations with intervals and thread *id*s yielded identical precision. That amounts to the (**Protection-I-TID**) configuration losing some precision when compared to the (**Protection-I**) configuration (for 12.6% of values read). For the other analyses, the configurations using thread *id*s and intervals yielded incomparable results to the configuration using intervals only (10.8% of values read more precise, 12.6% less precise).

Table 5.10: Number of programs for which each approach yielded the most precise result (and for how many no other approaches were better, but overall results were incomparable listed after the slash) for the 35 programs on which all approaches terminated. The (+6) for (**Protection-I**) indicates the 6 programs, for which only it terminated.

(**Protection-I**)	(**Miné-I**)	(**Lock-I**)	(**Write-I**)	(**Combined-I**)
28/3 (+6)	17/1	21/3	23/1	23/1

- For `ypbind`, the only configurations with intervals and thread *id*s that terminated are (**Protection-I-TID**) and (**Write-I-TID**). Both configurations lost precision when compared to the configurations without thread *id*s where (**Write-I**), (**Protection-I**), and (**Combined-I**) were equally precise. (**Protection-I-TID**) was less precise than these for 7.8% of values read, and (**Write-I-TID**) was less precise for an *additional* 2.2% of values read.

- For `dnspod-sr-fixed`, (**Protection-I-TID**) and (**Protection-I**) which were the only terminating configurations with intervals enabled, yielded results that were incomparable: (**Protection-I-TID**) was more precise for 2.2% of values read, while (**Protection-I**) was more precise for 0.9% of values read.

- For `dump1090`, the analyses using intervals became less precise when enabling thread *id*s: All approaches became less precise for 55.6% of values read.

- For `klib`, (**Write-I-TID**) (**Lock-I-TID**) and (**Combined-I-TID**) were equally precise as the corresponding analyses without thread *id*s. (**Protection-I-TID**) was equally precise as all these, meaning the precision edge for 23.2% of globals read that (**Protection-I**) had over all of these was lost.

- For `libfreenect`, (**Protection-I-TID**) was equally precise as (**Protection-I**). (**Lock-I-TID**) (**Write-I-TID**), and (**Combined-I-TID**) all improved over their respective configurations without thread *id*s, and were then equally precise as (**Protection-I-TID**). That amounts to an improvement for 6.3% of values read.

- For `shairport`, (**Protection-I-TID**), the only configuration with intervals and thread *id*s that terminated, was less precise than (**Protection-I**) for 0.3% of values read.

- For `snoopy`, the configurations additionally using thread *id*s were less precise for 25.0% of values read.

- For `wrk`, the configurations with intervals and thread *id*s yielded equal precision to the configurations without thread *id*s, except for (**Lock-I-TID**), which was less precise than (**Lock-I**) for 0.9% of values read.

- For `zmap`, the configurations with intervals and thread *id*s yielded precisions that were incomparable to the configurations without thread *id*s: They were more precise for 0.2% of values read, and less precise for another 0.2% of values read.

Thus, there is no clear advantage to enabling the refinement according to thread *id*s here. Given that there is some advantage from enabling thread *id*s for the relational analyses (see next section), we suspect that there may be the potential of attaining precision gains by using thread *id*s, but that perhaps different approaches to widening may be needed to realize such improvements.

5.2.4 Summary

For the non-relational analyses, the benchmark results seem to suggest that (**Protection**) is as precise as the other approaches for the majority of benchmarks (**(RQ3)**), and given its runtime is significantly faster than the other approaches (**(RQ1)**), it is the most promising approach for practical use.

For the approaches additionally using intervals, this holds true even more: As explained above, due to the effects of widening and narrowing when (efficiently) computing fixpoints, (**Protection-I**) is not only more scalable than the other approaches on the considered set of benchmarks, but also often more precise.

Despite many encountered thread *id*s and a high fraction of unique thread *id*s (**(RQ2)**), there seems to be little benefit (and even some potential harm) when enabling the refinement according to thread *id*s for the considered benchmark programs. This refinement should thus probably be enabled only when thread *id*s are needed for some other analysis such as an analysis of data races, but not when only an analysis of the values of global variables is to be conducted.

5.3 Evaluation of the Analyses Considering Clusters of Globals

To address **(RQ4–5)**, we experimented with four different configurations of our analyses from Section 4.2:

(**MM-I**): Mutex Meet with Intervals (Section 4.2.1)

(**MM-Oct**): Mutex Meet with Octagons (Section 4.2.1)

(**MM-Oct-TID**): (**MM-Oct**) with thread *id*s and joins (Section 4.2.4)

(**MM-Oct-TID-C**): (**MM-Oct-TID**) with clusters of size at most 2 (Section 4.2.5)

where all configurations except (**MM-I**) employ a relational domain. Further, we include two configurations based on the non-relational Protection-Based analysis from Section 4.1.1, namely, (**Protection-I**) and (**Protection-I-TID**) , which we have observed to be more precise than the other approaches when using intervals.

An experimental comparison with other tools is difficult. We considered comparing with the following abstract-interpretation based tools that can analyze concurrent C programs:

- Duet [45] — Its benchmarks are only available as binary goto-programs, which neither its current version nor any other tool considered here can consume. While we managed to run Duet successfully on some benchmarks, our configuration of the tool did not produce valid results for others: For dealing with code containing function calls, Duet relies on inlining. As the inlining implemented in the most recent version of Duet available at the time of writing was not working on some examples, we contacted the author, who sent us a fixed version of the module

responsible for inlining. In the experiments, we executed the tool with this implementation of inlining (and Octagons enabled); however, there were still cases in which our configuration of the tool reported a too low number of reachable asserts, indicating that some reachable code was not considered by the tool. Thus, for these benchmarks, no results are reported for DUET.

- ASTRÉEA [85] — A public version is available but not licensed for evaluation.
- WATTS [72] — Since we were unable to run the tool on any program, we compared with the numbers reported by the authors where they were available.
- FRAMA-C [70] — to handle concurrency, FRAMA-C relies on the MTHREAD plug-in, which is not publically available.

For **(RQ5)**, we first consider a set of small litmus tests also proposed in literature against which we were able to compare our analyses. For both **(RQ4)** and **(RQ5)**, we used the same set of programs as in the previous section to evaluate performance and precision on more real-world programs.

5.3.1 Litmus Tests

The results for this suite of litmus tests are summarized in Table 5.11. For these runs, a threshold widening [18] was enabled, which derives the considered thresholds from the constants appearing in the program to be analyzed.

In the following, we quickly describe the individual groups of benchmarks and the results.

Ratcop benchmarks [89]. These 19 programs were originally JAVA programs. After manual translation to C, our analyzer succeeded in proving all assertions any configuration of RATCOP could with *Octagons*, while RATCOP required polyhedra in one case. For these quite small benchmarks ($\leq$ 140 logical lines of code (LLoC)), the analysis time was negligible ($\leq$ 1s), but a bit slower than the runtimes reported for the RATCOP tool.

Our benchmarks. To capture particular challenges for multi-threaded relational analysis, we collected a set of small benchmarks (including most examples from this thesis) and instrumented them with assertions. Our relational analysis in the (**MM-Oct-TID-C**) configuration succeeded in verifying all assertions. The other tools and configurations could only prove a handful of relational assertions.

Watts Benchmark Suite. While we were unable to get WATTS to run, we executed our analyses on the benchmarks reported on in [72]. We took the benchmarks as available from the GitHub repository[7]. The benchmarks proposed in this paper consist of two sets of C-source files containing multi-threaded code with asserts.

[7] https://github.com/markus-kusano/watts

Table 5.11: Summary of evaluation results for the litmus tests, with individual programs grouped together. For each group the number of programs and the total number of assertions are given. ✓ (✗) indicates that all (no) assertions are proven, otherwise the number of proven assertions is given. (—) indicates that invalid results were produced.

					Relational					
Set	Group	#	Asserts	(MM-I)	(MM-Oct)	(MM-Oct-TID)	(MM-Oct-TID-C)	(Protection-I)	(Protection-I-TID)	Duet
Our	Basic	3	4	✓	✓	✓	✓	✓	3	3
	Relational	10	35	✗	✓	✓	✓	4	4	2
	TID	12	19	✗	3	✓	✓	✗	✗	2
	Cluster	2	3	✗	✗	1	✓	✗	✗	1
Watts	Created	4	4	2	2	2	2	2	2	✗
	SV-COMP	3	3	✗	✗	✗	✗	✗	✗	✗
	LKMPG	1	2	✗	✗	✗	✗	✗	✗	✗
	DDVerify	29	1072	1044	1044	✓	✓	1044	1044	—
	Scalability	5	740	735	735	✓	✓	735	735	—
Ratcop		19	34	8	14	22	22	6	6	4

Table 5.12: Runtimes, in seconds, of our analyzer on the five scalability benchmarks from [72]. The second column indicates the number of concrete threads in the benchmark program (including the `main` thread). Runtimes are considerably lower than those reported for Watts but were obtained on different hardware.

			Relational				
Benchmark	#Threads	**(MM-I)**	**(MM-Oct)**	**(MM-Oct-TID)**	**(MM-Oct-TID-C)**	**(Protection-I)**	**(Protection-I-TID)**
`i8xx_tco_03_thr01`	31	0.4	0.4	1.5	4.2	0.3	0.3
`i8xx_tco_03_thr02`	41	0.4	0.4	2.4	6.6	0.3	0.3
`i8xx_tco_03_thr03`	51	0.5	0.5	3.5	9.5	0.4	0.4
`i8xx_tco_03_thr04`	61	0.5	0.6	4.8	12.8	0.4	0.4
`i8xx_tco_03_thr05`	71	0.6	0.6	6.5	16.7	0.4	0.5

The first set of benchmarks consists of 37 C-files, originating from different sources, that were adapted for [72]. We took the benchmark set as-is, except for removing an obviously misplaced semicolon in the `wdt977_02` benchmark that rendered one `assert(0)` reachable in the concrete.

The second set of benchmarks consists of five versions of the benchmark `i8xx_tco_03`, contained in the first benchmarks set, instrumented to create different numbers of threads. In [72], the number of threads created in the benchmarks varies from 30 to 70. In the repository, there were two files creating 40 threads and no file that created 30, rendering one test case redundant. Thus, we removed ten thread creates from `i8xx_tco_03_thr01`. We also fixed the number of function stubs in that file to be the same as in the other benchmark files.

The runtimes of our tool can be seen in Table 5.12. Our most expensive analysis takes around 17 seconds to complete on the largest benchmark, which creates 70 threads. While exact runtimes are not reported in [72], the graph (found in Fig. 11 of that paper) indicates that the runtime of their most expensive analysis was close to 400 seconds, while the least expensive configuration still took more than 200 seconds on the benchmark creating 70 threads. We remark that while runtimes reported for Watts in [72] were obtained on a different machine and the numbers are thus not directly comparable, the comparison is still meaningful as the magnitudes differ greatly.

The configuration (**MM-Oct-TID**) succeeded in verifying that all asserts in 36 of these benchmarks hold. The six benchmark files for which, in total, seven asserts could not be proven, contain data-dependent thread-synchronization that our tool cannot handle.

Unlike the other sets of benchmarks considered, there was a significant number of `assert(0)` statements in the benchmark files, where code should be proven to be

Table 5.13: Tasks from the Concrat suite for which none of the approaches from Section 4.2 terminated normally within the 15 min time limit. The symbol ≡ denotes termination due to a stack overflow, whereas 🐞 denotes termination due to an internal error of the analyzer or the compiler frontend, and ⧗ indicates timeout. For programs prefixed with ⋆, there was at least one configuration of the analyses from Section 4.1 that terminated normally within the time limit.

Name	(MM-I)	(MM-Oct)	(MM-Oct-TID)	(MM-Oct-TID-C)
`AirConnect`	≡	≡	≡	⧗
`axel`	⧗	⧗	⧗	⧗
`clib`	🐞	🐞	🐞	🐞
⋆`lmdb`	⧗	⧗	⧗	⧗
`minimap2`	⧗	⧗	⧗	⧗
`phpspy`	⧗	⧗	⧗	⧗

Name	(MM-I)	(MM-Oct)	(MM-Oct-TID)	(MM-Oct-TID-C)
⋆`pigz-fixed`	⧗	⧗	⧗	⧗
`Remotery`	≡	≡	≡	≡
`sshfs`	⧗	⧗	⧗	⧗
`streem`	⧗	⧗	⧗	⧗
`the_silver_searcher`	≡	≡	≡	≡

unreachable. Consequently, for these benchmarks, we include the number of asserts that could be proven unreachable in the number of verified asserts we report.

Altogether, the results on this set of litmus tests show that the analysis in the (**MM-Oct-TID-C**) configuration is able to verify more assertions than any other of the considered configurations and tools, with the (**MM-Oct-TID**) configuration in a close second place. For these programs, taking thread *id*s into account seems to be crucial, as evidenced by the difference between those two configurations and (**MM-Oct**).

5.3.2 Real-World Benchmarks

To address **(RQ4)** and provide additional insights into **(RQ5)**, once more runtimes of the approaches as well as their relative precision were evaluated on the set of real-world benchmarks described in Section 5.1.

Runtime Comparison ((RQ4))

For 11 benchmarks from the Concrat suite, none of the configurations terminated normally within the 15-minute time limit. These benchmarks are listed in Table 5.13. This set includes all benchmarks for which no non-relational configuration terminated normally within the time limit, as well as the programs `lmdb` and `pigz-fixed`.

For the remaining benchmarks, the runtimes are given in Table 5.14. For quick

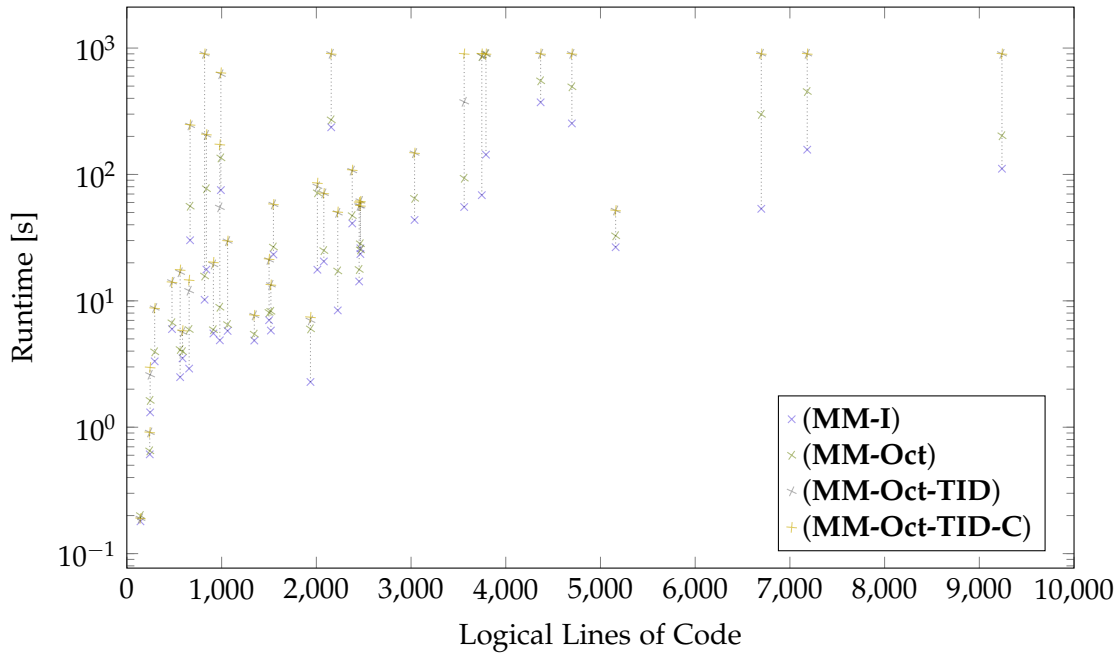


Figure 5.5: Runtimes per benchmark program for the relational analyses, where a runtime of 900s corresponds to a timeout or other non-normal termination and the runtimes for each benchmark are connected by a dotted vertical line.

reference, the table also includes the number of logical lines of code[8] as well as the number of computed thread *id*s and how many of these were found to be unique.

[8]The number of LLoC differs by a few lines between the relational and non-relational analyses due to the preprocessor not including some stub files in the former setting. For the sake of internal consistency, we list the number of LLoC computed by the non-relational analysis (as in Table 5.4) here.

Table 5.14: Runtimes of the approaches from Section 4.2 on the benchmarks in seconds. Some benchmark names were abbreviated for space (as indicated by ellipses).

Name	LLoC	TIDs	Unique TIDs	(MM-I)	(MM-Oct)	(MM-Oct-TID)	(MM-Oct-TID-C)
pfscan	562	3	2	2.5	4.1	16.8	17.6
aget	587	6	4	3.5	4.0	5.6	5.8
ctrace	657	3	3	2.9	5.9	12.0	14.6
knot	981	9	5	4.9	8.9	54.9	172.4
ypbind	992	107	41	75.2	135.2	620.7	638.1
smtprc	3037	4	3	43.7	64.7	147.5	147.7
iowarrior	1345	4	4	4.9	5.5	7.7	7.7
adutux	1520	4	4	5.8	8.3	13.4	13.4
w83977af	1501	6	4	7.0	8.1	21.1	21.4
tegra20	1547	7	5	23.3	26.7	57.9	58.1
nsc	2379	11	7	41.0	47.2	107.3	108.3
marvell1	2465	6	5	23.5	25.3	56.4	56.5
marvell2	2465	6	5	26.1	28.3	61.7	61.4
brubeck	2240	32	16	8.4	17.2	49.6	50.5
C-Thread-Pool	241	6	3	0.6	0.7	0.9	0.9
cava	2011	20	11	17.6	71.6	80.6	85.5
dnspod-sr-fixed	4698	9	6	253.3	493.8	⧗	⧗
dump1090	2079	9	5	20.5	25.0	70.8	70.9
EasyLogger	839	2	2	17.7	77.1	205.4	207.1
fzy	1077	8	4	5.8	6.5	29.7	29.8
klib	293	6	3	3.3	4.0	8.8	8.7
level-ip	2452	9	7	14.2	17.7	56.4	60.0
libaco	667	35	13	30.2	56.1	244.4	247.1
libfaketime	143	4	3	0.2	0.2	0.2	0.2
libfreenect	245	14	8	1.3	1.6	2.6	3.0
Mirai-...-fixed	820	6	4	10.2	15.7	🐞	🐞
nnn	6712	23	12	53.5	298.9	⧗	⧗
pianobar	4382	26	13	372.1	550.6	⧗	⧗
pingfs	913	35	18	5.5	5.9	19.4	20.2
ProcDump-...	2157	24	16	236.0	270.5	⧗	⧗
shairport	3791	21	10	143.4	⧗	⧗	⧗
siege	9239	5	4	111.0	202.5	⧗	⧗
snoopy	1938	7	4	2.3	6.0	7.0	7.4
sysbench-fixed	3575	14	9	55.3	93.4	375.6	⧗
uthash	476	5	5	6.0	6.7	14.1	14.0
vanitygen	5160	33	17	26.6	32.7	51.9	51.9
wrk	3747	39	18	68.6	856.9	868.1	⧗
zmap	7183	11	7	157.0	452.9	⧗	⧗

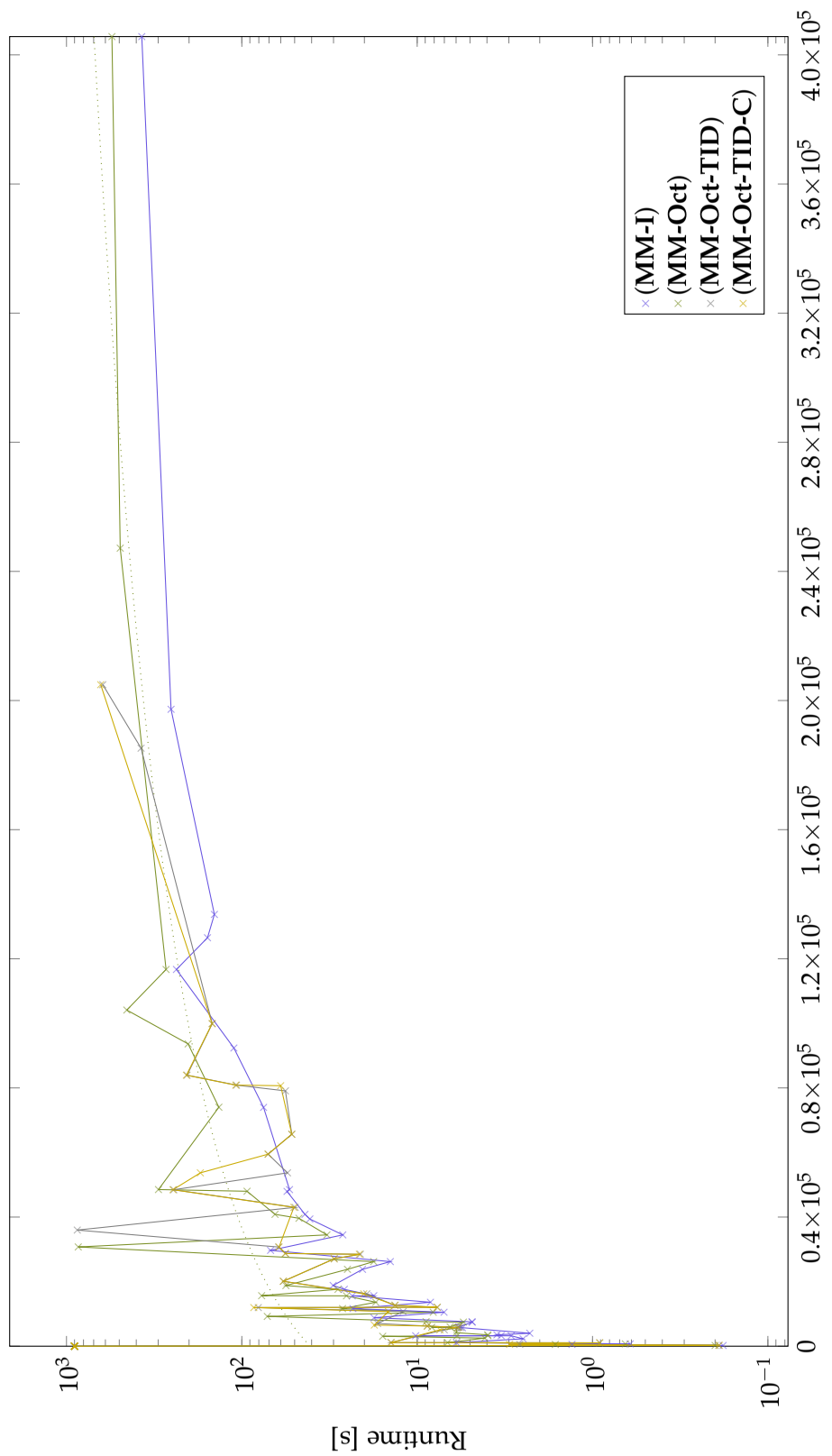


Figure 5.6: Runtimes of the approaches considering cluster of globals plotted against the number of encountered unknowns. The dotted line represents a linear regression for the (**MM-Oct**) configuration.

As expected, going from the non-relational configuration (**MM-I**) to the relational configuration (**MM-Oct**) using the octagon domain incurs a slowdown. Ignoring one outlier where the slowdown is 1149% (`wrk`) and the program `shairport` where (**MM-Oct**) times out, the average slowdown is 86.6%, and the median slowdown is 41.9%.

Enabling refinement according to thread *id*s also incurs considerable overhead: six additional programs time out. Among those programs where both approaches terminated normally, the slowdown ranges between -5.0% and 517.0% with an average slowdown of 153.4% and a median slowdown of 122.4%.

When comparing (**MM-Oct-TID**) to (**MM-Oct-TID-C**) on the other hand, the observed slowdown is usually negligible (between -0.8% and 6.4%) with four outliers where slowdowns of 14.2% (`libfreenect`), 21.3% (`ctrace`), $\geq$ 139.7% (`sysbench-fixed`), and 214.0% (`knot`) were observed. Excluding those four outliers, the average slowdown is 1.5%, and the median slowdown is 0.4%.

These results complement the earlier preliminary experiments [108] performed on a subset of these benchmarks, where smaller runtime differences were observed.

Fig. 5.5 plots the runtimes of the different approaches against the number of logical lines of code. Data points belonging to the same program are connected by a dashed line. While there is once more no clear trend of how runtime changes as the number of lines of code increases, Fig. 5.6 shows that, also here, the runtime scales linearly with the number of encountered unknowns.

The bottom four bars of Fig. 5.7 show the accumulated runtime for those programs where all configurations terminated normally.

Fig. 5.7 as a whole compares the runtimes of all the approaches described in this thesis, providing an insight into **(RQ1)** and **(RQ4)**.

Precision Comparison ((RQ5))

The Goblint and Concrat benchmark sets do not come pre-equipped with assertions. We therefore evaluated the precision of our analyses on these benchmarks in two ways:

First, we performed an internal comparison, comparing abstract values at each program point (joined over all contexts). For 37 programs at least two different approaches terminated — including (**MM-I**) and (**MM-Oct**) in all cases. For 31 of these programs, the configuration (**MM-Oct**) was more precise than (**MM-I**), whereas for `dump1090`, `cava`, `pianobar`, `siege`, `vanitygen`, and `zmap` the results were incomparable. For the benchmarks where there was an improvement, it ranged between 4.3% (for `w83977af`) and 95.7% (for `snoopy`) with an average improvement of 26.0%, and a median improvement of 19.2%. For the benchmarks where the results were incomparable, when considering individual program points, (**MM-Oct**) was more precise for between 18.0% and 47.1% of them. For `cava`, `pianobar`, and `vanitygen`, there were program points where (**MM-I**) was more precise (at most 1.6% of total program points); for all other benchmarks, there were no such program points. For all benchmarks where overall precision was incomparable, there also were individual program points where the results were incomparable (between 0.1% and 33.9% of total program points). Altogether, this comparison between

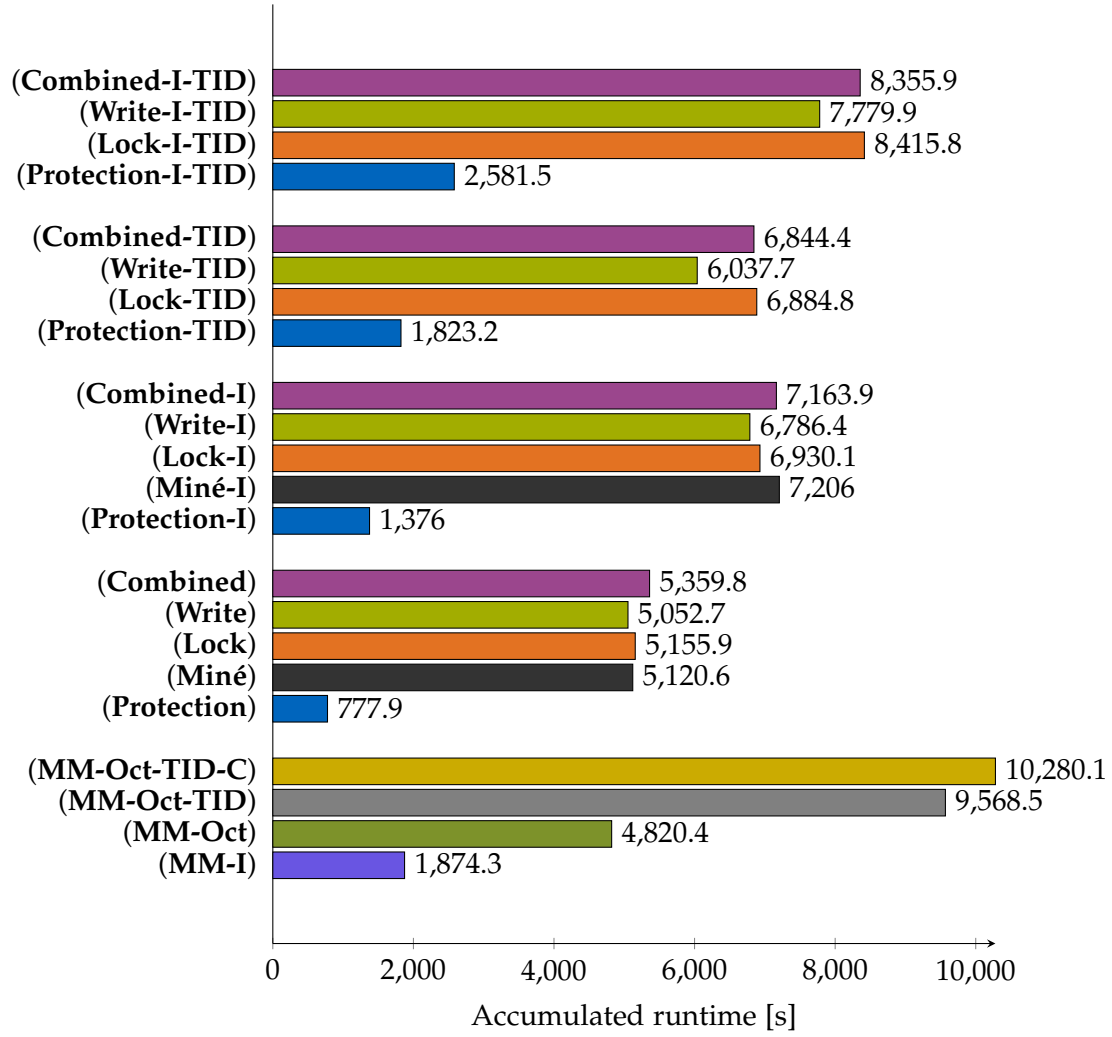


Figure 5.7: Accumulated runtime for all approaches presented in this thesis provide all approaches terminated on the given benchmark. This subset of the complete suite comprises around 86k LLoC.

(**MM-I**) and (**MM-Oct**) is an indication that the analyses manage to establish meaningful relational invariants, that go beyond relationships that are implied by interval information.

For 7 programs, only (**MM-I**) and (**MM-Oct**) terminated normally. For the remaining programs, the precision relationships were as follows:

- For 14 programs, (**MM-Oct**) was more precise than (**MM-I**), and there was no precision difference between (**MM-Oct**), (**MM-Oct-TID**), and (**MM-Oct-TID-C**).
- For 9 programs, (**MM-Oct-TID**) was more precise than (**MM-Oct**). Here, the fraction of more precise unknowns ranged between 0.1% (for `sysbench-fixed`) and 25.7% (for `libfreenect`). The average improvement was 4.8%, and the median improvement was 0.7%. For 8 of those programs, there was no precision difference between (**MM-Oct-TID**) and (**MM-Oct-TID-C**). For `sysbench-fixed`, (**MM-Oct-TID-C**) did not terminate normally within 15 min.
- For `dump1090` and `vanitygen`, the results for (**MM-Oct**) and (**MM-I**) were incomparable, as discussed above. The results for (**MM-Oct**), (**MM-Oct-TID**), and (**MM-Oct-TID-C**) did coincide for these programs.
- For `wrk`, (**MM-I**) was less precise than (**MM-Oct**) which was as precise as (**MM-Oct-TID**). The configuration (**MM-Oct-TID-C**) did not terminate normally within 15 min for this program.
- For four programs, the relationship between the different results is more intricate. The precision comparisons for these cases are highlighted in Table 5.16. Of particular interest is the program `fzy` where enabling thread *id*s in the analysis leads to a *decrease* in precision for 0.4% of program points. For `ypbind`, enabling thread *id*s decreases the precision for 0.1% of program points, and, on the other hand improves precision for 0.2% of program points. These two cases are the only cases where the effect of enabling thread *id*s is not purely beneficial from a precision perspective. `level-ip` is the only program from this set where a precision difference between (**MM-Oct-TID**) and (**MM-Oct-TID-C**) can be observed. Here, the clusters help increase precision for 1.4% of program points.

A summary of which approach provided the best result for how many programs is provided in Table 5.15.

To be able to also relate the precision of the approaches from Section 4.1 with the approaches from Section 4.2, we automatically instrumented all benchmarks for which the (**MM-Oct-TID-C**) configuration terminated normally with the assertions this approach can establish at program points succeeding locking of a mutex. In this way, 29 annotated programs were obtained. Each of these contains between 0 and about 2000 assertions. On top of all the configurations from Section 4.2, we once again experimented with (**Protection-I**) and (**Protection-I-TID**). As DUET already failed to produce valid results for most of the litmus tests, we did not include it in this comparison.

These evaluation results are summarized in Table 5.17. Both (**MM-Oct-TID-C**) and (**MM-Oct-TID**) succeeded in (re-)establishing all assertions across all benchmarks. It is of particular interest that many of the assertions can already be established by the non-relational configurations (**MM-I**), (**Protection-I**), and (**Protection-I-TID**)— hinting at the fact that a majority of the invariants are not truly relational. For the relational analyses, enabling refinement according to thread *id*s yielded a significant improvement in proven assertions. Interestingly, this is not the case for the protection-based approach, where the precision remains the same, except for the program `snoopy`, where (**Protection-I-TID**) fails to establish 15 invariants that (**Protection-I**) can establish.

5.3.3 Summary

To sum up the results w.r.t. **(RQ4–5)**, enabling thread *id*s and the relational octagon domain did — while costly when compared with non-relational approaches — yield a significant improvement in precision. As the additional overhead of tracking all clusters of size up to 2 instead of only monolithic clusters is often negligible, the (**MM-Oct-TID-C**) configuration seems to be an attractive choice for the analysis of concurrent C programs. It would be interesting to investigate to which degree the orthogonal techniques of packing and online decomposition can be applied in the setting of this analysis — to overcome the encountered scalability issues.

5.4 Threats to Validity

When it comes to how well the results of this experimental evaluation are expected to generalize to other programs that do not form part of the benchmark suite, the following threats to validity need to be considered:

- The selection of benchmarks might not be diverse enough to generalize. In particular, as precision differences between the various non-relational analyses can be observed on litmus examples with domains which do not require widening,

Table 5.15: Number of programs for which each approach considering clusters of globals yielded the most precise result for the 28 programs on which all approaches terminated. The number after the slash indicates for how many programs no other approach yielded a more precise result, but there are incomparable results The numbers in parentheses indicate the number of programs where the best result was achieved by the approach (resp., no other approach reached a more precise result), but not all approaches terminated.

(**MM-I**)	(**MM-Oct**)	(**MM-Oct-TID**)	(**MM-Oct-TID-C**)
0/3 (+1/3)	15/3 (+5/3)	22/4 (+2/0)	**23**/4

Table 5.16: Precision comparison for the 4 programs where the relationship between precisions of analyses considering multiple globals is not straightforward. The table shows the comparison between the results from the analysis corresponding to the row and the one corresponding to the column. Incomparable results are denoted by ⫋⫌.

	(MM-I)	(MM-Oct)	(MM-Oct-TID)	(MM-Oct-TID-C)	(MM-I)	(MM-Oct)	(MM-Oct-TID)	(MM-Oct-TID-C)	(MM-I)	(MM-Oct)	(MM-Oct-TID)	(MM-Oct-TID-C)
	`ypbind`				`cava`				`fzy`			
(MM-I)		⊐	⫋⫌	⫋⫌		⫋⫌	⫋⫌	⫋⫌		⊐	⊐	⊐
(MM-Oct)	⊏		⫋⫌	⫋⫌	⫋⫌		⊐	⊐	⊏		⊏	⊏
(MM-Oct-TID)	⫋⫌	⫋⫌		=	⫋⫌	⊏		=	⊏	⊐		=
(MM-Oct-TID-C)	⫋⫌	⫋⫌	=		⫋⫌	⊏	=		⊏	⊐	=	
	`level-ip`											
(MM-I)		⊐	⫋⫌	⊐								
(MM-Oct)	⊏		⫋⫌	⊐								
(MM-Oct-TID)	⫋⫌	⫋⫌		⊐								
(MM-Oct-TID-C)	⊏	⊏	⊏									

Table 5.17: Summary of evaluation results on real-world benchmark enhanced with invariants with individual programs grouped together. For each group the number of programs and the total number of assertions are given. ✓ indicates that all assertions are proven, otherwise the number of proven assertions is given.

					Relational				
Set	Group	#	Asserts	(MM-I)	(MM-Oct)	(MM-Oct-TID)	(MM-Oct-TID-C)	(Protection-I)	(Protection-I-TID)
Goblint	POSIX	6	3304	3221	3291	✓	✓	3221	3221
	SV-COMP	7	190	✓	✓	✓	✓	✓	✓
Concrat		16	3962	3346	3910	✓	✓	3719	3704

it would seem plausible that there also exist real-world programs, where such differences are observable. Such situations may be relatively rare, but in these cases, the more involved techniques may be crucial to obtaining meaningful invariants.

- The Goblint analyzer comes with built-in support for single-threaded mode, where the values are tracked flow-sensitively until a first thread is potentially started. This support is deeply integrated into the framework and could not be disabled for the experiments conducted here. This has some systematic impact on the results: The approaches that use sets of protecting mutexes benefit to a greater extent than other approaches, as accesses to globals are likely not protected by mutexes while no threads have been started. As the set of write protecting mutexes computed by Goblint does not account for accesses in single-threaded mode, the set will more often be non-empty than when implemented exactly as presented here. By the same token, this single-threaded mode decreases the potential effect of refining according to thread *id*s, as the analysis does not need to rely on thread *id*s to be able to distinguish writes that happen before the first additional thread is started.

- Due to the difficulties comparing the results of context-sensitive analyses with potentially different contexts, all comparisons were done after performing a join over all contexts. This join loses information that may be of interest depending on the application. For example, if the analysis establishes that a variable can have only two different values that occur in different contexts, a compiler could specialize this code. However, this information is lost in the join. As a result, the joined values tend to underestimate precision differences between the approaches.

- For the relational analyses, the problems associated with carry-over effects [12] are not taken into account in this evaluation. This applies where abstract states at program points were compared with each other. However, the approach where programs were annotated with computed invariants does not suffer from this issue, partly mitigating this threat to validity. Nevertheless, it would be interesting to evaluate to which degree carry-over effects are present, perhaps by employing an improved comparison algorithm as recently proposed by Ballou and Sherman [12].

- The experiment runs were performed without systematic limiting of computational resources as offered, e.g., by the BENCHEXEC tool [17] making the reported runtimes potentially unreliable. To remedy this, whenever runtime was measured, tasks were executed sequentially with only the background processes of the operating system running concurrently. Thus, any random fluctuations of runtimes likely affect all approaches equally. Also, where we draw conclusions from runtimes, the differences in runtime are significant and can thus not be explained by such effects.

- As explained before, the widening/narrowing strategy employed by the solver affects the results, as does the evaluation order. While nothing seems to hint at the strategy employed by the TD solvers used for this evaluation particularly (dis-)advantaging one of the approaches, it is perhaps important to remark that the solver originally employed for the analyses proposed by Miné [83, 84] has a different iteration order: There, threads are iterated to an (intermediate) local fixpoint before the next thread is considered, whereas the solver of the GOBLINT system may work on unknowns belonging to several threads at the same time. It would be interesting to repeat the evaluation with different solvers and see to what degree they affect the result.

We believe that all these concerns are sufficiently addressed in the design of our experiments and that thus the conclusions we draw are well-supported.

5.5 Summary of Experimental Evaluation

To sum up the results of our experimental evaluation in a nutshell, we find that w.r.t. **(RQ1)** the configuration (**Protection**) is the fastest across the board, also when comparing settings where intervals and thread *id*s enabled. The other approaches are often considerably slower — with the runtime difference within these approaches usually not too high. For **(RQ2)**, we observe that quite a high number of thread *id*s is identified, and the number of thread *id*s determined to be unique is encouraging. W.r.t. **(RQ3)**, we find that, while other analyses are more precise, (**Protection**) offers the same precision on the majority of benchmarks. For the configurations using intervals, the configuration (**Protection-I**) is not only more scalable than the other analyses, but also often more

precise. For the non-relational analyses, there is little benefit (and even some potential harm) when enabling thread *id*s, making (**Protection**) and (**Protection-I**) perhaps the most practically relevant configurations.

W.r.t. **(RQ4–5)**, we find that enabling thread *id*s and the relational octagon domain — while costly when compared with non-relational approaches — offers a significantly improved precision. As the additional overhead of tracking all clusters of size up to 2 instead of only monolithic clusters is often negligible, the (**MM-Oct-TID-C**) configuration seems to be an attractive choice for the analysis of concurrent C programs.

Nevertheless, none of the configurations seems to be a *silver bullet*, indicating opportunities for future research, e.g., around automatically selecting the most promising approach for a given program, or perhaps developing techniques to combine these approaches such that more expensive techniques are only used locally where needed, and the rest of the program is analyzed in a more coarse manner.

6 Soundness Proofs for the Analyses

This chapter provides soundness proofs for the analyses presented in Chapter 4. In our previous work [107, 108], we quickly sketch ideas for such soundness proofs. We also have informally published appendices on ArXiv detailing some of these arguments. Here, we provide more extensive and detailed proofs and also cover the extensions developed in this work, such as, e.g., the notion of ego-lane digests or the extended language also featuring signal and wait.

The proofs often require some form of induction over approximations to (least) solutions of the concrete and abstract constraint systems, with intermediate steps being taken to bring these closer together. This unfortunately renders them quite technical at times. Readers who are in a hurry may want to skip the details of the proofs and only read the statements of the theorems, propositions, and (selectively) the most interesting cases in the inductive proofs, which we will attempt to highlight wherever possible.

The structure of this chapter follows the same structure as Chapter 4: Section 6.1 provides proofs for the analyses from Section 4.1 which consider globals in isolation. The succeeding Section 6.2 provides proofs for the analyses proposed in Section 4.2 which consider clusters of globals.

6.1 Soundness Proofs for Analyses Considering Globals in Isolation

To be able to provide soundness proofs for the Lock-Centered and Write-Centered Reading analyses with ego-lane digests, we first define a few helper functions on local traces that will be useful for both proofs.

For every node $\bar{u} = (j, u, \sigma) \in \mathbf{V}$ of a local trace t, we define the lockset $L_t[\bar{u}]$ as the set of mutexes which have been acquired by the thread with thread *id* σ self and not yet released. This set can be defined inductively by keeping track of the lock and unlock operations of the thread with thread *id* σ self, akin to how the digest computing locksets is defined (Fig. 2.11). We abbreviate by L_t the lockset $L_t[\bar{u}]$ where $\bar{u}$ is the maximal node of t, i.e., the lockset of the ego thread at the sink.

Recall that $\mathsf{last\_write}_g\ t$ extracts the last write to g (or the call of initMT if none exists) from a local trace t. We call a write occurring at an edge $(\bar{u}, g = x, \bar{u}')$ in the raw ego trace, i.e., along the ego lane of the ego thread in the local trace, *thread-local*. If there is a thread-local write to a global g in a local trace t, there also is a *last* thread-local write to g in t. Let $\mathsf{last\_tl\_write}_g : \mathcal{T} \to \mathbf{E} \cup \{\bot\}$ be a function to extract the last thread-local write to g from a local trace if it exists, and return $\bot$ otherwise. For a set T of local

traces, we define the set of values that are written at last thread-local writes to g by

$$\mathsf{eval\_tl}_g\, T = \{\sigma\, x \mid t \in T, \mathsf{last\_tl\_write}_g\, t = ((j-1, u, \sigma), g = x, \bar{u}')\}$$

Similarly, we call a lock at an edge $(\bar{u}, \mathsf{lock}(a), \bar{u}')$ in the raw ego trace *thread-local*. If there is a thread-local lock of a mutex a, there also is a *last* thread-local lock of a. Let $\mathsf{last\_tl\_lock}_a : \mathcal{T} \to \mathbf{E} \cup \{\bot\}$ be a function to extract the last thread-local lock of a if it exists, and return $\bot$ otherwise. Analogously for $\mathsf{last\_tl\_unlock}_a : \mathcal{T} \to \mathbf{E} \cup \{\bot\}$ and the last thread-local unlock of a.

Additionally, we define a function $\mathsf{min\_lockset\_since} : \mathcal{T} \to \mathbf{V} \to \mathcal{U}_{\mathcal{M}}$ that extracts the upwards-closed set of minimal locksets the ego thread has held since a given node of the raw ego trace. Again, $\mathsf{min\_lockset\_since}\,(t, \bar{u})$ can be computed inductively by considering the raw ego trace only.

Furthermore, let $\mathsf{init\_v}\, t : \mathcal{T} \to \mathbf{V} \cup \{\bot\}$ be a function to extract the node from a local trace in which the call to initMT ends if it exists, and return $\bot$ otherwise.

In both proofs, we will consider some ego-lane digest $\mathcal{A}$ and an appropriate definition of $\mathsf{compat}^\sharp_{\mathcal{A}}$. We consider the refined constraint system for the concrete semantics (Eq. (2.12)) instantiated with the considered actions from Section 2.4 and the product digest of $\mathcal{A}$ and the lockset digest from Fig. 2.11. We remark that this product digest is once more ego-lane-derived with its compatibility function $\mathsf{compat}^\sharp_{\mathcal{A}}$ considering only the compatibility of $\mathcal{A}$. This constraint system then has the following set of unknowns:

- $[u, S, A]$ for $u \in \mathcal{N}$, $S \subseteq \mathcal{M}$ and $A \in \mathcal{A}$,
- $[a, S, A]$ for $g \in \mathcal{G}, a \in \mathcal{M}, S \subseteq \mathcal{M}, w \subseteq \mathcal{M}$ and $A \in \mathcal{A}$,
- $[\mathsf{initMT}, \emptyset, A]$ for $A \in \mathcal{A}$,
- $[\mathsf{return}, S, A]$ for $S \subseteq \mathcal{M}$ and $A \in \mathcal{A}$, and
- $[s, S, A]$ for $s \in \mathcal{S}$, $S \subseteq \mathcal{M}$ and $A \in \mathcal{A}$.

where we (above and subsequently) abbreviate $\mathsf{unlock}(a)$ by a and $\mathsf{signal}(s)$ by s. In the subsequent two proofs, let us refer to this constraint system by $\mathcal{C}$ and to the corresponding right-hand sides by $[\![\cdot]\!]$.

As the unknowns for Write-Centered Reading with ego-lane digests perform more splitting than the unknowns for Lock-Centered Reading with ego-lane digests, we will show the soundness of both w.r.t. a modified constraint system over sets of local traces where the unknowns match those for Write-Centered Reading with ego-lane digests. Therefore, the proofs in this section are not in the same order in which the analyses are presented in Section 4.1.

6.1.1 Write-Centered Reading

Consider some ego-lane digest $\mathcal{A}$ and an appropriate definition of $\mathsf{compat}^\sharp_{\mathcal{A}}$. Let us refer to the constraint system of the analysis from Section 4.1.7 refined with $\mathcal{A}$ by $\mathcal{C}^\sharp_{\mathsf{wc}}$. As a first step, we construct a new constraint system $\mathcal{C}_{\mathsf{wc}}$ over sets of local traces for which the unknowns match those of $\mathcal{C}^\sharp_{\mathsf{wc}}$ — up to the unknowns used for thread returns. We then show the equivalence of $\mathcal{C}_{\mathsf{wc}}$ and $\mathcal{C}$.

$\mathcal{C}_{\mathsf{wc}}$ thus has the following set of unknowns:

- $[u, S, A]$ for $u \in \mathcal{N}$, $S \subseteq \mathcal{M}$, and $A \in \mathcal{A}$,
- $[g, a, S, w, A]$ for $g \in \mathcal{G}, a \in \mathcal{M}, S \subseteq \mathcal{M}, w \subseteq \mathcal{M}$, and $A \in \mathcal{A}$,
- $[i, A]$ for $i \in \mathcal{V}_{\mathbf{tid}}$ and $A \in \mathcal{A}$, and
- $[s, A]$ for $s \in \mathcal{S}$ and $A \in \mathcal{A}$.

The constraints of $\mathcal{C}_{\mathsf{wc}}$ are defined as follows:

$$
\begin{aligned}
&[u_0, \emptyset, A] && \supseteq \quad \mathbf{fun}\ \_ \to (\emptyset, \{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\}) \\
&\quad \text{for } A \in \mathsf{init}^\sharp_{\mathcal{A}} \\
&[u', S, A'] && \supseteq \quad [\![([u, S, A_0], x = \mathsf{create}(u_1), u')]\!]_{\mathsf{wc}} \\
&\quad \text{for } (u, x = \mathsf{create}(u_1), u') \in \mathcal{E}, A' \in [\![u, x = \mathsf{create}(u_1)]\!]^\sharp_{\mathcal{A}}(A_0) \\
&[u', S \cup \{a\}, A'] && \supseteq \quad [\![([u, S, A_0], \mathsf{lock}(a), u')]\!]_{\mathsf{wc}} \\
&\quad \text{for } (u, \mathsf{lock}(a), u') \in \mathcal{E}, A' \in \textstyle\bigcup_{A_1 \in \mathcal{A}} \{[\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}(A_0, A_1)\} \\
&[u', S, A'] && \supseteq \quad [\![([u, S, A_0], \mathsf{act}, u')]\!]_{\mathsf{wc}} \\
&\quad \text{for } (u, \mathsf{act}, u') \in \mathcal{E}, \mathsf{act} \in \mathcal{A}ct_{\mathsf{observing}}, \text{not lock}, A' \in \textstyle\bigcup_{A_1 \in \mathcal{A}} \{[\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0, A_1)\} \\
&[u', S \setminus \{a\}, A'] && \supseteq \quad [\![([u, S, A_0], \mathsf{unlock}(a), u'), A']\!]_{\mathsf{wc}} \\
&\quad \text{for } (u, \mathsf{unlock}(a), u') \in \mathcal{E}, A' \in [\![u, \mathsf{unlock}(a)]\!]^\sharp_{\mathcal{A}}(A_0) \\
&[u', S, A'] && \supseteq \quad [\![([u, S, A_0], \mathsf{act}, u'), A']\!]_{\mathsf{wc}} \\
&\quad \text{for } (u, \mathsf{act}, u') \in \mathcal{E}, \mathsf{act} \in \mathcal{A}ct_{\mathsf{observable}}, \text{not unlock or initMT}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0) \\
&[u', S, A'] && \supseteq \quad [\![([u, S, A_0], \mathsf{act}, u')]\!]_{\mathsf{wc}} \\
&\quad \text{for } (u, \mathsf{act}, u') \in \mathcal{E}, \mathsf{act} \in \mathcal{A}ct_{\mathsf{local}}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0) \\
&[u', \emptyset, A'] && \supseteq \quad [\![([u, \emptyset, A_0], \mathsf{initMT}, u')]\!]_{\mathsf{wc}} \\
&\quad \text{for } (u, \mathsf{initMT}, u') \in \mathcal{E}, A' \in [\![u, \mathsf{initMT}]\!]^\sharp_{\mathcal{A}}(A_0)
\end{aligned}
$$

The right-hand sides of $\mathcal{C}_{\mathsf{wc}}$ are then given by

$$
\begin{aligned}
&[\![([u, S, A_0], x = \mathsf{create}(u_1), u')]\!]_{\mathsf{wc}}\, \eta_{\mathsf{wc}} = \\
&\quad \mathbf{let}\ T = [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{wc}}\,[u, S, A_0])\ \mathbf{in} \\
&\quad (\{[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0] \mapsto \mathsf{new}\, u_1\, (\eta_{\mathsf{wc}}\,[u, S, A_0])\}, T)
\end{aligned}
$$

$[\![([u, S, A_0], \mathsf{lock}(a), u')]\!]_{\mathsf{wc}}\, \eta_{\mathsf{wc}} =$
$\quad$ **let** $T_1 = \bigcup\{\eta_{\mathsf{wc}}\,[g, a, S', w, A_1] \mid g \in \mathcal{G}, S' \subseteq \mathcal{M}, w \subseteq \mathcal{M}, A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\}$ **in**
$\quad$ **let** $T_2 = \{t' \mid t \in \eta_{\mathsf{wc}}\,[u, S, A_0], \bar{\imath} = \mathsf{init\_v}\, t, \bar{\imath} \neq \bot, t' = \downarrow_{\bar{\imath}}(t), \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, (\alpha_{\mathcal{A}}\, t')\}$ **in**
$\quad$ **let** $T = [\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{wc}}\,[u, S, A_0], T_1 \cup T_2)$ **in**
$\quad$ $(\emptyset, T)$

$[\![([u, S, A_0], \mathsf{unlock}(a), u'), A']\!]_{\mathsf{wc}}\, \eta_{\mathsf{wc}} =$
$\quad$ **let** $T = [\![(u, \mathsf{unlock}(a), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{wc}}\,[u, S, A_0])$ **in**
$\quad$ **let** $\rho = \{[g, a, S \setminus \{a\}, w, A'] \mapsto \{t\} \mid t \in T, g \in \mathcal{G}, w \subseteq \mathcal{M},$
$\qquad ((\mathsf{last\_tl\_write}_g\, t = (\bar{u}, g = x, \bar{u}') \wedge L_t[\bar{u}] \subseteq w) \vee (\mathsf{last\_tl\_write}_g\, t = \bot))\}$
$\quad$ **in**
$\quad$ (ρ, T)

$[\![([u, S, A_0], \mathsf{return}, u'), A']\!]_{\mathsf{wc}}\, \eta_{\mathsf{wc}} =$
$\quad$ **let** $T = [\![(u, \mathsf{return}, u')]\!]_{\mathcal{T}}(\eta_{\mathsf{wc}}\,[u, S, A_0])$ **in**
$\quad$ **let** $\rho = \{[i, A'] \mapsto \{t \mid t \in T, \mathsf{id}\, t = i\} \mid i \in \mathcal{V}_{\mathbf{tid}}\}$ **in**
$\quad$ (ρ, T)

$[\![([u, S, A_0], x = \mathsf{join}(x'), u')]\!]_{\mathsf{wc}}\, \eta_{\mathsf{wc}} =$
$\quad$ **let** $T_1 = \bigcup\{\eta_{\mathsf{wc}}\,[t(x'), A_1] \mid t \in \eta_{\mathsf{wc}}\,[u, S, A_0], A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\}$ **in**
$\quad$ **let** $T = [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{wc}}\,[u, S, A_0], T_1)$ **in**
$\quad$ $(\emptyset, T)$

$[\![([u, S, A_0], \mathsf{signal}(s), u'), A']\!]_{\mathsf{wc}}\, \eta_{\mathsf{wc}} =$
$\quad$ **let** $T = [\![(u, \mathsf{signal}(s), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{wc}}\,[u, S, A_0])$ **in**
$\quad$ **let** $\rho = \{[s, A'] \mapsto T\}$ **in**
$\quad$ $(\emptyset, T)$

$[\![([u, S, A_0], \mathsf{wait}(s), u')]\!]_{\mathsf{wc}}\, \eta_{\mathsf{wc}} =$
$\quad$ **let** $T_1 = \bigcup\{\eta_{\mathsf{wc}}\,[s, A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\}$ **in**
$\quad$ **let** $T = [\![(u, \mathsf{wait}(s), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{wc}}\,[u, S, A_0], T_1)$ **in**
$\quad$ $(\emptyset, T)$

$[\![([u, S, A_0], \mathsf{act}, u')]\!]_{\mathsf{wc}}\, \eta_{\mathsf{wc}} =$
$\quad$ **let** $T = [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta_{\mathsf{wc}}\,[u, S, A_0])$ **in**
$\quad$ $(\emptyset, T)$

where act is either a local action or initMT.

Proposition 14. *The right-hand side function of constraint system $\mathcal{C}_{wc}$ over the lattice mapping (extended) unknowns to sets of local traces with the order as discussed in Section 2.2.2 is Scott-continuous.*

Proof. The proof here proceeds in the same manner as the proof of Proposition 10 in Section 2.3. After collecting all right-hand sides into one constraint using the Scott-continuous helper function $\mathsf{flat}_{[x]}$, it remains to show that the individual right-hand sides are a composition of Scott-continuous functions. This follows from the Scott-continuity

of $[\![\cdot]\!]_{\mathcal{T}}$, new, and $\pi_{[x]}$ for $[x] \in \mathsf{X}$, where the same insight into the right-hand side for observable actions already used in the proof of Proposition 8 is used once again. Furthermore, all arguments to $[\![\cdot]\!]_{\mathcal{T}}$ are constructed in a way where either the union of sets is taken, or the resulting set is constructed as the union of some function applied to each element of the set in isolation, also rendering these functions Scott-continuous. Then, the combined right-hand side is given as the least upper bound of (compositions of) functions that are Scott-continuous, and is thus also Scott-continuous. □

Proposition 15. *The constraint system $\mathcal{C}_{\mathsf{wc}}$ has a least solution which is obtained as the least-upper bound of all Kleene iterates.*

For a mapping η from unknowns of $\mathcal{C}$ to sets of local traces, we define a mapping $\mathsf{split}[\eta]$ from unknowns of $\mathcal{C}_{\mathsf{wc}}$ to sets of local traces as follows:

$$
\begin{array}{lll}
\mathsf{split}[\eta][u,S,A] & = \eta[u,S,A] & (\text{for } u \in \mathcal{N}, S \subseteq \mathcal{M}, A \in \mathcal{A}) \\
\mathsf{split}[\eta][g,a,S,w,A] & = \eta[a,S,A] \cap \{t \in \mathcal{T} \mid & (\text{for } g \in \mathcal{G}, a \in \mathcal{M}, S \subseteq \mathcal{M}, \\
 & \quad (\mathsf{last\_tl\_write}_g\, t = (\bar{u}, g = x, \bar{u}') & \quad w \subseteq \mathcal{M}, A \in \mathcal{A}) \\
 & \qquad \wedge L_t[\bar{u}'] \subseteq w) & \\
 & \quad \vee (\mathsf{last\_tl\_write}_g\, t = \bot)\} & \\
\mathsf{split}[\eta][i,A] & = \{t \in \bigcup_{S \subseteq \mathcal{M}} \eta[\mathsf{return}, S, A] \mid & (\text{for } i \in \mathcal{V}_{\mathbf{tid}}, A \in \mathcal{A}) \\
 & \quad \mathsf{id}\, t = i\} & \\
\mathsf{split}[\eta][s,A] & = \bigcup_{S \subseteq \mathcal{M}} \eta[s,S,A] & (\text{for } s \in \mathcal{S}, S \subseteq \mathcal{M}, A \in \mathcal{A})
\end{array}
$$

Conversely, we obtain

$$
\begin{array}{lcl}
\eta[u,S,A] & = & \mathsf{split}[\eta][u,S,A] \\
\eta[a,S,A] & = & \bigcup_{g \in \mathcal{G}, w \subseteq \mathcal{M}} \mathsf{split}[\eta][g,a,S,w,A] \\
\eta[\mathsf{return},S,A] & = & \{t \in \bigcup_{i \in \mathcal{V}_{\mathbf{tid}}} \mathsf{split}[\eta][i,A] \mid L_t = S\} \\
\eta[s,S,A] & = & \{t \in \mathsf{split}[\eta][s,A] \mid L_t = S\}
\end{array}
$$

and obtain a new mapping η' extending it back to all unknowns of $\mathcal{C}$ by setting

$$
\begin{array}{lcl}
 & & \eta'[\mathsf{initMT}, \emptyset, A] = \mathsf{uni}[\eta][\mathsf{initMT}, \emptyset, A] \\
= & & \{t' \mid t \in \bigcup_{u \in \mathcal{N}, A' \in \mathcal{A}, S \subseteq \mathcal{M}} \mathsf{split}[\eta][u,S,A'], \bar{t} = \mathsf{init\_v}\, t, \bar{t} \neq \bot, t' = \downarrow_{\bar{t}} (t), \\
 & & \quad (\alpha_{\mathcal{A}}\, t') = A\}
\end{array}
$$

where we abbreviate the right-hand side by $\mathsf{uni}[\eta][\mathsf{initMT}, \emptyset, A]$.

Proposition 16. *The following two statements hold:*

(A) If η is the least solution of $\mathcal{C}$, $\mathsf{split}[\eta]$ is the least solution of $\mathcal{C}_{\mathsf{wc}}$.

(B) If $\mathsf{split}[\eta]$ is the least solution of $\mathcal{C}_{\mathsf{wc}}$, η' as constructed above is the least solution of $\mathcal{C}$.

Proof. We prove both statements simultaneously by fixpoint induction: To simplify the proof, we once again consider the contributions of constraints one at a time. Consider the $i-$th approximation η^i to the least solution of $\mathcal{C}$, and the $i-$th approximation η^i_{wc} to the least solution of $\mathcal{C}_{\mathsf{wc}}$. Let us call property (1) that

$$\begin{array}{lrcll}
(a) & \eta^i_{\mathsf{wc}}\,[u,S,A] & = & \mathsf{split}[\eta^i]\,[u,S,A] & (\text{for } u \in \mathcal{N}, S \subseteq \mathcal{M}, A \in \mathcal{A}) \\
(b) & \eta^i_{\mathsf{wc}}\,[g,a,S,w,A] & = & \mathsf{split}[\eta^i][g,a,S,w,A] & (\text{for } g \in \mathcal{G}, a \in \mathcal{M}, S \subseteq \mathcal{M}, \\
 & & & & \qquad w \subseteq \mathcal{M}, A \in \mathcal{A}) \\
(c) & \eta^i_{\mathsf{wc}}\,[i,A] & = & \mathsf{split}[\eta^i][i,A] & (\text{for } i \in \mathcal{V}_{\mathbf{tid}}, A \in \mathcal{A}) \\
(d) & \eta^i_{\mathsf{wc}}[s,A] & = & \mathsf{split}[\eta^i][s,A] & (\text{for } s \in \mathcal{S}, A \in \mathcal{A}) \\
(e) & \eta^i[\mathsf{initMT},\emptyset,A] & = & \mathsf{uni}[\eta^i][\mathsf{initMT},\emptyset,A] & (\text{for } A \in \mathcal{A})
\end{array}$$

For $i = 0$, the value of all unknowns in both constraint systems is $\emptyset$, and property (1) holds trivially.

Next, we show that for constraints corresponding to a control-flow edge as well as the constraint for initialization executed in lock-step, provided that property (1) holds before the update, it still holds after the update. Considering the constraint's contribution to the unknown on the left and its side-effects (if any are triggered) suffices for this.

First, consider the constraints for **initialization**. They take identical form in both constraint systems:

$$[u_0,\emptyset,A] \quad \supseteq \quad \mathbf{fun}\,\_ \to (\emptyset, \{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\}) \qquad \text{for } A \in \mathsf{init}^\sharp_{\mathcal{A}}$$

As no unknown is accessed, only the contributions to the unknowns $[u_0,\emptyset,A]$ need to be considered. These new contributions could potentially affect sub-properties (a) and (e). If (a) holds, it still holds after the update, as the unknowns receive the same new contribution. For subproperty (e), we remark that for all contributions t, $\mathsf{last}\,t = \bot$ and thus $\mathsf{init\_v}\,t = \bot$. Therefore $\mathsf{uni}[\eta^{i+1}][\mathsf{initMT},\emptyset,A] = \mathsf{uni}[\eta^i][\mathsf{initMT},\emptyset,A]$, and as $\mathcal{C}_{\mathsf{wc}}$ does not cause any side-effects, subproperty (e) also holds after the update. Thus, if property (1) holds for the i-th approximations, and constraints of this form are considered, it also holds for the $(i+1)$-th approximations.

Next, consider the constraints for **initMT**. Consider an edge $(u,\mathsf{initMT},u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u,\mathsf{initMT}]\!]^\sharp_{\mathcal{A}}(A_0)$. We remark that, by construction, the lockset is empty when executing initMT. For $\mathcal{C}_{\mathsf{wc}}$, the constraints take the following form:

$$[u',\emptyset,A'] \quad \supseteq \quad [\![([u,\emptyset,A_0],\mathsf{initMT},u')]\!]_{\mathsf{wc}}$$

with right-hand side

$$[\![([u,\emptyset,A_0],\mathsf{initMT},u')]\!]_{\mathsf{wc}}\,\eta_{\mathsf{wc}} = (\emptyset, [\![(u,\mathsf{initMT},u')]\!]_{\mathcal{T}}(\eta_{\mathsf{wc}}\,[u,\emptyset,A_0]))$$

For $\mathcal{C}$, the constraints take the following form:

$$[u', \emptyset, A'] \quad \supseteq \quad [\![([u, \emptyset, A_0], \mathsf{initMT}, [u', \emptyset, A'])]\!]$$

with right-hand side

$$[\![([u, \emptyset, A_0], \mathsf{initMT}, [u', \emptyset, A'])]\!]\,\eta \quad = \quad \textbf{let } T = [\![(u, \mathsf{initMT}, u')]\!]_{\mathcal{T}}(\eta\,[u, \emptyset, A_0]) \textbf{ in } (\{[\mathsf{initMT}, \emptyset, A'] \mapsto T\}, T)$$

Provided property (1) holds for the i-th approximations, the unknowns $[u', \emptyset, A']$ of both constraint systems receive the same new contribution (subproperty (a)). Now, for the additional contribution T via side-effect of $\mathcal{C}$ to $[\mathsf{initMT}, \emptyset, A']$: We first remark that $T = \{t' \mid t \in T, \bar{i} = \mathsf{init\_v}\, t, \bar{i} \neq \bot, t' = \downarrow_{\bar{i}}(t), (\alpha_{\mathcal{A}}\, t') = A'\}$ holds here, as well as $\mathsf{split}[\eta^{i+1}][u', \emptyset, A'] = \mathsf{split}[\eta^{i}][u', \emptyset, A'] \cup T$. Thus, we have that

$$\begin{array}{ll}
 & \eta^{i+1}[\mathsf{initMT}, \emptyset, A'] = \eta^{i}[\mathsf{initMT}, \emptyset, A'] \cup T = \mathsf{uni}[\eta^{i}][\mathsf{initMT}, \emptyset, A'] \cup T \\
= & \{t' \mid t \in \bigcup_{u \in \mathcal{N}, A'' \in \mathcal{A}} \mathsf{split}[\eta^{i}][u, \emptyset, A''], \bar{i} = \mathsf{init_v}\, t, \bar{i} \neq \bot, t' = \downarrow_{\bar{i}}(t), (\alpha_{\mathcal{A}}\, t') = A'\} \cup \\
 & T \\
= & \{t' \mid t \in \bigcup \{\mathsf{split}[\eta^{i}][u, \emptyset, A''] \mid u \in \mathcal{N}, A'' \in \mathcal{A}, (u \neq u' \vee A'' \neq A')\}, \\
 & \quad \bar{i} = \mathsf{init_v}\, t, \bar{i} \neq \bot, t' = \downarrow_{\bar{i}}(t), (\alpha_{\mathcal{A}}\, t') = A'\} \cup \\
 & \{t' \mid t \in \mathsf{split}[\eta^{i}][u', \emptyset, A'], \bar{i} = \mathsf{init_v}\, t, \bar{i} \neq \bot, t' = \downarrow_{\bar{i}}(t), (\alpha_{\mathcal{A}}\, t') = A'\} \cup T \\
= & \{t' \mid t \in \bigcup \{\mathsf{split}[\eta^{i+1}][u, \emptyset, A''] \mid u \in \mathcal{N}, A'' \in \mathcal{A}, (u \neq u' \vee A'' \neq A')\}, \\
 & \quad \bar{i} = \mathsf{init_v}\, t, \bar{i} \neq \bot, t' = \downarrow_{\bar{i}}(t), (\alpha_{\mathcal{A}}\, t') = A'\} \cup \\
 & \{t' \mid t \in (\mathsf{split}[\eta^{i}][u', \emptyset, A'] \cup T), \bar{i} = \mathsf{init_v}\, t, \bar{i} \neq \bot, t' = \downarrow_{\bar{i}}(t), (\alpha_{\mathcal{A}}\, t') = A'\} \\
= & \{t' \mid t \in \bigcup_{u \in \mathcal{N}, A'' \in \mathcal{A}} \mathsf{split}[\eta^{i+1}][u, \emptyset, A''], \bar{i} = \mathsf{init_v}\, t, \bar{i} \neq \bot, t' = \downarrow_{\bar{i}}(t), \\
 & \quad (\alpha_{\mathcal{A}}\, t') = A'\} \\
= & \mathsf{uni}[\eta^{i+1}][\mathsf{initMT}, \emptyset, A']
\end{array}$$

and subproperty (e) also holds after the update. Thus, if property (1) holds for the i-th approximations, and constraints of this form are considered, it also holds for the $(i+1)$-th approximations.

Next, consider the constraints for **local** actions. Consider an edge $(u, \mathsf{act}, u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, \mathsf{act}]\!]^{\sharp}_{\mathcal{A}}(A_0)$. For $\mathcal{C}_{\mathsf{wc}}$, the constraints take the following form:

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{act}, u')]\!]_{\mathsf{wc}}$$

with right-hand side

$$[\![([u, S, A_0], \mathsf{act}, u')]\!]_{\mathsf{wc}}\,\eta_{\mathsf{wc}} = (\emptyset, [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta_{\mathsf{wc}}\,[u, S, A_0]))$$

For $\mathcal{C}$, the constraints take the following form:

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{act}, u')]\!]$$

with right-hand side

$$[\![([u, S, A_0], \mathsf{act}, u')]\!]\,\eta \quad = \quad (\emptyset, [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta\,[u, S, A_0]))$$

Provided property (1) holds for the i-th approximations, the unknowns $[u', S, A']$ of both constraint systems receive the same new contribution (subproperty (a)). Now consider subproperty (e). As the action act is local, it is not initMT. Thus, for the new contribution T to $[u', S, A']$, and all $A \in \mathcal{A}$,

$$\begin{array}{rl} & \{t' \mid t \in T, \bar{i} = \mathsf{init\_v}\, t, \bar{i} \neq \bot, t' = \downarrow_{\bar{i}} (t), (\alpha_{\mathcal{A}}\, t') = A\} \\ \subseteq & \{t' \mid t \in \eta^i_{\mathsf{wc}}\, [u, S, A_0], \bar{i} = \mathsf{init\_v}\, t, \bar{i} \neq \bot, t' = \downarrow_{\bar{i}} (t), (\alpha_{\mathcal{A}}\, t') = A\} \end{array}$$

and thus $\mathsf{uni}[\eta^i][\mathsf{initMT}, \emptyset, A] = \mathsf{uni}[\eta^{i+1}][\mathsf{initMT}, \emptyset, A]$, which, as the constraint in $\mathcal{C}$, does not cause any side-effects, implies that subproperty (e) also holds after the update. Thus, if property (1) holds for the i-th approximations, and constraints of this form are considered, it also holds for the $(i+1)$-th approximations.

Next, consider the constraints corresponding to **thread creation**. Consider an edge $(u, x = \mathsf{create}(u_1), u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, x = \mathsf{create}(u_1)]\!]^\sharp_{\mathcal{A}}(A_0)$. For $\mathcal{C}_{\mathsf{wc}}$, the constraints take the following form:

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], x = \mathsf{create}(u_1), u')]\!]_{\mathsf{wc}}$$

with right-hand side

$$\begin{array}{l} [\![([u, S, A_0], x = \mathsf{create}(u_1), u')]\!]_{\mathsf{wc}}\, \eta_{\mathsf{wc}} = \\ \quad \textbf{let}\ T = [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{wc}}\, [u, S, A_0])\ \textbf{in} \\ \quad (\{[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0] \mapsto \mathsf{new}\, u_1\, (\eta_{\mathsf{wc}}\, [u, S, A_0])\}, T) \end{array}$$

For $\mathcal{C}$, the constraints take the following form:

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], x = \mathsf{create}(u_1), u')]\!]$$

with right-hand side

$$\begin{array}{l} [\![([u, S, A_0], x = \mathsf{create}(u_1), u')]\!]\, \eta = \\ \quad \textbf{let}\ T = [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\eta\, [u, S, A_0])\ \textbf{in} \\ \quad (\{[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0] \mapsto \mathsf{new}\, u_1\, (\eta\, [u, S, A_0])\}, T) \end{array}$$

Provided property (1) holds for the i-th approximations, the unknowns on the left-hand sides of both constraint systems receive the same new contribution, as do the unknowns that receive side-effects (subproperty (a)). W.r.t. subproperty (e), the argument provided for the case of local actions also applies here. Thus, if property (1) holds for the i-th approximations, and constraints of this form are considered, it also holds for the $(i+1)$-th approximations.

Next, consider the constraints corresponding to **lock**. Consider an edge $(u, \mathsf{lock}(a), u') \in \mathcal{E}$ and digests A', A_0, s.t. $\forall A_1 \in \mathcal{A}$, $[\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}(A_0, A_1) \in \{\{A'\}, \emptyset\}$, which exists as $\mathcal{A}$ here is ego-lane-derived. For $\mathcal{C}_{\mathsf{wc}}$, the constraints take the following form:

$$[u', S \cup \{a\}, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{lock}(a), u')]\!]_{\mathsf{wc}}$$

with right-hand side

$$
\begin{array}{l}
[\![([u,S,A_0],\mathsf{lock}(a),u')]\!]_{\mathsf{wc}}\,\eta_{\mathsf{wc}} = \\
\quad \textbf{let } T_1 = \bigcup\{\eta_{\mathsf{wc}}\,[g,a,S',w,A_1] \mid g \in \mathcal{G}, S' \subseteq \mathcal{M}, w \subseteq \mathcal{M}, A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1\} \textbf{ in} \\
\quad \textbf{let } T_2 = \{t' \mid t \in \eta_{\mathsf{wc}}\,[u,S,A_0], \bar{i} = \mathsf{init\_v}\,t, \bar{i} \neq \bot, t' = \downarrow_{\bar{i}}(t), \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,(\alpha_{\mathcal{A}}\,t')\} \textbf{ in} \\
\quad \textbf{let } T = [\![(u,\mathsf{lock}(a),u')]\!]_{\mathcal{T}}(\eta_{\mathsf{wc}}\,[u,S,A_0], T_1 \cup T_2) \textbf{ in} \\
\quad (\emptyset, T)
\end{array}
$$

For $\mathcal{C}$, the constraints take the following form for all A_1 where $[\![u,\mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}(A_0,A_1) = \{A'\}$,

$$[u', S \cup \{a\}, A'] \quad \supseteq \quad [\![([u,S,A_0],\mathsf{lock}(a),u'),A_1]\!]$$

with right-hand side

$$
\begin{array}{l}
[\![([u,S,A_0],\mathsf{lock}(a),u'),A_1]\!]\,\eta = \\
\quad \textbf{let } T_1 = [\![(u,\mathsf{lock}(a),u')]\!]_{\mathcal{T}}\,(\eta\,[u,S,A_0], \eta\,[\mathsf{initMT},\emptyset,A_1]) \textbf{ in} \\
\quad \textbf{let } T_2 = \bigcup_{S' \subseteq \mathcal{M}} [\![(u,\mathsf{lock}(a),u')]\!]_{\mathcal{T}}\,(\eta\,[u,S,A_0], \eta\,[a,S',A_1]) \textbf{ in} \\
\quad (\emptyset, T_1 \cup T_2)
\end{array}
$$

As a first step, we consider all contributions to $[u, S \cup \{a\}, A']$ caused by $\mathcal{C}$ together in order to relate them to the contributions of $\mathcal{C}_{\mathsf{wc}}$. These are then given by

$$
\begin{array}{rl}
 & \bigcup_{A_1 \in \mathcal{A}, [\![u,\mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}(A_0,A_1)=\{A'\}} ([\![(u,\mathsf{lock}(a),u')]\!]_{\mathcal{T}}\,(\eta^i\,[u,S,A_0], \eta^i\,[\mathsf{initMT},\emptyset,A_1]) \cup \\
 & \qquad \bigcup_{S' \subseteq \mathcal{M}} [\![(u,\mathsf{lock}(a),u')]\!]_{\mathcal{T}}\,(\eta^i\,[u,S,A_0], \eta^i\,[a,S',A_1])) \\
= & \bigcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1} ([\![(u,\mathsf{lock}(a),u')]\!]_{\mathcal{T}}\,(\eta^i\,[u,S,A_0], \eta^i\,[\mathsf{initMT},\emptyset,A_1]) \cup \\
 & \qquad \bigcup_{S' \subseteq \mathcal{M}} [\![(u,\mathsf{lock}(a),u')]\!]_{\mathcal{T}}\,(\eta^i\,[u,S,A_0], \eta^i\,[a,S',A_1])) \\
= & [\![(u,\mathsf{lock}(a),u')]\!]_{\mathcal{T}}(\eta^i\,[u,S,A_0], \bigcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1} (\eta^i\,[\mathsf{initMT},\emptyset,A_1] \cup \\
 & \qquad \bigcup_{S' \subseteq \mathcal{M}} \eta^i\,[a,S',A_1])) \\
= & [\![(u,\mathsf{lock}(a),u')]\!]_{\mathcal{T}}(\eta^i_{\mathsf{wc}}\,[u,S,A_0], \bigcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1} (\eta^i\,[\mathsf{initMT},\emptyset,A_1] \cup \\
 & \qquad \bigcup_{S' \subseteq \mathcal{M}} \eta^i\,[a,S',A_1]))
\end{array}
$$

where the first step exploits that the digest is ego-lane-derived, i.e., for all traces additionally admitted as arguments by only requiring $\mathsf{compat}^\sharp_{\mathcal{A}}$ instead of the stronger property, the right-hand side for the edge will yield the empty set. The second step exploits the fact that $[\![\cdot]\!]_{\mathcal{T}}$ is defined point-wise, and the last step uses the induction hypothesis. Consider now the second argument to $[\![\cdot]\!]_{\mathcal{T}}$.

$$\bigcup_{A_1 \in A, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1} (\eta^i\,[\mathsf{initMT},\emptyset,A_1] \cup \bigcup_{S' \subseteq \mathcal{M}} \eta^i\,[a,S',A_1])$$

For the first part, consider

$$\begin{aligned}
& \bigcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1} \eta^i\,[\mathsf{initMT}, \emptyset, A_1] \\
= \; & \bigcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1} \mathsf{uni}[\eta^i][\mathsf{initMT}, \emptyset, A_1] \\
= \; & \bigcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1} \{t' \mid t \in \bigcup_{u \in \mathcal{N}, A' \in \mathcal{A}, S \subseteq \mathcal{M}} \mathsf{split}[\eta^i][u, S, A'], \\
& \qquad\qquad \bar{t} = \mathsf{init_v}\, t, \bar{t} \neq \bot, t' = \downarrow_{\bar{t}} (t), (\alpha_{\mathcal{A}}\, t') = A_1\} \\
= \; & \bigcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1} \{t' \mid t \in \bigcup_{u \in \mathcal{N}, A' \in \mathcal{A}, S \subseteq \mathcal{M}} \eta^i_{\mathsf{wc}}[u, \emptyset, A'], \\
& \qquad\qquad \bar{t} = \mathsf{init_v}\, t, \bar{t} \neq \bot, t' = \downarrow_{\bar{t}} (t), (\alpha_{\mathcal{A}}\, t') = A_1\}
\end{aligned}$$

We remark that as $[\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}(t_0, t_1)$ is only defined for t_1 that end in $\mathsf{unlock}(a)$, or those t_1 that end in initMT and are a sub-trace of t_0, it suffices to consider these here.

$$\begin{aligned}
& \bigcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1} \{t' \mid t \in \bigcup_{u \in \mathcal{N}, A' \in \mathcal{A}, S' \subseteq \mathcal{M}} \eta^i_{\mathsf{wc}}[u, S', A'], \\
& \qquad\qquad \bar{t} = \mathsf{init_v}\, t, \bar{t} \neq \bot, t' = \downarrow_{\bar{t}} (t), (\alpha_{\mathcal{A}}\, t') = A_1\} \cap \\
& \{t' \mid t \in \eta^i_{\mathsf{wc}}\,[u, S, A_0], \bar{t} = \mathsf{init_v}\, t, \bar{t} \neq \bot, t' = \downarrow_{\bar{t}} (t), \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, (\alpha_{\mathcal{A}}\, t')\} \\
= \; & \{t' \mid t \in \bigcup_{u \in \mathcal{N}, A' \in \mathcal{A}, S' \subseteq \mathcal{M}} \eta^i_{\mathsf{wc}}[u, S', A'], \bar{t} = \mathsf{init_v}\, t, \bar{t} \neq \bot, t' = \downarrow_{\bar{t}} (t), \\
& \quad \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, (\alpha_{\mathcal{A}}\, t')\} \cap \\
& \{t' \mid t \in \eta^i_{\mathsf{wc}}\,[u, S, A_0], \bar{t} = \mathsf{init_v}\, t, \bar{t} \neq \bot, t' = \downarrow_{\bar{t}} (t), \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, (\alpha_{\mathcal{A}}\, t')\} \\
= \; & \{t' \mid t \in \eta^i_{\mathsf{wc}}\,[u, S, A_0], \bar{t} = \mathsf{init_v}\, t, \bar{t} \neq \bot, t' = \downarrow_{\bar{t}} (t), \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, (\alpha_{\mathcal{A}}\, t')\}
\end{aligned}$$

Thus, w.r.t. the resulting local traces where the second argument to $[\![\cdot]\!]_{\mathcal{T}}$ ends in initMT, the contributions of $\mathcal{C}$ and $\mathcal{C}_{\mathsf{wc}}$ are identical. For the second half of the second argument to $[\![\cdot]\!]_{\mathcal{T}}$, we have:

$$\begin{aligned}
& \bigcup_{A_1 \in A, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1} (\bigcup_{S' \subseteq \mathcal{M}} \eta^i\,[a, S', A_1]) \\
= \; & \bigcup_{A_1 \in A, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1} (\bigcup_{S' \subseteq \mathcal{M}} \bigcup_{g \in \mathcal{G}, w \subseteq \mathcal{M}} \eta^i_{\mathsf{wc}}[g, a, S', w, A_1]) \\
= \; & \bigcup \{\eta^i_{\mathsf{wc}}\,[g, a, S', w, A_1] \mid g \in \mathcal{G}, S' \subseteq \mathcal{M}, w \subseteq \mathcal{M}, A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\}
\end{aligned}$$

Thus, the contributions of $\mathcal{C}$ and $\mathcal{C}_{\mathsf{wc}}$ to the unknown $[u, S \cup \{a\}, A']$ are identical if property (1) holds for the i-th approximations (subproperty (a)). W.r.t. subproperty (e), the argument provided for the case of local actions also applies here. Thus, if property (1) holds for the i-th approximations, and constraints of this form are considered, it also holds for the $(i+1)$-th approximations.

Next, for constraints corresponding to **unlock**. Consider an edge $(u, \mathsf{unlock}(a), u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, \mathsf{unlock}(a)]\!]^\sharp_{\mathcal{A}}(A_0)$. For $\mathcal{C}_{\mathsf{wc}}$, the constraints take the following form:

$$[u', S \setminus \{a\}, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{unlock}(a), u'), A']\!]_{\mathsf{wc}}$$

with right-hand side

$$\begin{aligned}
&[\![([u, S, A_0], \mathsf{unlock}(a), u'), A']\!]_{\mathsf{wc}}\, \eta_{\mathsf{wc}} = \\
&\quad \textbf{let } T = [\![(u, \mathsf{unlock}(a), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{wc}}\, [u, S, A_0]) \textbf{ in} \\
&\quad \textbf{let } \rho = \{[g, a, S \setminus \{a\}, w, A'] \mapsto \{t\} \mid t \in T, g \in \mathcal{G}, w \subseteq \mathcal{M}, \\
&\qquad ((\mathsf{last_tl_write}_g\, t = (\bar{u}, g = x, \bar{u}') \wedge L_t[\bar{u}] \subseteq w) \vee (\mathsf{last_tl_write}_g\, t = \bot))\} \\
&\quad \textbf{in} \\
&\quad (\rho, T)
\end{aligned}$$

For $\mathcal{C}$, the constraints take the following form:

$$[u', S \setminus \{a\}, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{unlock}(a), [u', S \setminus \{a\}, A'])]\!]$$

with right-hand side

$$\begin{aligned}
&[\![([u, S, A_0], \mathsf{unlock}(a), [u', S \setminus \{a\}, A'])]\!]\, \eta = \\
&\quad \textbf{let } T = [\![(u, \mathsf{unlock}(a), u')]\!]_{\mathcal{T}}(\eta\, [u, S, A_0]) \textbf{ in} \\
&\quad (\{[a, S \setminus \{a\}, A'] \mapsto T\}, T)
\end{aligned}$$

Provided property (1) holds for the i-th approximations, the unknowns $[u', S \setminus \{a\}, A']$ of both constraint systems receive the same new contribution (subproperty (a)). W.r.t. subproperty (e), the argument provided for the case of local actions also applies here.

For the side-effects, consider the constraint system $\mathcal{C}$ and the new contribution $T = [\![(u, \mathsf{unlock}(a), u')]\!]_{\mathcal{T}}(\eta^i\, [u, S, A_0])$ to $[a, S \setminus \{a\}, A']$. Now consider some $g \in \mathcal{G}$, and $w \subseteq \mathcal{M}$. The new contribution to $\mathsf{split}[\eta^{i+1}][g, a, S \setminus \{a\}, w, A']$ is then given by

$$\begin{aligned}
&T \cap \{t \in \mathcal{T} \mid (\mathsf{last_tl_write}_g\, t = (\bar{u}, g = x, \bar{u}') \wedge L_t[\bar{u}'] \subseteq w) \vee (\mathsf{last_tl_write}_g\, t = \bot)\} \\
&= \{t \in T \mid (\mathsf{last_tl_write}_g\, t = (\bar{u}, g = x, \bar{u}') \wedge L_t[\bar{u}'] \subseteq w) \vee (\mathsf{last_tl_write}_g\, t = \bot)\}
\end{aligned}$$

which is the same set of traces $[g, a, S \setminus \{a\}, w, A']$ receives via side-effect in $\mathcal{C}_{\mathsf{wc}}$ (subproperty (b)). Thus, if property (1) holds for the i-th approximations, and constraints of this form are considered, it also holds for the $(i+1)$-th approximations.

Next, consider the constraints corresponding to **return**. Consider an edge $(u, \mathsf{return}, u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, \mathsf{return}]\!]^\sharp_{\mathcal{A}}(A_0)$. For $\mathcal{C}_{\mathsf{wc}}$, the constraints take the following form:

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{return}, u'), A']\!]_{\mathsf{wc}}$$

with right-hand side

$$\begin{aligned}
&[\![([u, S, A_0], \mathsf{return}, u'), A']\!]_{\mathsf{wc}}\, \eta_{\mathsf{wc}} = \\
&\quad \textbf{let } T = [\![(u, \mathsf{return}, u')]\!]_{\mathcal{T}}(\eta_{\mathsf{wc}}\, [u, S, A_0]) \textbf{ in} \\
&\quad \textbf{let } \rho = \{[i, A'] \mapsto \{t \mid t \in T, \mathsf{id}\, t = i\} \mid i \in \mathcal{V}_{\mathbf{tid}}\} \textbf{ in} \\
&\quad (\rho, T)
\end{aligned}$$

For $\mathcal{C}$, the constraints take the following form:

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{return}, [u', S, A'])]\!]$$

with right-hand side

$$\begin{aligned}[\![([u,S,A_0],\mathsf{return},[u',S,A'])]\!]\,\eta \quad &= \quad \textbf{let } T = [\![(u,\mathsf{return},u')]\!]_{\mathcal{T}}(\eta\,[u,S,A_0]) \textbf{ in} \\ &\quad\ (\{[\mathsf{return},S,A'] \mapsto T\},T)\end{aligned}$$

Provided property (1) holds for the i-th approximations, the unknowns $[u',S,A']$ of both constraint systems receive the same new contribution (subproperty (a)). W.r.t. subproperty (e), the argument provided for the case of local actions also applies here.

For the side-effects, consider the constraint system $\mathcal{C}$ and the new contribution $T = [\![(u,\mathsf{return},u')]\!]_{\mathcal{T}}(\eta^i\,[u,S,A_0])$ to $[\mathsf{return},S,A']$. Now consider some $i \in \mathcal{V}_{\mathbf{tid}}$. The new contribution to $\mathsf{split}[\eta^{i+1}][i,A']$ is then given by

$$T \cap \{t \in \mathcal{T} \mid \mathsf{id}\,t = i\} = \{t \in T \mid \mathsf{id}\,t = i\}$$

which is the same set of traces $[i,A']$ receives via side-effect in $\mathcal{C}_{\mathsf{wc}}$ (subproperty (c)). Thus, if property (1) holds for the i-th approximations, and constraints of this form are considered, it also holds for the $(i+1)$-th approximations.

Next, consider the constraints corresponding to **join**. Consider an edge $(u, x = \mathsf{join}(x'), u') \in \mathcal{E}$ and digests A', A_0, s.t. $\forall A_1 \in \mathcal{A}$, $[\![u, x = \mathsf{join}(x')]\!]^\sharp_{\mathcal{A}}(A_0,A_1) \in \{\{A'\},\emptyset\}$, which exists as $\mathcal{A}$ here is ego-lane-derived. For $\mathcal{C}_{\mathsf{wc}}$, the constraints take the following form:

$$[u',S,A'] \quad \supseteq \quad [\![([u,S,A_0],x = \mathsf{join}(x'),u')]\!]_{\mathsf{wc}}$$

with right-hand side

$$\begin{aligned}&[\![([u,S,A_0],x = \mathsf{join}(x'),u')]\!]_{\mathsf{wc}}\,\eta_{\mathsf{wc}} = \\ &\quad \textbf{let } T_1 = \bigcup\{\eta_{\mathsf{wc}}\,[t(x'),A_1] \mid t \in \eta_{\mathsf{wc}}\,[u,S,A_0], A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1\} \textbf{ in} \\ &\quad \textbf{let } T = [\![(u,x = \mathsf{join}(x'),u')]\!]_{\mathcal{T}}(\eta_{\mathsf{wc}}\,[u,S,A_0],T_1) \textbf{ in} \\ &\quad (\emptyset,T)\end{aligned}$$

For $\mathcal{C}$, the constraints for all digests A_1 where $[\![u, x = \mathsf{join}(x')]\!]^\sharp_{\mathcal{A}}(A_0,A_1) = \{A'\}$ take the following form,

$$[u',S,A'] \quad \supseteq \quad [\![([u,S,A_0],x = \mathsf{join}(x'),u'),A_1]\!]$$

with right-hand side

$$\begin{aligned}&[\![([u,S,A_0],x = \mathsf{join}(x'),u'),A_1]\!]\,\eta = \\ &\quad \textbf{let } T = \textstyle\bigcup_{S' \subseteq \mathcal{M}} [\![(u,x = \mathsf{join}(x'),u')]\!]_{\mathcal{T}}\,(\eta\,[u,S,A_0],\eta\,[\mathsf{return},S',A_1]) \textbf{ in} \\ &\quad (\emptyset,T)\end{aligned}$$

As a first step, we consider all contributions to $[u,S,A']$ caused by $\mathcal{C}$ together in order to

relate them to the contributions of $\mathcal{C}_{\mathsf{wc}}$. These are then given by

$$\begin{aligned}
&\bigcup_{A_1 \in \mathcal{A}, [\![u, x = \mathsf{join}(x')]\!]^\sharp_{\mathcal{A}}(A_0, A_1) = \{A'\}} \\
&\qquad (\bigcup_{S' \subseteq \mathcal{M}} [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}} \left(\eta^i\,[u, S, A_0], \eta^i\,[\mathsf{return}, S', A_1]\right)) \\
= \;& \bigcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1} (\bigcup_{S' \subseteq \mathcal{M}} [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}} \left(\eta^i\,[u, S, A_0], \eta^i\,[\mathsf{return}, S', A_1]\right)) \\
= \;& [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}} (\eta^i\,[u, S, A_0], \bigcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1} (\bigcup_{S' \subseteq \mathcal{M}} \eta^i\,[\mathsf{return}, S', A_1])) \\
= \;& [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}} (\eta^i_{\mathsf{wc}}\,[u, S, A_0], \bigcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1} (\bigcup_{S' \subseteq \mathcal{M}} \eta^i\,[\mathsf{return}, S', A_1]))
\end{aligned}$$

where the first step exploits that the digest is ego-lane-derived, i.e., for all traces additionally admitted as arguments by only requiring $\mathsf{compat}^\sharp_{\mathcal{A}}$ instead of the stronger property, the right-hand side for the edge will yield the empty set. The second step exploits the fact that $[\![\cdot]\!]_{\mathcal{T}}$ is defined point-wise, and the last step uses the induction hypothesis. Consider now the second argument to $[\![\cdot]\!]_{\mathcal{T}}$.

$$\bigcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1} (\bigcup_{S' \subseteq \mathcal{M}} \eta^i\,[\mathsf{return}, S', A_1])$$

We thus have

$$\begin{aligned}
& \bigcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1} (\bigcup_{S' \subseteq \mathcal{M}} \eta^i\,[\mathsf{return}, S', A_1]) \\
= \;& \bigcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1} (\bigcup_{S' \subseteq \mathcal{M}} \{t \in \bigcup_{j \in \mathcal{V}_{\mathsf{tid}}} \eta^i_{\mathsf{wc}}\,[j, A_1] \mid L_t = S'\} \\
= \;& \bigcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1} (\bigcup_{j \in \mathcal{V}_{\mathsf{tid}}} \eta^i_{\mathsf{wc}}\,[j, A_1])
\end{aligned}$$

By contrast, the second argument to $[\![\cdot]\!]_{\mathcal{T}}$ in $\mathcal{C}_{\mathsf{wc}}$ is given by

$$\bigcup \{\eta_{\mathsf{wc}}\,[t(x'), A_1] \mid t \in \eta_{\mathsf{wc}}\,[u, S, A_0], A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\}$$

It thus remains to show that excluding those traces from the second argument that have a thread *id* that does not appear as the value of variable x' at the sink of any of the traces in the first argument does not change the result. First, we remark that, by construction, $\forall t \in \eta^i_{\mathsf{wc}}\,[j, A_1] : \mathsf{id}\, t = j$. Consider now a local trace $t \in \eta^i_{\mathsf{wc}}\,[u, S, A_0]$ and some local trace $t' \in \eta^i_{\mathsf{wc}}\,[j, A_1]$ where $(u, \sigma) = \mathsf{sink}\, t$ and $\sigma\, x' \neq j$. Then $[\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(t, t') = \emptyset$ by the consistency condition on the join order $\rightarrow_j$ (see Section 2.5). We thus have

$$\begin{aligned}
& [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}} (\eta^i_{\mathsf{wc}}\,[u, S, A_0], \\
& \qquad \bigcup \{\eta_{\mathsf{wc}}\,[t(x'), A_1] \mid t \in \eta_{\mathsf{wc}}\,[u, S, A_0], A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\}) \\
= \;& [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}} (\eta^i_{\mathsf{wc}}\,[u, S, A_0], \bigcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1} (\bigcup_{j \in \mathcal{V}_{\mathsf{tid}}} \eta^i_{\mathsf{wc}}\,[j, A_1]))
\end{aligned}$$

Therefore, provided property (1) holds for the i-th approximations, the unknowns $[u', S, A']$ of both constraint systems receive the same new contribution (subproperty (a)). W.r.t. subproperty (e), the argument provided for the case of local actions also applies here. Thus, if property (1) holds for the i-th approximations, and constraints of this form are considered, it also holds for the $(i+1)$-th approximations.

Next, consider the constraints corresponding to **signal** for a condition variable s. Consider an edge $(u, \mathsf{signal}(s), u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, \mathsf{signal}(s)]\!]^\sharp_{\mathcal{A}}(A_0)$. For $\mathcal{C}_{\mathsf{wc}}$, the constraints take the following form:

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{signal}(s), u'), A']\!]_{\mathsf{wc}}$$

with right-hand side

$$\begin{array}{l}
[\![([u, S, A_0], \mathsf{signal}(s), u'), A']\!]_{\mathsf{wc}}\, \eta_{\mathsf{wc}} = \\
\quad \textbf{let } T = [\![(u, \mathsf{signal}(s), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{wc}}\, [u, S, A_0]) \textbf{ in} \\
\quad \textbf{let } \rho = \{[s, A'] \mapsto T\} \textbf{ in} \\
\quad (\rho, T)
\end{array}$$

For $\mathcal{C}$, the constraints take the following form:

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{signal}(s), [u', S, A'])]\!]$$

with right-hand side

$$\begin{array}{lcl}
[\![([u, S, A_0], \mathsf{signal}(s), [u', S, A'])]\!]\, \eta & = & \textbf{let } T = [\![(u, \mathsf{signal}(s), u')]\!]_{\mathcal{T}}(\eta\, [u, S, A_0]) \textbf{ in} \\
& & (\{[s, S, A'] \mapsto T\}, T)
\end{array}$$

Provided property (1) holds for the i-th approximations, the unknowns $[u', S, A']$ of both constraint systems receive the same new contribution (subproperty (a)). W.r.t. subproperty (e), the argument provided for the case of local actions also applies here.

For the side-effects, consider the constraint system $\mathcal{C}$ and the new contribution $T = [\![(u, \mathsf{signal}(s), u')]\!]_{\mathcal{T}}(\eta^i\, [u, S, A_0])$ to $[s, S, A']$. The new contribution to $\mathsf{split}[\eta^{i+1}][s, A']$ is then given by T which is the same set of traces $[s, A']$ receives via side-effect in $\mathcal{C}_{\mathsf{wc}}$ (subproperty (d)). Thus, if property (1) holds for the i-th approximations, and constraints of this form are considered, it also holds for the $(i+1)$-th approximations.

Lastly, consider the constraints corresponding to **wait**. Consider an edge $(u, \mathsf{wait}(s), u') \in \mathcal{E}$ and digests A', A_0, s.t. $\forall A_1 \in \mathcal{A}$, $[\![u, \mathsf{wait}(s)]\!]^\sharp_{\mathcal{A}}(A_0, A_1) \in \{\{A'\}, \emptyset\}$, which exists as $\mathcal{A}$ here is ego-lane-derived. For $\mathcal{C}_{\mathsf{wc}}$, the constraints take the following form:

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{wait}(s), u')]\!]_{\mathsf{wc}}$$

with right-hand side

$$\begin{array}{l}
[\![([u, S, A_0], \mathsf{wait}(s), u')]\!]_{\mathsf{wc}}\, \eta_{\mathsf{wc}} = \\
\quad \textbf{let } T_1 = \bigcup\{\eta_{\mathsf{wc}}\, [s, A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\} \textbf{ in} \\
\quad \textbf{let } T = [\![(u, \mathsf{wait}(s), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{wc}}\, [u, S, A_0], T_1) \textbf{ in} \\
\quad (\emptyset, T)
\end{array}$$

For $\mathcal{C}$, the constraints take the following form for all A_1 where $[\![u, \mathsf{wait}(s)]\!]^\sharp_{\mathcal{A}}(A_0, A_1) = \{A'\}$,

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{wait}(s), u'), A_1]\!]$$

with right-hand side

$$\begin{array}{l}[\![([u,S,A_0],\mathsf{wait}(s),u'),A_1]\!]\,\eta = \\ \quad \textbf{let } T = \bigcup_{S'\subseteq\mathcal{M}}[\![(u,\mathsf{wait}(s),u')]\!]_{\mathcal{T}}\,(\eta\,[u,S,A_0],\eta\,[s,S',A_1]) \textbf{ in} \\ \quad (\emptyset,T)\end{array}$$

As a first step, we consider all contributions to $[u,S,A']$ caused by $\mathcal{C}$ together in order to relate them to the contributions of $\mathcal{C}_{\mathsf{wc}}$. These are then given by

$$\begin{array}{rl} & \bigcup_{A_1\in\mathcal{A},[\![u,\mathsf{wait}(s)]\!]^\sharp_{\mathcal{A}}(A_0,A_1)=\{A'\}}(\bigcup_{S'\subseteq\mathcal{M}}[\![(u,\mathsf{wait}(s),u')]\!]_{\mathcal{T}}\,(\eta^i\,[u,S,A_0],\eta^i\,[s,S',A_1])) \\ = & \bigcup_{A_1\in\mathcal{A},\mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1}(\bigcup_{S'\subseteq\mathcal{M}}[\![(u,\mathsf{wait}(s),u')]\!]_{\mathcal{T}}\,(\eta^i\,[u,S,A_0],\eta^i\,[s,S',A_1])) \\ = & [\![(u,\mathsf{wait}(s),u')]\!]_{\mathcal{T}}(\eta^i\,[u,S,A_0],\bigcup_{A_1\in\mathcal{A},\mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1}(\bigcup_{S'\subseteq\mathcal{M}}\eta^i\,[s,S',A_1])) \\ = & [\![(u,\mathsf{wait}(s),u')]\!]_{\mathcal{T}}(\eta^i_{\mathsf{wc}}\,[u,S,A_0],\bigcup_{A_1\in\mathcal{A},\mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1}(\bigcup_{S'\subseteq\mathcal{M}}\eta^i\,[s,S',A_1])) \end{array}$$

where the first step exploits that the digest is ego-lane-derived, i.e., for all traces additionally admitted as arguments by only requiring $\mathsf{compat}^\sharp_{\mathcal{A}}$ instead of the stronger property, the right-hand side for the edge will yield the empty set. The second step exploits the fact that $[\![\cdot]\!]_{\mathcal{T}}$ is defined point-wise, and the last step uses the induction hypothesis. Consider now the second argument to $[\![\cdot]\!]_{\mathcal{T}}$.

$$\bigcup_{A_1\in\mathcal{A},\mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1}(\bigcup_{S'\subseteq\mathcal{M}}\eta^i\,[s,S',A_1])$$

We thus have

$$\begin{array}{rl} & \bigcup_{A_1\in\mathcal{A},\mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1}(\bigcup_{S'\subseteq\mathcal{M}}\eta^i\,[s,S',A_1]) \\ = & \bigcup_{A_1\in\mathcal{A},\mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1}(\bigcup_{S'\subseteq\mathcal{M}}\{t\in\eta^i_{\mathsf{wc}}\,[s,A_1]\mid L_t=S'\} \\ = & \bigcup_{A_1\in\mathcal{A},\mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1}(\eta^i_{\mathsf{wc}}\,[s,A_1]) \end{array}$$

which is the second argument to $[\![\cdot]\!]_{\mathcal{T}}$ in $\mathcal{C}_{\mathsf{wc}}$. Therefore, provided property (1) holds for the i-th approximations, the unknowns $[u',S,A']$ of both constraint systems receive the same new contribution (subproperty (a)). W.r.t. subproperty (e), the argument provided for the case of local actions also applies here. Thus, if property (1) holds for the i-th approximations, and constraints of this form are considered, it also holds for the $(i+1)$-th approximations.

Therefore, by induction hypothesis, if property (1) holds for the i-th approximations, and any of the constraints are considered property (1) also holds for the $(i+1)$-th approximation. This concludes the case distinction and the proof of Proposition 16. □

The next proposition needed for the soundness proof of Write-Centered Reading indicates that the new unknown $[g,a,S,w,A]$ collects a superset of local traces whose last write to the global g can be read by a thread satisfying the specific assumptions (W0) through (W5) below.

Proposition 17. *Consider the i-th approximation η^i_{wc} to the least solution η_{wc} of constraint system $\mathcal{C}_{wc}$, a control-flow edge $(u, x = g, u')$ of the program, and a local trace $t \in \eta^i\,[u', S, A]$ in which the last action is $x = g$, that ends in $\bar{u}' = (j, u', \sigma)$, i.e., $t = \downarrow_{\bar{u}'}\ (t)$. Let $P = \mathsf{min\_lockset\_since}\, t\, \bar{v}'$ denote the upwards-closed set of minimal locksets held by the ego thread since the endpoint $\bar{v}'$ of its last thread-local write to g, or $\{\emptyset\}$ if there is no thread-local write to g in t.*

Then, the value $d = \sigma\, x$ that is read for g,

- *is the initial value 0 of g; or*
- *is produced by a write to g that is the last thread-local write to g in t; or*
- *is produced by a write to g that is the last* thread-local *write to g in some local trace stored at $\eta^{i'}_{wc}\,[g, a, S', w', A']$ for some $i' < i$ i.e.,*

$$d \in \mathsf{eval\_tl}_g\,(\eta^{i'}_{wc}\,[g, a, S', w', A'])$$

where

(W0) $a \in S$,

(W1) $w' \subseteq \mathcal{M}$,

(W2) $S \cap S' = \emptyset$,

(W3) $\exists S'' \in P : S'' \cap w' = \emptyset$,

(W4) $\exists S''' \in P : a \notin S'''$, *and*

(W5) $\mathsf{compat}^\sharp_{\mathcal{A}}\ A\ A'$

Proof. The proof is by fixpoint induction: We prove that the values (that are not the initial values of globals) read non-thread-locally for a global g at some $(u, x = g, u')$ during the computation of η^i_{wc} are the last *thread-local* writes of a local trace t' ending in an unlock operation that is side-effected to an appropriate $\eta^{i'}_{\mathsf{wc}}\,[g, a, S', w', A']$ in some prior iteration $i' < i$ for some a, S', w', and A' satisfying (W0) through (W5).

This property holds for $i = 0$, as in η^0_{wc}, all unknowns have the value $\emptyset$, and, therefore, no reads from globals or unlocks can happen.

For the induction step $i > 0$, there are two proof obligations: First, that the property holds for all reads from a global, and, additionally, that all traces ending in an unlock operation are once more side-effected to appropriate unknowns in this iteration.

For the first obligation, consider a local trace $t \in \eta^i_{\mathsf{wc}}\,[u', S, A]$ where the last action is $x = g$ and the result of $\mathsf{last\_write}_g\, t$. Recall that $\mathsf{last\_write}_g\, t$ returns either the edge along which the last write to g happened or the edge corresponding to initMT in case g has not been written. In the latter case, the value read for g is the initial value 0 of g, and the proposition holds. Now consider the case where there is a write to g in t:

$$\mathsf{last\_write}_g\, t = ((j' - 1, u_{j'-1}, \sigma_{j'-1}), g = x', \bar{u}'') = l.$$

Let $i_0 = \mathsf{id}\, t$ and $i_1 = \sigma_{j-1}\, \mathsf{self}$ the thread *id*s of the reading ego thread and the thread performing the last write, respectively. We distinguish two cases:
Case 1: $i_0 = i_1$. The last write is thread-local to t (and l is therefore also the last thread-local write to g in t) and the proposition holds.
Case 2: $i_0 \neq i_1$. The last write is not thread-local. Let w_l the set of locks held on that last write to g, i.e., $L_t[\bar{u}'']$. Consider the maximal sub-trace t' of t with $\mathsf{id}(t') = i_1$ so that $\mathsf{last}(t')$ unlocks some mutex in S. Let this mutex be a. Such a sub-trace must exist since accessing g is necessarily protected by m_g. Let S' denote the background lockset held at this last action in t', and let $A' = \alpha_{\mathcal{A}'}\, t'$. t' was first produced during some earlier iteration $i' < i$. By induction hypothesis, we may assume this t' was side-effected to $\eta^{i'}_{\mathsf{wc}}[g, a, S', w'', A']$ during the i'-th iteration, for all $w'' \supseteq w_l$. Therefore, the read value d is given by

$$d = \sigma_{j-1}\, x' \in \mathsf{eval\_tl}_g\, (\eta^{i'}_{\mathsf{wc}}\, [g, a, S', w', A']) \subseteq \mathsf{eval\_tl}_g\, (\eta^{i}_{\mathsf{wc}}\, [g, a, S', w', A'])$$

It remains to prove that then the conditions hold for a, S', w', A':

(W0) $a \in S$ (by construction above)

(W1) $w' = w_l \subseteq \mathcal{M}$ (by construction of the constraint for edges with unlock operations in $\mathcal{C}_{\mathsf{wc}}$)

(W2) $S \cap S' = \emptyset$
Assume that this were not the case, i.e., $\exists c \in \mathcal{M} : c \in S \cap S'$. Then thread i_1 holds the lock of mutex c at the sink of every super-trace t'' of t' with $\mathsf{id}(t'') = i_1$ that is still a sub-trace of t. By construction, we know c is not released in any t''. Thus, thread i_0 cannot acquire c — which would be necessary to hold S at the sink of t. Contradiction.

(W3) $\exists S'' \in P\, g,\, S'' \cap w' = \emptyset$
Let $l' = \mathsf{last\_tl\_write}_g\, t$ the last thread-local write to g in t. If there is no last thread-local write, i.e., $l' = \bot$, then $P\, g = \{\emptyset\}$, and the condition holds. Otherwise, assume for a contradiction that i_0 has always maintained a non-empty lockset intersection with w' since l', i.e., since action l' thread i_0 has at each point held one of the locks held when the write l was performed. Then l can not have happened after l', and l cannot be the last write to g in t.

(W4) $\exists S''' \in P\, g,\, a \notin S'''$
Let $l' = \mathsf{last\_tl\_write}_g\, t$ the last thread-local write to g in t. If there is no last thread-local write, i.e., $l' = \bot$, then $P\, g = \{\emptyset\}$, and the condition holds. Otherwise, assume for a contradiction that $a \in S'''$ for all $S''' \in P\, g$. Since a is unlocked by i_1 after l, l can not have happened after l' and l can not be the last write to g in t.

(W5) $\mathsf{compat}^{\sharp}_{\mathcal{A}}\, A\, A'$
We distinguish two cases: First, the case in which the local trace t' is incorporated

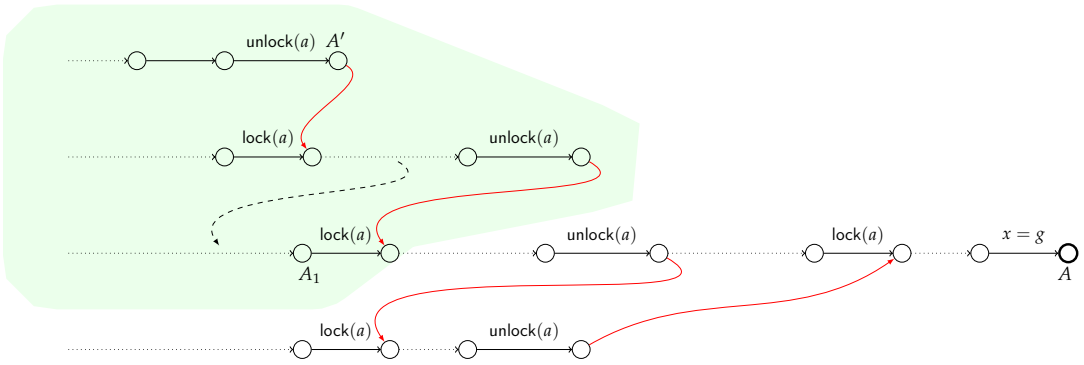


Figure 6.1: Illustration for the case (W5) where t' is not directly incorporated into the creation-extended ego lane of i_0, where the red edges represent $\rightarrow_a$, the dashed edge represents some other dependency, and the area shaded in green corresponds to t''.

into the *creation-extended ego lane* of i_0 at some operation $\mathsf{lock}(a)$ with digest information A_0. Then $\mathsf{compat}^\sharp_{\mathcal{A}}\ A_0\ A'$ and by repeated application of *Eq.* (2.17) therefore also $\mathsf{compat}^\sharp_{\mathcal{A}}\ A\ A'$.

Now consider the other case, i.e., t' is not directly incorporated into the *creation-extended ego lane* of i_0. We remark that, as lock and unlock operations are totally ordered, there is a *first* operation $\mathsf{lock}(a)$ along the *creation-extended ego lane* of i_0, at which a supertrace of t' is incorporated. Let us refer to the subtrace ending in this $\mathsf{lock}(a)$ in the *creation-extended ego lane* of i_0 by t'' and now denote the digest information for the predecessor node $\bar{u}_1$ of the lock by A_1. An illustration of this situation with t'' highlighted is given in Fig. 6.1.

The idea now is to construct a local trace of a modified program such that this lock operation becomes the direct (instead of only indirect) successor of the $\mathsf{unlock}(a)$ in t'. Here, we exploit that, instead of giving a grammar, we defined our programs in terms of control-flow graphs and did not require these control-flow graphs to have at most two successors per program point. Thus, it is possible to modify a program in a way that the digest associated with some program point remains the same, but the program behavior after that program point may change, by adding new edges that introduce non-determinism.

For now, consider the case where $a \not\equiv m_g$. We now replace all uses of a succeeding the $\mathsf{unlock}(a)$ in t' with uses of a fresh mutex $\bar{a}$, except for the last $\mathsf{lock}(a)$ in t'' which remains unmodified and receives a new successor $\mathsf{lock}(\bar{a})$. Additionally, after the $\mathsf{unlock}(a)$ in t' we insert new operations $\mathsf{lock}(\bar{a})$ and $\mathsf{unlock}(\bar{a})$ and adapt the mutex order $\rightarrow_{\bar{a}}$ to reflect that this $\mathsf{lock}(\bar{a})$ is the first lock operation of $\bar{a}$ and that the mutex order for the renamed mutex a follows the original order for a, except that the last lock of $\bar{a}$ is the newly inserted node. Then, directly connect the

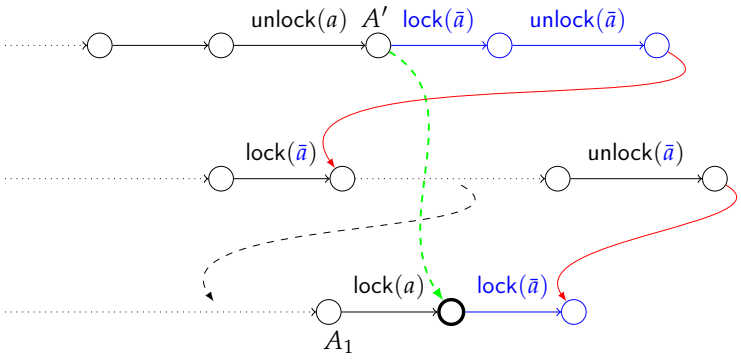


Figure 6.2: Example trace t'' from Fig. 6.1 adapted so it is a local trace of a modified program where t' (with digest A') is directly incorporated into the ego lane of i_0 at an edge with predecessor digest A_1. Modification from the original t'' are given in blue. The red edges now represent $\rightarrow_{\bar{a}}$, the dashed line still represents some other dependency, and the green dashed edge is the new direct dependency for $\rightarrow_a$.

$\mathsf{unlock}(a)$ in t' to the last $\mathsf{lock}(a)$ in t''. This construction is exemplified in Fig. 6.2.

The resulting local trace is a valid local trace of some program (which can be extracted from the local trace), and we have that the digest for $\bar{u}_1$ in the new local trace is still A_1, as the digests are ego-lane-derived, and the modification does not affect the creation-extended ego lane of i_0 up to this point. Similarly, the digest associated with the $\mathsf{unlock}(a)$ in t' is still A'. As the operation $\mathsf{lock}(a)$ now directly incorporates t', we have $[\![(u, \mathsf{lock}(a), u_1)]\!]^\sharp_{\mathcal{A}}(A_1, A') \neq \emptyset$, and thus $\mathsf{compat}^\sharp_{\mathcal{A}}\, A_1\, A'$.

Now consider again the unmodified local trace: As $\mathsf{compat}^\sharp_{\mathcal{A}}\, A_1\, A'$, and $\mathcal{A}$ is ego-lane-derived, we also have $\mathsf{compat}^\sharp_{\mathcal{A}}\, A\, A'$.

For the case where $a \equiv m_g$, the argument proceeds similarly, with the difference that further modifications are needed to ensure m_g still surrounds all accesses to g. As the last write to g occurs in t', there are no further write operations between this write and the currently considered read. Also, there are no further reads along the creation-extended ego lane of i_0 before the $\mathsf{lock}(m_g)$ in which t'' ends. All reads $x = g$ ordered between these are then replaced by actions $x =?$, while the successor state remains unmodified. This corresponds to randomly choosing the value that g has in the unmodified program. With this additional modification, a local trace as constructed above with the random value set to the value that g would have, once again is a valid local trace of some program (which can be read off the local trace), and m_g still directly surrounds all accesses to g. Also, the digest A' associated with t' and the digest A_1 associated with $\bar{u}_1$ remain unmodified as the digest is ego-lane-derived. The operation $\mathsf{lock}(m_g)$ now directly incorporates t', and we have $[\![(u, \mathsf{lock}(m_g), u_1)]\!]^\sharp_{\mathcal{A}}(A_1, A') \neq \emptyset$, and thus $\mathsf{compat}^\sharp_{\mathcal{A}}\, A_1\, A'$, and —

by the same argument as in the other case — also $\mathsf{compat}^\sharp_{\mathcal{A}}\ A\ A'$.

It now remains to show that any trace t with $\mathsf{last}(t) = \mathsf{unlock}(a)$, $a \in \mathcal{M}$ ending in (j, u, σ), i.e., $t = (j, u, \sigma)\downarrow_t$, produced in this iteration i is side-effected to the unknowns $\eta^i_{\mathsf{wc}}[g, a, S, w, \alpha_{\mathcal{A}}(t)]$ for $S = L_t[(j, u, \sigma)]$ and $w'_l \subseteq w \subseteq \mathcal{M}$, where w'_l is the set of mutexes held when writing to g for the last time thread-locally in t, if g was written to at all. This, however, follows directly from the construction of $\mathcal{C}_{\mathsf{wc}}$. □

Our goal is to relate solutions of the constraint systems $\mathcal{C}_{\mathsf{wc}}$ and $\mathcal{C}^\sharp_{\mathsf{wc}}$ to each other. While the sets of unknowns of these two systems (except for the unknowns related to thread returns, which depend on the abstraction) are the same, the side-effects to unknowns are still not fully comparable. Therefore, we modify the side-effects produced by $\mathcal{C}^\sharp_{\mathsf{wc}}$ for unlock operations to obtain yet another constraint system $\mathcal{C}^{\sharp\prime}_{\mathsf{wc}}$, which we abbreviate by $\mathcal{C}^\sharp$ in this section. All right-hand-side functions remain the same except for $\mathsf{unlock}(a)$ which is now given by:

$$\begin{aligned}
[\![[u, S, A_0], \mathsf{unlock}(a), A']\!]^\sharp\eta \quad = \quad & \textbf{let } (W, P, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
& \textbf{let } P' = \{g \mapsto P\,g \sqcup \{S \setminus \{a\}\} \mid g \in \mathcal{G}\} \textbf{ in} \\
& \textbf{let } \rho = \{[g, a, S \setminus \{a\}, w, A'] \mapsto \sigma\,g \\
& \qquad \mid g \in \mathcal{G}, w' \in W\,g, w' \subseteq w\} \textbf{ in} \\
& (\rho, (W, P', \sigma))
\end{aligned}$$

Instead of only side-effecting to *minimal* sets w' of locks held on a write to g, the value now is side-effected to *all* supersets w of such minimal elements. This modification of the constraint system only changes the values computed for unknowns associated with global variables, but not those for all other unknowns: Upon reading, all $[g, a, S, w, A]$ are consulted where there is an empty intersection of w and some $P\,g$. If this is the case for w, it also holds for $w' \subseteq w$. Accordingly, the values additionally published to $[g, a, S, w, A]$ because of the modified side-effects, are already read from $[g, a, S, w', A]$ directly in $\mathcal{C}^\sharp_{\mathsf{wc}}$. More formally, for $\eta^\sharp_{\mathsf{wc}}$ a solution of $\mathcal{C}^\sharp_{\mathsf{wc}}$, define $\eta^\sharp$ by

$$\begin{aligned}
\eta^\sharp[u, S, A] &= \eta^\sharp_{\mathsf{wc}}[u, S, A] && (\text{for } u \in \mathcal{N}, S \subseteq \mathcal{M}, A \in \mathcal{A}) \\
\eta^\sharp[g, a, S, w, A] &= \bigsqcup\{\eta^\sharp_{\mathsf{wc}}[g, a, S, w', A] \mid w' \subseteq w\} && (\text{for } g \in \mathcal{G}, a \in \mathcal{M}, S \subseteq \mathcal{M}, \\
& && \quad w \subseteq \mathcal{M}, A \in \mathcal{A}) \\
\eta^\sharp[i, A] &= \eta^\sharp_{\mathsf{wc}}[i, A] && (\text{for } i \in S_{\mathcal{V}^\sharp_{\mathbf{tid}}}, A \in \mathcal{A}) \\
\eta^\sharp[s, A] &= \eta^\sharp_{\mathsf{wc}}[s, A] && (\text{for } s \in \mathcal{S}, A \in \mathcal{A})
\end{aligned}$$

Then, we have:

Proposition 18. *$\eta^\sharp$ as constructed above is a solution of $\mathcal{C}^\sharp$.*

Proof. The proof of Proposition 18 is by verifying for each edge (u, act, v) of the control-flow graph, each possible lockset S and appropriate digests, that the constraint is satisfied. This is of particular interest for constraints corresponding to edges $(u, \mathsf{unlock}(a), u')$

that cause additional side-effects in $\mathcal{C}^\sharp$, and for constraints corresponding to edges $(u, x = g, u')$ where the values of these unknowns receiving novel side-effects are accessed. For the additional side-effects of $\mathsf{unlock}(a)$, we easily verify that the additional side-effects in $\mathcal{C}^\sharp$ are accounted for by the construction of $\eta^\sharp$.

Now, for an edge $(u, x = g, u')$ and $A_0, A' \in \mathcal{A}$ where $[\![u, x = g]\!]^\sharp_{\mathcal{A}}(A_0) = \{A_1\}$, and lockset S:

$$[u', S, A_1] \quad \sqsupseteq \quad [\![[u, S, A_0], x = g]\!]^\sharp$$

where

$$\begin{array}{lcl}
[\![[u, S, A_0], x = g]\!]^\sharp \eta^\sharp & = & \textbf{let } (W, P, \sigma) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\
& & \textbf{let } d = \sigma\, g \sqcup \bigsqcup\{\eta^\sharp[g, a, S', w, A'] \mid a \in S, S \cap S' = \emptyset, \\
& & \qquad \exists S'' \in P\, g : S'' \cap w = \emptyset, \\
& & \qquad \exists S''' \in P\, g : a \notin S''', \\
& & \qquad A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \textbf{ in} \\
& & (\emptyset, (W, P, \sigma \oplus \{x \mapsto d\})) \\
[\![[u, S, A_0], x = g]\!]^\sharp_{\mathsf{wc}} \eta^\sharp_{\mathsf{wc}} & = & \textbf{let } (W, P, \sigma) = \eta^\sharp_{\mathsf{wc}}\,[u, S, A_0] \textbf{ in} \\
& & \textbf{let } d = \sigma\, g \sqcup \bigsqcup\{\eta^\sharp_{\mathsf{wc}}[g, a, S', w, A'] \mid a \in S, S \cap S' = \emptyset, \\
& & \qquad \exists S'' \in P\, g : S'' \cap w = \emptyset, \\
& & \qquad \exists S''' \in P\, g : a \notin S''', \\
& & \qquad A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \textbf{ in} \\
& & (\emptyset, (W, P, \sigma \oplus \{x \mapsto d\}))
\end{array}$$

As $\eta^\sharp\,[u, S, A_0] = \eta^\sharp_{\mathsf{wc}}\,[u, S, A_0]$, and $\eta^\sharp[g, a, S, w, A] = \bigsqcup\{\eta^\sharp_{\mathsf{wc}}\,[g, a, S, w', A] \mid w' \subseteq w\}$ it thus suffices to show

$$\begin{array}{lll}
\bigsqcup\{\bigsqcup\{ & \eta^\sharp_{\mathsf{wc}}\,[g, a, S, w', A] \mid w' \subseteq w\} & \mid a \in S, S \cap S' = \emptyset, \exists S'' \in P\, g : S'' \cap w = \emptyset, \\
& & \exists S''' \in P\, g : a \notin S''', A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \\
\sqsubseteq \bigsqcup\{ & \eta^\sharp_{\mathsf{wc}}[g, a, S', w, A'] & \mid a \in S, S \cap S' = \emptyset, \exists S'' \in P\, g : S'' \cap w = \emptyset, \\
& & \exists S''' \in P\, g : a \notin S''', A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\}
\end{array}$$

which holds as in case $w \cap S'' = \emptyset$ for any $w' \subseteq w$, $w' \cap S'' = \emptyset$ also holds and the value of the corresponding unknown is thus included in the second least upper bound as well. As a consequence, the new contribution to $\eta^\sharp\,[u', S, A_1]$ for this constraint is subsumed by the contribution in $\mathcal{C}^\sharp_{\mathsf{wc}}$, which in turn is subsumed by $\eta^\sharp_{\mathsf{wc}}\,[u', S, A_1]$ as $\eta^\sharp_{\mathsf{wc}}$ is a solution of $\mathcal{C}^\sharp_{\mathsf{wc}}$. As $\eta^\sharp\,[u', S, A_1] = \eta^\sharp_{\mathsf{wc}}\,[u', S, A_1]$ and this constraint does not cause any side-effects, it is satisfied. □

It thus remains to relate solutions of $\mathcal{C}_{\mathsf{wc}}$ and solutions of $\mathcal{C}^{\sharp\prime}_{\mathsf{wc}}$ to each other. As a first step, we define a function β that extracts from a local trace t for each global g the minimal lockset $W\,g$ held at the last *thread-local* write to g, as well as all minimal locksets $P\,g$ since the last *thread-local* write to g. Additionally, it extracts a map σ that contains the values of the locals at the sink of t as well as the last-written thread-local values of

globals. Thus, we define

$$\begin{array}{lcl}
\beta\, t & = & (W, P, \sigma) \qquad \text{where} \\
W & = & \{g \mapsto \{L_t[\bar{u}']\} \mid g \in \mathcal{G}, (_, g = x, \bar{u}') = \mathsf{last_tl_write}_g\, t\} \\
 & & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t\} \\
P & = & \{g \mapsto \mathsf{min_lockset_since}\, t\, \bar{u}' \mid g \in \mathcal{G}, (_, g = x, \bar{u}') = \mathsf{last_tl_write}_g\, t\} \\
 & & \cup\, \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t\} \\
\sigma & = & \{x \mapsto \{t(x)\} \mid x \in \mathcal{X}\} \\
 & & \cup\, \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\, t\} \\
 & & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t, \bot \neq \mathsf{last_write}_g\, t\} \\
 & & \cup\, \{g \mapsto \{\sigma_{j-1}\, x\} \mid g \in \mathcal{G}, \\
 & & \qquad ((j-1, u_{j-1}, \sigma_{j-1}), g = x, _) = \mathsf{last_tl_write}_g\, t\}
\end{array} \tag{6.1}$$

The abstraction function β is used to specify concretization functions for the values of unknowns $[u, S, A]$ for program points, currently held locksets, and digests as well as for the other unknowns.

$$\begin{array}{lcl}
\gamma_{u,S,A}(P^\sharp, W^\sharp, \sigma^\sharp) & = & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u, L_t = S, \alpha_{\mathcal{A}}\, t = A, \beta\, t = (W, P, \sigma), \\
 & & \qquad \sigma \subseteq \bar{\gamma}_{\bar{\mathcal{V}}^\sharp} \circ \sigma^\sharp, W \sqsubseteq W^\sharp, P \sqsubseteq P^\sharp\}
\end{array}$$

where $\subseteq$ on maps is understood to be defined point-wise, and $\sqsubseteq$ for W and P is defined point-wise as well. We will also subsequently abbreviate $\bar{\gamma}_{\bar{\mathcal{V}}^\sharp}$ by $\bar{\gamma}$. Furthermore, we have

$$\begin{array}{lcl}
\gamma_{g,a,S,w,A}(v) & = & \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{unlock}(a), L_t = S, \alpha_{\mathcal{A}}\, t = A, \\
 & & \qquad (_, _, \sigma_{j-1}), g = x, \bar{u}') = \mathsf{last_tl_write}_g\, t, \\
 & & \qquad \sigma_{j-1}\, x \in \gamma_{\mathcal{V}^\sharp}(v), w \subseteq L_t[\bar{u}']\} \\
 & & \cup\, \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{unlock}(a), L_t = S, \alpha_{\mathcal{A}}\, t = A, \\
 & & \qquad \mathsf{last_tl_write}_g\, t = \bot\} \\
\gamma_{i,A}(v) & = & \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{return}, \alpha_{\mathcal{A}}\, t = A, \mathsf{id}\, t = i, t(\mathsf{ret}) \in \gamma_{\mathcal{V}^\sharp}(v)\} \\
\gamma_{s,A}(P^\sharp, W^\sharp, \sigma^\sharp) & = & \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{signal}(s), \alpha_{\mathcal{A}}\, t = A, \beta\, t = (W, P, \sigma), \\
 & & \qquad \sigma \subseteq \bar{\gamma} \circ \sigma^\sharp, W \sqsubseteq W^\sharp, P \sqsubseteq P^\sharp\}
\end{array}$$

Let $\eta^\sharp$ be a solution of $\mathcal{C}^\sharp$. We then construct from it a mapping η'_{wc} by:

$$\begin{array}{lcll}
\eta'_{\mathsf{wc}}[u, S, A] & = & \gamma_{u,S,A}(\eta^\sharp\,[u, S, A]) & u \in \mathcal{N}, S \subseteq \mathcal{M}, A \in \mathcal{A} \\
\eta'_{\mathsf{wc}}[g, a, S, w, A] & = & \gamma_{g,a,S,w,A}(\eta^\sharp\,[g, a, S, w, A]) & g \in \mathcal{G}, a \in \mathcal{M}, S \subseteq \mathcal{M}, \\
 & & & \qquad w \subseteq \mathcal{M}, A \in \mathcal{A} \\
\eta'_{\mathsf{wc}}[i, A] & = & \bigcup\{\gamma_{i,A}(\eta^\sharp\,[i^\sharp, A]) \mid & i \in \mathcal{V}_{\mathbf{tid}}, A \in \mathcal{A} \\
 & & \qquad i^\sharp \in S_{\mathcal{V}^\sharp_{\mathbf{tid}}}, i \in (\gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}\, \{i^\sharp\})\} & \\
\eta'_{\mathsf{wc}}[s, A] & = & \gamma_{s,A}(\eta^\sharp\,[s, A]) & s \in \mathcal{S}, A \in \mathcal{A}
\end{array}$$

Altogether, correctness of $\mathcal{C}^\sharp_{\mathsf{wc}}$ follows from the following theorem:

Theorem 17. *Every solution of $\mathcal{C}^\sharp_{\mathsf{wc}}$ is sound w.r.t. the local trace semantics.*

Proof. Recall from Proposition 16, that the least solution of $\mathcal{C}_{\mathsf{wc}}$ is sound w.r.t. the local trace semantics as specified by the constraint system $\mathcal{C}$. By Proposition 18, which relates solutions of $\mathcal{C}^\sharp_{\mathsf{wc}}$ to solutions of $\mathcal{C}^\sharp$ (which in this section is an abbreviation for $\mathcal{C}^{\sharp\prime}_{\mathsf{wc}}$), it thus suffices to prove that the mapping η'_{wc} as constructed from a solution of $\mathcal{C}^\sharp$ above, is a solution of the constraint system $\mathcal{C}_{\mathsf{wc}}$. For that, we verify by fixpoint induction that for the i-th approximation η^i to the least solution η of $\mathcal{C}_{\mathsf{wc}}$, $\eta^i \subseteq \eta'_{\mathsf{wc}}$ holds.

To this end, we first consider the constraints for **initialization**, the start point u_0 and the empty lockset. We verify that for all $A \in \mathsf{init}^\sharp_{\mathcal{A}}$:

$$(\emptyset, \{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\}) \quad \subseteq \quad (\eta'_{\mathsf{wc}}, \eta'_{\mathsf{wc}}\,[u_0, \emptyset, A])$$

As no side-effects are triggered, it suffices to check that $\{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\} \subseteq \eta'_{\mathsf{wc}}\,[u_0, \emptyset, A]$ holds.

$$\begin{array}{rcl}
\mathsf{init}(A)^\sharp\,_ & = & \textbf{let } W^\sharp = \{g \mapsto \emptyset \mid g \in \mathcal{G}\} \textbf{ in} \\
& & \textbf{let } P^\sharp = \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}\} \textbf{ in} \\
& & \textbf{let } \sigma^\sharp = \{x \mapsto \top \mid x \in \mathcal{X} \setminus \{\mathsf{self}\}\} \cup \{\mathsf{self} \mapsto [\![i_0]\!]^\sharp_{\mathcal{E}xp}\,\top\} \\
& & \qquad\qquad \cup \{g \mapsto [\![0]\!]^\sharp_{\mathcal{E}xp}\,\top \mid g \in \mathcal{G}\} \\
& & \textbf{in} \\
& & (\emptyset, (W^\sharp, P^\sharp, \sigma^\sharp))
\end{array}$$

Let $\eta^\sharp\,[u_0, \emptyset, A] = (W^{\sharp\prime}, P^{\sharp\prime}, \sigma^{\sharp\prime})$ the value provided by $\eta^\sharp$ for the start point and the empty lockset. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $W^\sharp \sqsubseteq W^{\sharp\prime}$, $P^\sharp \sqsubseteq P^{\sharp\prime}$, and $\sigma^\sharp \sqsubseteq \sigma^{\sharp\prime}$ all hold. Then, by definition:

$$\begin{array}{rcl}
 & & \eta'_{\mathsf{wc}}[u_0, \emptyset, A] = \gamma_{u_0,\emptyset,A}(W^{\sharp\prime}, P^{\sharp\prime}, \sigma^{\sharp\prime}) \\
& = & \{t \in \mathcal{T} \mid \mathsf{loc}\,t = u_0, L_t = \emptyset, \alpha_{\mathcal{A}}\,t = A, \beta\,t = (W, P, \sigma), \\
& & \qquad \sigma \subseteq \bar{\gamma} \circ \sigma^{\sharp\prime}, W \sqsubseteq W^{\sharp\prime}, P \sqsubseteq P^{\sharp\prime}\}
\end{array}$$

For every trace $t \in \{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\}$, let

$$\begin{array}{rcl}
\beta\,t & = & (W, P, \sigma) \text{ where} \\
W & = & \{g \mapsto \{L_t[\bar{u}']\} \mid g \in \mathcal{G}, (_, g = x, \bar{u}') = \mathsf{last_tl_write}_g\,t\} \\
& & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\,t\} \\
& = & \{g \mapsto \emptyset \mid g \in \mathcal{G}\} \\
P & = & \{g \mapsto \mathsf{min_lockset_since}\,t\,\bar{u}' \mid g \in \mathcal{G}, (_, g = x, \bar{u}') = \mathsf{last_tl_write}_g\,t\} \\
& & \cup\, \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\,t\} \\
& = & \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}\} \\
\sigma & = & \{x \mapsto \{t(x)\} \mid x \in \mathcal{X}\} \\
& & \cup\, \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\,t\} \\
& & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\,t, \bot \neq \mathsf{last_write}_g\,t\} \\
& & \cup\, \{g \mapsto \{\sigma_{j-1}\,x\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x, _) = \mathsf{last_tl_write}_g\,t\} \\
& = & \{x \mapsto \{t(x)\} \mid x \in \mathcal{X}\} \cup \{g \mapsto \{0\} \mid g \in \mathcal{G}\}
\end{array}$$

Thus, $W = W^\sharp \sqsubseteq W^{\sharp'}$, $P = P^\sharp \sqsubseteq P^{\sharp'}$, and

$$\begin{array}{rcl}\sigma & = & \{x \mapsto \{t(x)\} \mid x \in \mathcal{X}\} \cup \{g \mapsto \{0\} \mid g \in \mathcal{G}\} \\ & \subseteq & \bar{\gamma} \circ (\{x \mapsto \top \mid x \in \mathcal{X} \setminus \{\mathsf{self}\}\} \cup \{\mathsf{self} \mapsto [\![i_0]\!]^\sharp_{\mathcal{E}xp}\, \top\} \cup \{g \mapsto [\![0]\!]^\sharp_{\mathcal{E}xp}\, \top \mid g \in \mathcal{G}\}) \\ & = & \bar{\gamma} \circ \sigma^\sharp \subseteq \bar{\gamma} \circ \sigma^{\sharp'}\end{array}$$

Altogether $t \in \eta'_{\mathsf{wc}}\,[u_0, \emptyset, A]$ holds for all $t \in \{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\}$.

Next, consider the constraints for **initMT**. Consider an edge $(u, \mathsf{initMT}, u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, \mathsf{initMT}]\!]^\sharp_{\mathcal{A}}(A_0)$. We remark that, by construction, the lockset is empty when executing initMT. We verify that

$$[\![([u, \emptyset, A_0], \mathsf{initMT}, u')]\!]_{\mathsf{wc}}\, \eta^{i-1} \quad \subseteq \quad (\eta'_{\mathsf{wc}}, \eta'_{\mathsf{wc}}\,[u', \emptyset, A'])$$

We have

$$\begin{array}{lcl}[\![([u, \emptyset, A_0], \mathsf{initMT}, u')]\!]_{\mathsf{wc}}\, \eta & = & (\emptyset, [\![(u, \mathsf{initMT}, u')]\!]_{\mathcal{T}}(\eta\,[u, \emptyset, A_0])) \\ [\![[u, \emptyset, A_0], \mathsf{initMT}]\!]^\sharp \eta^\sharp & = & \big(\emptyset, \eta^\sharp\,[u, \emptyset, A_0]\big)\end{array}$$

Let $\eta^\sharp\,[u, \emptyset, A_0] = (W^\sharp, P^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u', \emptyset, A'] = (W^{\sharp'}, P^{\sharp'}, \sigma^{\sharp'})$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge, the empty lockset, and the resulting digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $W^\sharp \sqsubseteq W^{\sharp'}$, $P^\sharp \sqsubseteq P^{\sharp'}$, and $\sigma^\sharp \sqsubseteq \sigma^{\sharp'}$ hold. Then, by definition:

$$\begin{array}{rl} & \eta'_{\mathsf{wc}}[u', \emptyset, A'] = \gamma_{u', \emptyset, A'}(W^{\sharp'}, P^{\sharp'}, \sigma^{\sharp'}) \\ = & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = \emptyset, \alpha_{\mathcal{A}}\, t = A', \beta\, t = (W, P, \sigma), \\ & \qquad \sigma \subseteq \bar{\gamma} \circ \sigma^{\sharp'}, W \sqsubseteq W^{\sharp'}, P \sqsubseteq P^{\sharp'}\}\end{array}$$

For every trace $t \in \eta^{i-1}\,[u, \emptyset, A_0]$, let $\beta\, t = (W, P, \sigma)$. By induction hypothesis, $W \sqsubseteq W^\sharp$, $P \sqsubseteq P^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let $t' = [\![(u, \mathsf{initMT}, u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = \emptyset$, $\alpha_{\mathcal{A}}\, t' = A'$, and

$$\begin{array}{rcl}\beta\, t' & = & (W', P', \sigma') \text{ where} \\ W' & = & \{g \mapsto \{L_{t'}[\bar{v}']\} \mid g \in \mathcal{G}, (\_, g = x, \bar{v}') = \mathsf{last\_tl\_write}_g\, t'\} \\ & & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\, t'\} \\ & = & W \\ P' & = & \{g \mapsto \mathsf{min\_lockset\_since}\, t'\, \bar{v}' \mid g \in \mathcal{G}, (\_, g = x, \bar{v}') = \mathsf{last\_tl\_write}_g\, t'\} \\ & & \cup\, \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\, t'\} \\ & = & P \\ \sigma' & = & \{x \mapsto \{t'(x)\} \mid x \in \mathcal{X}\} \\ & & \cup\, \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last\_write}_g\, t'\} \\ & & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\, t', \bot \neq \mathsf{last\_write}_g\, t'\} \\ & & \cup\, \{g \mapsto \{\sigma_{j-1}\, x\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x, \_) = \mathsf{last\_tl\_write}_g\, t'\} \\ & = & \sigma\end{array}$$

Thus, $W' = W \sqsubseteq W^\sharp \sqsubseteq W^{\sharp'}$, $P' = P \sqsubseteq P^\sharp \sqsubseteq P^{\sharp'}$, and $\sigma' = \sigma \subseteq \bar{\gamma} \circ \sigma^\sharp \subseteq \bar{\gamma} \circ \sigma^{\sharp'}$. Altogether, $t' \in \eta'_{\mathsf{wc}}\,[u', \emptyset, A']$ holds for all $t \in \eta^{i-1}\,[u, \emptyset, A_0]$. We conclude that the return value of $[\![([u, \emptyset, A_0], \mathsf{initMT}, u')]\!]_{\mathsf{wc}}\,\eta^{i-1}$ is subsumed by the value $\eta'_{\mathsf{wc}}\,[u', \emptyset, A']$ and since the constraint causes no side-effects, the claim holds.

Next, consider the constraints for a **read from a global** $x = g$. Consider an edge $(u, x = g, u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, x = g]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![([u, S, A_0], x = g, u')]\!]_{\mathsf{wc}}\,\eta^{i-1} \quad \subseteq \quad (\eta'_{\mathsf{wc}}, \eta'_{\mathsf{wc}}\,[u', S, A'])$$

We have

$$\begin{array}{lcl}
[\![([u, S, A_0], x = g, u')]\!]_{\mathsf{wc}}\,\eta & = & (\emptyset, [\![(u, x = g, u')]\!]_{\mathcal{T}}(\eta\,[u, S, A_0])) \\
[\![[u, S, A_0], x = g]\!]^\sharp \eta^\sharp & = & \textbf{let } (W^\sharp, P^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\
 & & \textbf{let } d = \sigma\,g \sqcup \bigsqcup\{\eta^\sharp[g, a, S', w, A'] \mid a \in S, S \cap S' = \emptyset, \\
 & & \quad \exists S'' \in P^\sharp\,g : S'' \cap w = \emptyset, \\
 & & \quad \exists S''' \in P^\sharp\,g : a \notin S''', \\
 & & \quad A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A'\} \textbf{ in} \\
 & & \textbf{let } \sigma^{\sharp''} = \sigma^\sharp \oplus \{x \mapsto d\} \textbf{ in} \\
 & & (\emptyset, (W^\sharp, P^\sharp, \sigma^{\sharp''}))
\end{array}$$

Let $\eta^\sharp\,[u, S, A_0] = (W^\sharp, P^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u', S, A'] = (W^{\sharp'}, P^{\sharp'}, \sigma^{\sharp'})$ the value provided by $\mathcal{C}^\sharp$ for the end point of the given control-flow edge and lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $W^\sharp \sqsubseteq W^{\sharp'}$, $P^\sharp \sqsubseteq P^{\sharp'}$, and $\sigma^{\sharp''} \sqsubseteq \sigma^{\sharp'}$ hold. Then, by definition:

$$\begin{array}{lcl}
 & & \eta'_{\mathsf{wc}}[u', S, A'] = \gamma_{u', S, A'}(W^{\sharp'}, P^{\sharp'}, \sigma^{\sharp'}) \\
 & = & \{t \in \mathcal{T} \mid \mathsf{loc}\,t = u', L_t = S, \alpha_{\mathcal{A}}\,t = A', \beta\,t = (W, P, \sigma), \\
 & & \qquad \sigma \subseteq \bar{\gamma} \circ \sigma^{\sharp'}, W \sqsubseteq W^{\sharp'}, P \sqsubseteq P^{\sharp'}\}
\end{array}$$

For every trace $t \in \eta^{i-1}\,[u, S, A_0]$, let $\beta\,t = (W, P, \sigma)$. By induction hypothesis, $W \sqsubseteq W^\sharp$, $P \sqsubseteq P^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let $t' = [\![(u, x = g, u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\,t' = u'$, $L_{t'} = S$, $\alpha_{\mathcal{A}}\,t' = A'$, and

$$\begin{array}{lcl}
\beta\,t' & = & (W', P', \sigma') \text{ where} \\
W' & = & \{g' \mapsto \{L_{t'}[\bar{v}']\} \mid g' \in \mathcal{G}, (_, g' = x', \bar{v}') = \mathsf{last_tl_write}_{g'}\,t'\} \\
 & & \cup\, \{g' \mapsto \emptyset \mid g' \in \mathcal{G}, \bot = \mathsf{last_tl_write}_{g'}\,t'\} \\
 & = & W \\
P' & = & \{g' \mapsto \mathsf{min_lockset_since}\,t'\,\bar{v}' \mid g' \in \mathcal{G}, (_, g' = x', \bar{v}') = \mathsf{last_tl_write}_{g'}\,t'\} \\
 & & \cup\, \{g' \mapsto \{\emptyset\} \mid g' \in \mathcal{G}, \bot = \mathsf{last_tl_write}_{g'}\,t'\} \\
 & = & P
\end{array}$$

$$
\begin{aligned}
\sigma' &= \{x' \mapsto \{t'(x')\} \mid x' \in \mathcal{X}\} \\
&\quad \cup \{g' \mapsto \{0\} \mid g' \in \mathcal{G}, \bot = \mathsf{last\_write}_{g'}\, t'\} \\
&\quad \cup \{g' \mapsto \emptyset \mid g' \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_{g'}\, t', \bot \neq \mathsf{last\_write}_{g'}\, t'\} \\
&\quad \cup \{g' \mapsto \{\sigma_{j-1}\, x'\} \mid g' \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g' = x', \_) = \mathsf{last\_tl\_write}_{g'}\, t'\} \\
&= \sigma \oplus \{x \mapsto \{t'(x)\}\} \\
&= \sigma \oplus \begin{cases} \{x \mapsto \{\sigma_{j'-1}\, x'\}\} & \text{if } \mathsf{last\_write}_g\, t' = ((j'-1, u_{j'-1}, \sigma_{j'-1}), g = x', \_) \\ \{x \mapsto \{0\}\} & \text{if } \mathsf{last\_write}_g\, t' = \bot \end{cases} \\
&= \sigma \oplus \begin{cases} \{x \mapsto \{\sigma_{j'-1}\, x'\}\} & \text{if } \mathsf{last\_write}_g\, t = ((j'-1, u_{j'-1}, \sigma_{j'-1}), g = x', \_) \\ \{x \mapsto \{0\}\} & \text{if } \mathsf{last\_write}_g\, t = \bot \end{cases}
\end{aligned}
$$

Thus, $W = W' \sqsubseteq W^\sharp \sqsubseteq W^{\sharp'}$ and $P = P' \sqsubseteq P^\sharp \sqsubseteq P^{\sharp'}$. Also $\sigma\, y = \sigma'\, y$ and therefore, $\sigma'\, y \subseteq (\bar{\gamma} \circ \sigma^{\sharp'})\, y$ for $y \not\equiv x$. For $y \equiv x$, we consider three cases:

- There is no write to g ($\mathsf{last\_write}_g\, t = \bot$): Then $\sigma\, g = \{0\} \subseteq (\bar{\gamma}\, \sigma^\sharp)\, g$, thus $\sigma'\, x \subseteq (\bar{\gamma} \circ \sigma^{\sharp''})\, x$ and accordingly, $\sigma' \subseteq \bar{\gamma} \circ \sigma^{\sharp'}$.
- The last write to g is thread-local ($\mathsf{last\_write}_g\, t = \mathsf{last\_tl\_write}_g\, t$): Then $\sigma\, g = \{\sigma_{j'-1}\, x'\} \subseteq (\bar{\gamma} \circ \sigma^\sharp)\, g$, thus $\sigma'\, x \subseteq (\bar{\gamma} \circ \sigma^{\sharp''})\, x$ and accordingly, $\sigma' \subseteq \bar{\gamma} \circ \sigma^{\sharp'}$.
- The last write to g is non-thread-local.

$$
\begin{aligned}
\sigma'\, x &\subseteq \bigcup\{\mathsf{eval\_tl}_g\, (\eta^{i-1}\, [g, a, S', w, A']) \mid a \in S, S \cap S' = \emptyset, w \subseteq \mathcal{M}, \\
&\qquad \exists S'' \in P : S'' \cap w = \emptyset, \exists S''' \in P : a \notin S''', \\
&\qquad \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \qquad \text{(By Proposition 17)} \\
&\subseteq \bigcup\{\mathsf{eval\_tl}_g\, (\eta'_{\mathsf{wc}}\, [g, a, S', w, A']) \mid a \in S, S \cap S' = \emptyset, w \subseteq \mathcal{M}, \\
&\qquad \exists S'' \in P : S'' \cap w = \emptyset, \exists S''' \in P : a \notin S''', \\
&\qquad \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \qquad \text{(By Induction Hypothesis)} \\
&\subseteq \bigcup\{\mathsf{eval\_tl}_g\, (\gamma_{g,a,S',w,A'}(\eta^\sharp\, [g, a, S', w, A'])) \mid a \in S, S \cap S' = \emptyset, w \subseteq \mathcal{M}, \\
&\qquad \exists S'' \in P : S'' \cap w = \emptyset, \exists S''' \in P : a \notin S''', \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \\
&\subseteq \bigcup\{\gamma_{\mathcal{V}^\sharp}(\eta^\sharp\, [g, a, S', w, A']) \mid a \in S, S \cap S' = \emptyset, w \subseteq \mathcal{M}, \\
&\qquad \exists S'' \in P : S'' \cap w = \emptyset, \exists S''' \in P : a \notin S''', \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \\
&\subseteq \gamma_{\mathcal{V}^\sharp}(\bigsqcup\{\eta^\sharp\, [g, a, S', w, A'] \mid a \in S, S \cap S' = \emptyset, w \subseteq \mathcal{M}, \\
&\qquad \exists S'' \in P : S'' \cap w = \emptyset, \exists S''' \in P : a \notin S''', \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\}) \\
&\subseteq \gamma_{\mathcal{V}^\sharp}(\sigma^\sharp\, g \sqcup \bigsqcup\{\eta^\sharp\, [g, a, S', w, A'] \mid a \in S, S \cap S' = \emptyset, w \subseteq \mathcal{M}, \\
&\qquad \exists S'' \in P : S'' \cap w = \emptyset, \exists S''' \in P : a \notin S''', \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\}) \\
&= \gamma_{\mathcal{V}^\sharp}(\sigma^{\sharp''}\, x) = (\bar{\gamma} \circ \sigma^{\sharp''})\, x \\
&\subseteq (\bar{\gamma} \circ \sigma^{\sharp'})\, x
\end{aligned}
$$

  and thus $\sigma' \subseteq \bar{\gamma} \circ \sigma^{\sharp''} \subseteq \bar{\gamma} \circ \sigma^{\sharp'}$.

Altogether, $t' \in \eta'_{\mathsf{wc}}\, [u', S, A']$ holds for all $t \in \eta^{i-1}\, [u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], x = g, u')]\!]_{\mathsf{wc}}\, \eta^{i-1}$ is subsumed by the value $\eta'_{\mathsf{wc}}\, [u', S, A']$ and since the constraint causes no side-effects, the claim holds.

Next, consider the constraints for a **write to a global** $g = x$. Consider an edge $(u, g = x, u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, g = x]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![([u, S, A_0], g = x, u')]\!]_{\mathsf{wc}}\, \eta^{i-1} \quad \subseteq \quad (\eta'_{\mathsf{wc}}, \eta'_{\mathsf{wc}}\,[u', S, A'])$$

We have

$$\begin{array}{lcl}
[\![([u, S, A_0], g = x, u')]\!]_{\mathsf{wc}}\, \eta & = & (\emptyset, [\![(u, g = x, u')]\!]_{\mathcal{T}}(\eta\,[u, S, A_0])) \\
[\![[u, S, A_0], g = x]\!]^\sharp \eta^\sharp & = & \textbf{let } (W^\sharp, P^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\
 & & \textbf{let } W^{\sharp\prime\prime} = W^\sharp \oplus \{g \mapsto \{S\}\} \textbf{ in} \\
 & & \textbf{let } P^{\sharp\prime\prime} = P^\sharp \oplus \{g \mapsto \{S\}\} \textbf{ in} \\
 & & \textbf{let } \sigma^{\sharp\prime\prime} = \sigma^\sharp \oplus \{g \mapsto \sigma^\sharp\, x\} \textbf{ in} \\
 & & (\emptyset, (W^{\sharp\prime\prime}, P^{\sharp\prime\prime}, \sigma^{\sharp\prime\prime}))
\end{array}$$

Let $\eta^\sharp\,[u, S, A_0] = (W^\sharp, P^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u', S, A'] = (W^{\sharp'}, P^{\sharp'}, \sigma^{\sharp'})$ the value provided by $\mathcal{C}^\sharp$ for the end point of the given control-flow edge and lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $W^\sharp \sqsubseteq W^{\sharp'}$, $P^\sharp \sqsubseteq P^{\sharp'}$, and $\sigma^{\sharp\prime\prime} \sqsubseteq \sigma^{\sharp'}$ hold. Then, by definition:

$$\begin{array}{lcl}
 & & \eta'_{\mathsf{wc}}[u', S, A'] = \gamma_{u', S, A'}(W^{\sharp'}, P^{\sharp'}, \sigma^{\sharp'}) \\
 & = & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = S, \alpha_{\mathcal{A}}\, t = A', \beta\, t = (W, P, \sigma), \\
 & & \qquad \sigma \subseteq \bar{\gamma} \circ \sigma^{\sharp'}, W \sqsubseteq W^{\sharp'}, P \sqsubseteq P^{\sharp'}\}
\end{array}$$

For every trace $t \in \eta^{i-1}\,[u, S, A_0]$, let $\beta\, t = (W, P, \sigma)$. By induction hypothesis, $W \sqsubseteq W^\sharp$, $P \sqsubseteq P^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let $t' = [\![(u, g = x, u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S$, $\alpha_{\mathcal{A}}\, t' = A'$, and

$$\begin{array}{lcl}
\beta\, t' & = & (W', P', \sigma') \text{ where} \\
W' & = & \{g' \mapsto \{L_{t'}[\bar{v}']\} \mid g' \in \mathcal{G}, (_, g' = x, \bar{v}') = \mathsf{last_tl_write}_{g'}\, t'\} \\
 & & \cup\, \{g' \mapsto \emptyset \mid g' \in \mathcal{G}, \bot = \mathsf{last_tl_write}_{g'}\, t'\} \\
 & = & W \oplus \{g \mapsto \{L_{t'}[\bar{v}']\} \mid (_, g = x', \bar{v}') = \mathsf{last_tl_write}_{g}\, t'\} \\
 & = & W \oplus \{g \mapsto \{S\}\} \\
P' & = & \{g' \mapsto \mathsf{min_lockset_since}\, t'\, \bar{v}' \mid g' \in \mathcal{G}, (_, g' = x', \bar{v}') = \mathsf{last_tl_write}_{g'}\, t'\} \\
 & & \cup\, \{g' \mapsto \{\emptyset\} \mid g' \in \mathcal{G}, \bot = \mathsf{last_tl_write}_{g'}\, t'\} \\
 & = & P \oplus \{g \mapsto \mathsf{min_lockset_since}\, t'\, \bar{v}'\} \mid (_, g = x', \bar{v}') = \mathsf{last_tl_write}_{g}\, t'\} \\
 & = & P \oplus \{g \mapsto \{S\}\} \\
\sigma' & = & \{x' \mapsto \{t'(x')\} \mid x' \in \mathcal{X}\} \\
 & & \cup\, \{g' \mapsto \{0\} \mid g' \in \mathcal{G}, \bot = \mathsf{last_write}_{g'}\, t'\} \\
 & & \cup\, \{g' \mapsto \emptyset \mid g' \in \mathcal{G}, \bot = \mathsf{last_tl_write}_{g'}\, t', \bot \neq \mathsf{last_write}_{g'}\, t'\} \\
 & & \cup\, \{g' \mapsto \{\sigma_{j-1}\, x'\} \mid g' \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g' = x', _) = \mathsf{last_tl_write}_{g'}\, t'\} \\
 & = & \sigma \oplus \{g \mapsto \{\sigma_{j-1}\, x'\} \mid g' \in ((j-1, u_{j-1}, \sigma_{j-1}), g = x', _) = \mathsf{last_tl_write}_{g}\, t'\} \\
 & = & \sigma \oplus \{g \mapsto \{t'(x)\}\} \\
 & = & \sigma \oplus \{g \mapsto \sigma\, x\}
\end{array}$$

Thus,

$$\begin{array}{lcl} W' & = & W \oplus \{g \mapsto \{S\}\} \sqsubseteq W^\sharp \oplus \{g \mapsto \{S\}\} = W^{\sharp''} \sqsubseteq W^{\sharp'} \\ P' & = & P \oplus \{g \mapsto \{S\}\} \sqsubseteq P^\sharp \oplus \{g \mapsto \{S\}\} = P^{\sharp''} \sqsubseteq P^{\sharp'} \\ \sigma' & = & \sigma \oplus \{g \mapsto \sigma\, x\} \subseteq \bar{\gamma} \circ (\sigma^\sharp \oplus \{g \mapsto \sigma^\sharp\, x\}) = \bar{\gamma} \circ \sigma^{\sharp''} \subseteq \bar{\gamma} \circ \sigma^{\sharp'} \end{array}$$

Altogether, $t' \in \eta'_{\mathsf{wc}}\,[u', S, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], g = x, u')]\!]_{\mathsf{wc}}\,\eta^{i-1}$ is subsumed by the value $\eta'_{\mathsf{wc}}\,[u', S, A']$ and since the constraint causes no side-effects, the claim holds.

Next, consider the constraints for **assignments involving local variables** $x = e$. Consider an edge $(u, x = e, u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, x = e]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![([u, S, A_0], x = e, u')]\!]_{\mathsf{wc}}\,\eta^{i-1} \quad \subseteq \quad (\eta'_{\mathsf{wc}}, \eta'_{\mathsf{wc}}\,[u', S, A'])$$

We have

$$\begin{array}{lcl} [\![([u, S, A_0], x = e, u')]\!]_{\mathsf{wc}}\,\eta & = & (\emptyset, [\![(u, x = e, u')]\!]_{\mathcal{T}}(\eta\,[u, S, A_0])) \\ [\![[u, S, A_0], x = e]\!]^\sharp \eta^\sharp & = & \textbf{let } (W^\sharp, P^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\ & & \textbf{let } \sigma^{\sharp''} = \textbf{if } e \equiv ? \textbf{ then } \sigma^\sharp|_{\mathcal{V}ars \setminus \{x\}} \\ & & \qquad\qquad \textbf{else } \sigma^\sharp \oplus \{x \mapsto [\![e]\!]^\sharp_{\mathcal{E}xp}\,\sigma^\sharp\} \\ & & \textbf{in} \\ & & (\emptyset, (W^\sharp, P^\sharp, \sigma^{\sharp''})) \end{array}$$

Let $\eta^\sharp\,[u, S, A_0] = (W^\sharp, P^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u', S, A'] = (W^{\sharp'}, P^{\sharp'}, \sigma^{\sharp'})$ the value provided by $\mathcal{C}^\sharp$ for the end point of the given control-flow edge and lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $W^\sharp \sqsubseteq W^{\sharp'}$, $P^\sharp \sqsubseteq P^{\sharp'}$, and $\sigma^{\sharp''} \sqsubseteq \sigma^{\sharp'}$ hold. Then, by definition:

$$\begin{array}{lcl} & & \eta'_{\mathsf{wc}}[u', S, A'] = \gamma_{u', S, A'}(W^{\sharp'}, P^{\sharp'}, \sigma^{\sharp'}) \\ & = & \{t \in \mathcal{T} \mid \mathsf{loc}\,t = u', L_t = S, \alpha_{\mathcal{A}}\,t = A', \beta\,t = (W, P, \sigma), \\ & & \qquad \sigma \subseteq \bar{\gamma} \circ \sigma^{\sharp'}, W \sqsubseteq W^{\sharp'}, P \sqsubseteq P^{\sharp'}\} \end{array}$$

For every trace $t \in \eta^{i-1}\,[u, S, A_0]$, let $\beta\,t = (W, P, \sigma)$. By induction hypothesis, $W \sqsubseteq W^\sharp$, $P \sqsubseteq P^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let $t' = [\![(u, x = e, u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\,t' = u'$, $L_{t'} = S$, $\alpha_{\mathcal{A}}\,t' = A'$, and

$$\begin{array}{lcl} \beta\,t' & = & (W', P', \sigma') \text{ where} \\ W' & = & \{g' \mapsto \{L_{t'}[\bar{v}']\} \mid g' \in \mathcal{G}, (\_, g' = x', \bar{v}') = \mathsf{last\_tl\_write}_{g'}\,t'\} \\ & & \cup\, \{g' \mapsto \emptyset \mid g' \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_{g'}\,t'\} \\ & = & W \\ P' & = & \{g' \mapsto \mathsf{min\_lockset\_since}\,t'\,\bar{v}' \mid g' \in \mathcal{G}, (\_, g' = x', \bar{v}') = \mathsf{last\_tl\_write}_{g'}\,t'\} \\ & & \cup\, \{g' \mapsto \{\emptyset\} \mid g' \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_{g'}\,t'\} \\ & = & P \end{array}$$

$$\begin{array}{rcl}
\sigma' & = & \{x' \mapsto \{t'(x')\} \mid x' \in \mathcal{X}\} \\
& & \cup\, \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\, t'\} \\
& & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t', \bot \neq \mathsf{last_write}_g\, t'\} \\
& & \cup\, \{g \mapsto \{\sigma_{j-1}\, x'\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x', _) = \mathsf{last_tl_write}_g\, t'\} \\
& = & \sigma \oplus \{x \mapsto \{t'(x)\}\} \\
& = & \sigma \oplus \{x \mapsto [\![e]\!]_{\mathcal{E}xp}\, \sigma\}
\end{array}$$

In case $e \not\equiv ?$, we have

$$\sigma \oplus \{x \mapsto \{t'(x)\}\} = \sigma \oplus \{x \mapsto [\![e]\!]_{\mathcal{E}xp}\, \sigma\}$$

where $[\![e]\!]_{\mathcal{E}xp}$ is appropriately lifted to work on maps from variables to singleton sets of values instead of on maps to values directly. In case of $e \equiv ?$, we have

$$\sigma \oplus \{x \mapsto \{t'(x)\}\} \in \{\sigma \oplus \{x \mapsto v\} \mid v \in \mathcal{V}_\tau\}$$

where τ corresponds to the type of x. Thus, $W = W' \sqsubseteq W^\sharp \sqsubseteq W^{\sharp'}$ and $P = P' \sqsubseteq P^\sharp \sqsubseteq P^{\sharp'}$. Also $\sigma\, y = \sigma'\, y$ and therefore, $\sigma'\, y \subseteq (\bar{\gamma} \circ \sigma^{\sharp'})\, y$ for $y \not\equiv x$. For $y \equiv x$ and $e \not\equiv ?$, we have

$$\begin{array}{rcl}
\sigma'\, x = [\![e]\!]_{\mathcal{E}xp}\, \sigma & \subseteq & \gamma_{\mathcal{V}^\sharp}([\![e]\!]^\sharp_{\mathcal{E}xp}\, \sigma^\sharp) = (\bar{\gamma} \circ (\sigma^\sharp \oplus \{x \mapsto [\![e]\!]^\sharp_{\mathcal{E}xp}\, \sigma^\sharp\}))\, x \\
& = & (\bar{\gamma} \circ \sigma^{\sharp''})\, x \\
& \subseteq & (\bar{\gamma} \circ \sigma^{\sharp'})\, x
\end{array}$$

as $[\![e]\!]^\sharp_{\mathcal{E}xp}$ is a sound abstraction of $[\![e]\!]_{\mathcal{E}xp}$. For $y \equiv x$ and $e \equiv ?$, on the other hand, we have

$$\sigma'\, x \in \mathcal{V}_\tau = (\bar{\gamma} \circ (\sigma^\sharp\big|_{\mathcal{V}ars \setminus \{x\}}))\, x = (\bar{\gamma} \circ \sigma^{\sharp''})\, x \subseteq (\bar{\gamma} \circ \sigma^{\sharp'})\, x$$

Thus, we have $\sigma' \subseteq \bar{\gamma} \circ \sigma^{\sharp'}$ and altogether, $t' \in \eta'_{\mathsf{wc}}\, [u', S, A']$ holds for all $t \in \eta^{i-1}\, [u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], x = e, u')]\!]_{\mathsf{wc}}\, \eta^{i-1}$ is subsumed by the value $\eta'_{\mathsf{wc}}\, [u', S, A']$ and since the constraint causes no side-effects, the claim holds.

Next, consider the constraints for a **guard involving local variables** $\mathsf{Pos}(e)$. Consider an edge $(u, \mathsf{Pos}(e), u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, Pos(e)]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![([u, S, A_0], \mathsf{Pos}(e), u')]\!]_{\mathsf{wc}}\, \eta^{i-1} \quad \subseteq \quad (\eta'_{\mathsf{wc}}, \eta'_{\mathsf{wc}}\, [u', S, A'])$$

We have

$$\begin{array}{rcl}
[\![([u, S, A_0], \mathsf{Pos}(e), u')]\!]_{\mathsf{wc}}\, \eta & = & (\emptyset, [\![(u, \mathsf{Pos}(e), u')]\!]_{\mathcal{T}}(\eta\, [u, S, A_0])) \\
[\![[u, S, A_0], \mathsf{Pos}(e)]\!]^\sharp \eta^\sharp & = & \textbf{let } (W^\sharp, P^\sharp, \sigma^\sharp) = \eta^\sharp\, [u, S, A_0] \textbf{ in} \\
& & \textbf{let } \sigma^{\sharp''} = [\![?(e \neq 0)]\!]^\sharp_{\mathcal{V}^\sharp} \sigma^\sharp \textbf{ in} \\
& & \textbf{if } \sigma^{\sharp''} = \bot \textbf{ then} \\
& & \quad (\emptyset, \underline{\bot}) \\
& & \textbf{else} \\
& & \quad (\emptyset, (W^\sharp, P^\sharp, \sigma^{\sharp''}))
\end{array}$$

Let $\eta^\sharp\,[u, S, A_0] = (W^\sharp, P^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u', S, A'] = (W^{\sharp'}, P^{\sharp'}, \sigma^{\sharp'})$ the value provided by $\mathcal{C}^\sharp$ for the end point of the given control-flow edge and lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, we either have $\sigma^{\sharp''} = \bot$ or $W^\sharp \sqsubseteq W^{\sharp'}$, $P^\sharp \sqsubseteq P^{\sharp'}$, and $\sigma^{\sharp''} \sqsubseteq \sigma^{\sharp'}$ hold. Then, by definition:

$$\begin{array}{rl} & \eta'_{\mathsf{wc}}[u', S, A'] = \gamma_{u',S,A'}(W^{\sharp'}, P^{\sharp'}, \sigma^{\sharp'}) \\ = & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = S, \alpha_{\mathcal{A}}\, t = A', \beta\, t = (W, P, \sigma), \\ & \qquad \sigma \subseteq \bar{\gamma} \circ \sigma^{\sharp'}, W \sqsubseteq W^{\sharp'}, P \sqsubseteq P^{\sharp'}\} \end{array}$$

For every trace $t \in \eta^{i-1}\,[u, S, A_0]$, let $\beta\, t = (W, P, \sigma)$. By induction hypothesis, $W \sqsubseteq W^\sharp$, $P \sqsubseteq P^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Then $[\![(u, \mathsf{Pos}(e), u')]\!]_{\mathcal{T}}\{t\}$ yields a non-empty set for those t where $\mathsf{sink}\, t = (\_, \sigma)$ and $[\![e]\!]_{\mathcal{E}xp}\, \sigma \neq 0$. If this is the case, let $t' = [\![(u, \mathsf{Pos}(e), u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S$, $\alpha_{\mathcal{A}}\, t' = A'$, and

$$\begin{array}{rcl} \beta\, t' & = & (W', P', \sigma') \text{ where} \\ W' & = & \{g' \mapsto \{L_{t'}[\bar{v}']\} \mid g' \in \mathcal{G}, (\_, g' = x', \bar{v}') = \mathsf{last\_tl\_write}_{g'}\, t'\} \\ & & \cup\, \{g' \mapsto \emptyset \mid g' \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_{g'}\, t'\} \\ & = & W \\ P' & = & \{g' \mapsto \mathsf{min\_lockset\_since}\, t'\, \bar{v}' \mid g' \in \mathcal{G}, (\_, g' = x', \bar{v}') = \mathsf{last\_tl\_write}_{g'}\, t'\} \\ & & \cup\, \{g' \mapsto \{\emptyset\} \mid g' \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_{g'}\, t'\} \\ & = & P \\ \sigma' & = & \{x' \mapsto \{t'(x')\} \mid x' \in \mathcal{X}\} \\ & & \cup\, \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last\_write}_g\, t'\} \\ & & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\, t', \bot \neq \mathsf{last\_write}_g\, t'\} \\ & & \cup\, \{g \mapsto \{\sigma_{j-1}\, x'\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x', \_) = \mathsf{last\_tl\_write}_g\, t'\} \\ & = & \sigma \end{array}$$

It thus remains to show that if such a t' exists, $\sigma^{\sharp''} \neq \bot$ holds, and that then $W \sqsubseteq W^{\sharp'}$, $P \sqsubseteq P^{\sharp'}$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^{\sharp'}$ holds. The first part follows directly from the soundness of $[\![?(e \neq 0)]\!]^\sharp_{\hat{\mathcal{V}}^\sharp}$, as $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$ and $[\![e]\!]_{\mathcal{E}xp}\, \sigma \neq 0$ implies $\bot \neq \sigma \subseteq \bar{\gamma} \circ ([\![?(e \neq 0)]\!]^\sharp_{\hat{\mathcal{V}}^\sharp} \sigma^\sharp) = \sigma^{\sharp''}$. Then, as $\sigma^{\sharp''} \neq \bot$, we have $\sigma^{\sharp''} \sqsubseteq \sigma^{\sharp'}$, $W^\sharp \sqsubseteq W^{\sharp'}$, and $P^\sharp \sqsubseteq P^{\sharp'}$ and altogether $W = W' \sqsubseteq W^\sharp \sqsubseteq W^{\sharp'}$, $P = P' \sqsubseteq P^\sharp \sqsubseteq P^{\sharp'}$, and $\sigma' = \sigma \subseteq \bar{\gamma} \circ \sigma^{\sharp'}$. Thus, $t' \in \eta'_{\mathsf{wc}}\,[u', S, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], \mathsf{Pos}(e), u')]\!]_{\mathsf{wc}}\, \eta^{i-1}$ is subsumed by the value $\eta'_{\mathsf{wc}}\,[u', S, A']$ and since the constraint causes no side-effects, the claim holds. The proof for negative guards $\mathsf{Neg}(e)$ is analogous.

Next, consider the constraints corresponding to **locking** a mutex a. Consider an edge $(u, \mathsf{lock}(a), u') \in \mathcal{E}$ and digests A', A_0 and all appropriate A_1 such that $A' \in [\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}(A_0, A_1)$. We verify that

$$[\![([u, S, A_0], \mathsf{lock}(a), u')]\!]_{\mathsf{wc}}\, \eta^{i-1} \quad \subseteq \quad (\eta'_{\mathsf{wc}}, \eta'_{\mathsf{wc}}\,[u', S \cup \{a\}, A'])$$

We have

$$
\begin{array}{l}
[\![([u,S,A_0],\mathsf{lock}(a),u')]\!]_{\mathsf{wc}}\,\eta = \\
\quad \textbf{let } T_1 = \bigcup\{\eta\,[g,a,S',w,A_1] \mid g \in \mathcal{G}, S' \subseteq \mathcal{M}, w \subseteq \mathcal{M}, A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1\} \textbf{ in} \\
\quad \textbf{let } T_2 = \{t' \mid t \in \eta\,[u,S,A_0], \bar{\imath} = \mathsf{init\_v}\,t, \bar{\imath} \neq \bot, t' = {\downarrow_{\bar{\imath}}}\,(t), \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,(\alpha_{\mathcal{A}}\,t')\} \textbf{ in} \\
\quad \textbf{let } T = [\![(u,\mathsf{lock}(a),u')]\!]_{\mathcal{T}}(\eta\,[u,S,A_0], T_1 \cup T_2) \textbf{ in} \\
\quad (\emptyset, T)
\end{array}
$$

$$
[\![[u,S,A_0],\mathsf{lock}(a)]\!]^\sharp \eta^\sharp = (\emptyset, \eta^\sharp\,[u,S,A_0])
$$

Let $\eta^\sharp\,[u,S,A_0] = (W^\sharp, P^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u',S\cup\{a\},A'] = (W^{\sharp'}, P^{\sharp'}, \sigma^{\sharp'})$ the value provided by $\mathcal{C}^\sharp$ for the end point of the given control-flow edge and lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $W^\sharp \sqsubseteq W^{\sharp'}$, $P^\sharp \sqsubseteq P^{\sharp'}$, and $\sigma^\sharp \sqsubseteq \sigma^{\sharp'}$ hold. Then, by definition:

$$
\begin{array}{rcl}
 & & \eta'_{\mathsf{wc}}[u',S\cup\{a\},A'] = \gamma_{u',S\cup\{a\},A'}(W^{\sharp'}, P^{\sharp'}, \sigma^{\sharp'}) \\
 & = & \{t \in \mathcal{T} \mid \mathsf{loc}\,t = u', L_t = S\cup\{a\}, \alpha_{\mathcal{A}}\,t = A', \beta\,t = (W,P,\sigma), \\
 & & \qquad \sigma \subseteq \bar{\gamma} \circ \sigma^{\sharp'}, W \sqsubseteq W^{\sharp'}, P \sqsubseteq P^{\sharp'}\}
\end{array}
$$

For every trace $t \in \eta^{i-1}\,[u,S,A_0]$, let $\beta\,t = (W,P,\sigma)$. By induction hypothesis, $W \sqsubseteq W^\sharp$, $P \sqsubseteq P^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let

$$
\begin{array}{rcl}
t' & \in & [\![(u,\mathsf{lock}(a),u')]\!]_{\mathcal{T}}(\{t\}, \\
 & & \quad \bigcup\{\eta\,[g,a,S',w,A_1] \mid g \in \mathcal{G}, S' \subseteq \mathcal{M}, w \subseteq \mathcal{M}, A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1\} \\
 & & \quad \cup\{t'' \mid t' \in \eta\,[u,S,A_0], \bar{\imath} = \mathsf{init\_v}\,t', \bar{\imath} \neq \bot, t'' = {\downarrow_{\bar{\imath}}}\,(t'), \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,(\alpha_{\mathcal{A}}\,t'')\}) \\
 & = & [\![(u,\mathsf{lock}(a),u')]\!]_{\mathcal{T}}(\{t\}, \\
 & & \quad \bigcup\{\eta\,[g,a,S',w,A_1] \mid g \in \mathcal{G}, S' \subseteq \mathcal{M}, w \subseteq \mathcal{M}, A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1\} \\
 & & \quad \cup\{t' \mid \bar{\imath} = \mathsf{init\_v}\,t, \bar{\imath} \neq \bot, t' = {\downarrow_{\bar{\imath}}}\,(t)\})
\end{array}
$$

where the equality exploits that for a given local trace t and a first lock of a, the second trace which contains the observed initMT must be a sub-trace of t. Then $\mathsf{loc}\,t' = u'$, $L_{t'} = S\cup\{a\}$, $\alpha_{\mathcal{A}}\,t' = A'$, and

$$
\begin{array}{rcl}
\beta\,t' & = & (W',P',\sigma') \text{ where} \\
W' & = & \{g \mapsto \{L_{t'}[\bar{v}']\} \mid g \in \mathcal{G}, (\_, g = x, \bar{v}') = \mathsf{last\_tl\_write}_g\,t'\} \\
 & & \cup\,\{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\,t'\} \\
 & = & W \\
P' & = & \{g \mapsto \mathsf{min\_lockset\_since}\,t'\,\bar{v}' \mid g \in \mathcal{G}, (\_, g = x, \bar{v}') = \mathsf{last\_tl\_write}_g\,t'\} \\
 & & \cup\,\{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\,t'\} \\
 & = & P \\
\sigma' & = & \{x \mapsto \{t'(x)\} \mid x \in \mathcal{X}\} \\
 & & \cup\,\{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last\_write}_g\,t'\} \\
 & & \cup\,\{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\,t', \bot \neq \mathsf{last\_write}_g\,t'\} \\
 & & \cup\,\{g \mapsto \{\sigma_{j-1}\,x'\} \mid g \in \mathcal{G}, ((j-1,u_{j-1},\sigma_{j-1}), g = x', \_) = \mathsf{last\_tl\_write}_g\,t'\} \\
 & = & \sigma \oplus \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\,t', \bot \neq \mathsf{last\_write}_g\,t'\} \\
 & \subseteq & \sigma
\end{array}
$$

Thus, $W = W' \sqsubseteq W^\sharp \sqsubseteq W^{\sharp\prime}$, $P = P' \sqsubseteq P^\sharp \sqsubseteq P^{\sharp\prime}$, and $\sigma' \subseteq \sigma \subseteq \bar{\gamma} \circ \sigma^\sharp \subseteq \bar{\gamma} \circ \sigma^{\sharp\prime}$. Altogether, $t' \in \eta'_{\mathsf{wc}}\,[u', S \cup \{a\}, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], \mathsf{lock}(a), u')]\!]_{\mathsf{wc}}\,\eta^{i-1}$ is subsumed by the value $\eta'_{\mathsf{wc}}\,[u', S \cup \{a\}, A']$ and since the constraint causes no side-effects, the claim holds.

Next, consider the constraints **unlocking** some mutex a. Consider an edge of the form $(u, \mathsf{unlock}(a), u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, \mathsf{unlock}(a)]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![([u, S, A_0], \mathsf{unlock}(a), u'), A']\!]_{\mathsf{wc}}\,\eta^{i-1} \quad \subseteq \quad (\eta'_{\mathsf{wc}}, \eta'_{\mathsf{wc}}\,[u', S \setminus \{a\}, A'])$$

We have

$$\begin{array}{l}
[\![([u, S, A_0], \mathsf{unlock}(a), u'), A']\!]_{\mathsf{wc}}\,\eta = \\
\quad \textbf{let } T = [\![(u, \mathsf{unlock}(a), u')]\!]_{\mathcal{T}}(\eta\,[u, S, A_0]) \textbf{ in} \\
\quad \textbf{let } \rho = \{[g, a, S \setminus \{a\}, w, A'] \mapsto \{t\} \mid t \in T, g \in \mathcal{G}, w \subseteq \mathcal{M}, \\
\qquad ((\mathsf{last_tl_write}_g\,t = (\bar{u}, g = x, \bar{u}') \wedge L_t[\bar{u}] \subseteq w) \vee (\mathsf{last_tl_write}_g\,t = \bot))\} \\
\quad \textbf{in} \\
\quad (\rho, T)
\end{array}$$

$$\begin{array}{l}
[\![[u, S, A_0], \mathsf{unlock}(a), A']\!]^\sharp \eta^\sharp = \\
\quad \textbf{let } (W^\sharp, P^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\
\quad \textbf{let } P^{\sharp\prime\prime} = \{g \mapsto P^\sharp\,g \sqcup \{S \setminus \{a\}\} \mid g \in \mathcal{G}\} \textbf{ in} \\
\quad \textbf{let } \rho^\sharp = \{[g, a, S \setminus \{a\}, w, A'] \mapsto \sigma^\sharp\,g \mid g \in \mathcal{G}, w' \in W^\sharp\,g, w' \subseteq w\} \textbf{ in} \\
\quad (\rho^\sharp, (W^\sharp, P^{\sharp\prime\prime}, \sigma^\sharp))
\end{array}$$

Let $\eta^\sharp\,[u, S, A_0] = (W^\sharp, P^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u', S \setminus \{a\}, A'] = (W^{\sharp\prime}, P^{\sharp\prime}, \sigma^{\sharp\prime})$ the value provided by $\mathcal{C}^\sharp$ for the end point of the given control-flow edge and lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $W^\sharp \sqsubseteq W^{\sharp\prime}$, $P^{\sharp\prime\prime} \sqsubseteq P^{\sharp\prime}$, and $\sigma^\sharp \sqsubseteq \sigma^{\sharp\prime}$ hold. Then, by definition:

$$\begin{array}{rcl}
 & & \eta'_{\mathsf{wc}}[u', S \setminus \{a\}, A'] = \gamma_{u', S\setminus\{a\}, A'}(W^{\sharp\prime}, P^{\sharp\prime}, \sigma^{\sharp\prime}) \\
= & & \{t \in \mathcal{T} \mid \mathsf{loc}\,t = u', L_t = S \setminus \{a\}, \alpha_{\mathcal{A}}\,t = A', \beta\,t = (W, P, \sigma), \\
 & & \qquad \sigma \subseteq \bar{\gamma} \circ \sigma^{\sharp\prime}, W \sqsubseteq W^{\sharp\prime}, P \sqsubseteq P^{\sharp\prime}\}
\end{array}$$

For every trace $t \in \eta^{i-1}\,[u, S, A_0]$, let $\beta\,t = (W, P, \sigma)$. By induction hypothesis, $W \sqsubseteq W^\sharp$, $P \sqsubseteq P^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let $t' = [\![(u, \mathsf{unlock}(a), u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\,t' = u'$, $L_{t'} = S \setminus \{a\}$, $\alpha_{\mathcal{A}}\,t' = A'$, and

$$\begin{array}{rcl}
\beta\,t' & = & (W', P', \sigma') \text{ where} \\
W' & = & \{g \mapsto \{L_{t'}[\bar{v}']\} \mid g \in \mathcal{G}, (_, g = x, \bar{v}') = \mathsf{last_tl_write}_g\,t'\} \\
 & & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\,t'\} \\
 & = & W
\end{array}$$

$$\begin{aligned}
P' &= \{g \mapsto \mathsf{min_lockset_since}\, t'\, \bar{v}' \mid g \in \mathcal{G}, (_, g = x, \bar{v}') = \mathsf{last_tl_write}_g\, t'\} \\
&\quad \cup \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t'\} \\
&= \{g \mapsto (\mathsf{min_lockset_since}\, t\, \bar{v}' \sqcup \{S \setminus \{a\}\}) \mid \\
&\qquad\qquad g \in \mathcal{G}, (_, g = x, \bar{v}') = \mathsf{last_tl_write}_g\, t'\} \\
&\quad \cup \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t'\} \\
&= \{g \mapsto (P\, g \sqcup \{S \setminus \{a\}\}) \mid g \in \mathcal{G}\} \\
\sigma' &= \{x \mapsto \{t'(x)\} \mid x \in \mathcal{X}\} \\
&\quad \cup \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\, t'\} \\
&\quad \cup \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t', \bot \neq \mathsf{last_write}_g\, t'\} \\
&\quad \cup \{g \mapsto \{\sigma_{j-1}\, x'\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x', _) = \mathsf{last_tl_write}_g\, t'\} \\
&= \sigma
\end{aligned}$$

Thus, $W' = W \sqsubseteq W^\sharp \sqsubseteq W^{\sharp'}$, $\sigma' = \sigma \subseteq \bar{\gamma} \circ \sigma^\sharp \subseteq \bar{\gamma} \circ \sigma^{\sharp'}$, and

$$\begin{aligned}
P' &= \{g \mapsto (P\, g \sqcup \{S \setminus \{a\}\}) \mid g \in \mathcal{G}\} \\
&\sqsubseteq \{g \mapsto (P^\sharp\, g \sqcup \{S \setminus \{a\}\}) \mid g \in \mathcal{G}\} = P^{\sharp''} \sqsubseteq P^{\sharp'}
\end{aligned}$$

Altogether, $t' \in \eta'_{\mathsf{wc}}\,[u', S \setminus \{a\}, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], \mathsf{unlock}(a), u'), A']\!]_{\mathsf{wc}}\, \eta^{i-1}$ is subsumed by the value $\eta'_{\mathsf{wc}}\,[u', S \setminus \{a\}, A']$ Next, we consider the side-effects of the corresponding right-hand-side functions. For each $g \in \mathcal{G}$, we distinguish two cases for t':

- $\mathsf{last\_tl\_write}_g\, t' = \bot$: Then side-effects $\{[g, a, S \setminus \{a\}, w, A'] \mapsto \{t'\} \mid w \subseteq \mathcal{M}\}$ are caused. These are accounted for by construction of η'_{wc}:

  $$\begin{aligned}
  t' &\in \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{unlock}(a), L_t = S \setminus \{a\}, \alpha_{\mathcal{A}}\, t = A, \mathsf{last_tl_write}_g\, t = \bot\} \\
  &\subseteq \eta'_{\mathsf{wc}}\,[g, a, S \setminus \{a\}, w, A]
  \end{aligned}$$

- $\mathsf{last\_tl\_write}_g\, t' = ((j-1, u_{j-1}, \sigma_{j-1}), g = x, \bar{u}')$ with $(j-1, u_{j-1}, \sigma_{j-1}) = \bar{u}$: Then the side-effects caused to unknowns associated with g in $\mathcal{C}_{\mathsf{wc}}$ and $\mathcal{C}^\sharp$, respectively, are given by

  $$\begin{aligned}
  \rho' &= \{[g, a, S \setminus \{a\}, w, A'] \mapsto \{t'\} \mid L_t[\bar{u}] \subseteq w\} \\
  \rho^{\sharp'} &= \{[g, a, S \setminus \{a\}, w'', A'] \mapsto \sigma^\sharp\, g \mid w' \in W^\sharp\, g, w' \subseteq w''\}
  \end{aligned}$$

  We have $\sigma\, g = \{\sigma_{j-1}\, x\} \subseteq (\bar{\gamma} \circ \sigma^\sharp)g$ and as $W \sqsubseteq W^\sharp$, there is $w' \in W^\sharp\, g$ where $w' \subseteq L_t[\bar{u}']$. Thus, we have

  $$t' \in \gamma_{g,a,S\setminus\{a\},w,A'}(\sigma^\sharp\, g) \subseteq \eta'_{\mathsf{wc}}\,[g, a, S \setminus \{a\}, w, A']$$

  for all w such that $L_t[\bar{u}] \subseteq w$ where we used that $\sigma^\sharp\, g \sqsubseteq \eta^\sharp\,[g, a, S \setminus \{a\}, w, A']$ holds as $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$.

Hence, all side-effects for $\mathsf{unlock}(a)$ of $\mathcal{C}_{\mathsf{wc}}$ are accounted for in η'_{wc}, and the claim holds.

Next, consider the constraints corresponding to **starting a new thread**. Consider an edge $(u, x = \mathsf{create}(u_1), u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, x = \mathsf{create}(u_1)]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![([u, S, A_0], x = \mathsf{create}(u_1), u')]\!]_{\mathsf{wc}}\, \eta^{i-1} \quad \subseteq \quad (\eta'_{\mathsf{wc}}, \eta'_{\mathsf{wc}}\,[u', S, A'])$$

We have

$$\begin{array}{l}
[\![([u, S, A_0], x = \mathsf{create}(u_1), u')]\!]_{\mathsf{wc}}\, \eta = \\
\quad \textbf{let } T = [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\eta\,[u, S, A_0]) \textbf{ in} \\
\quad \textbf{let } \rho = \{[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0] \mapsto \mathsf{new}\, u_1\, (\eta\,[u, S, A_0])\} \textbf{ in} \\
\quad (\rho, T) \\
\\
[\![[u, S, A_0], x = \mathsf{create}(u_1)]\!]^\sharp \eta^\sharp = \textbf{let } (W^\sharp, P^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\
\quad \textbf{let } W^\sharp_\rho = \{g \mapsto \emptyset \mid g \in \mathcal{G}\} \textbf{ in} \\
\quad \textbf{let } P^\sharp_\rho = \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}\} \textbf{ in} \\
\quad \textbf{let } i = \nu^\sharp\, u\, \sigma^\sharp\, u_1 \textbf{ in} \\
\quad \textbf{let } \sigma^\sharp_\rho = \sigma^\sharp \oplus \left((\{\mathsf{self} \mapsto i\}) \cup \left\{g \mapsto \left(\sigma\, g \sqcap [\![0]\!]^\sharp_{\mathcal{E}xp}\, \top\right) \mid g \in \mathcal{G}\right\}\right) \textbf{ in} \\
\quad \textbf{let } \rho^\sharp = \{[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0] \mapsto (W^\sharp_\rho, P^\sharp_\rho, \sigma^\sharp_\rho)\} \textbf{ in} \\
\quad \textbf{let } \sigma^{\sharp\prime\prime} = \sigma^\sharp \oplus \{x \mapsto i\} \textbf{ in} \\
\quad (\rho^\sharp, (W^\sharp, P^\sharp, \sigma^{\sharp\prime\prime}))
\end{array}$$

where we, for notational convenience, denote by $\mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0$ the only element of this singleton set. Let $\eta^\sharp\,[u, S, A_0] = (W^\sharp, P^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u', S, A'] = (W^{\sharp\prime}, P^{\sharp\prime}, \sigma^{\sharp\prime})$ the value provided by $\mathcal{C}^\sharp$ for the end point of the given control-flow edge and lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $W^\sharp \sqsubseteq W^{\sharp\prime}$, $P^\sharp \sqsubseteq P^{\sharp\prime}$, and $\sigma^{\sharp\prime\prime} \sqsubseteq \sigma^{\sharp\prime}$ hold. Then, by definition:

$$\begin{array}{rl}
& \eta'_{\mathsf{wc}}[u', S, A'] = \gamma_{u', S, A'}(W^{\sharp\prime}, P^{\sharp\prime}, \sigma^{\sharp\prime}) \\
= & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = S, \alpha_{\mathcal{A}}\, t = A', \beta\, t = (W, P, \sigma), \\
& \qquad \sigma \subseteq \bar{\gamma} \circ \sigma^{\sharp\prime}, W \sqsubseteq W^{\sharp\prime}, P \sqsubseteq P^{\sharp\prime}\}
\end{array}$$

For every trace $t \in \eta^{i-1}\,[u, S, A_0]$, let $\beta\, t = (W, P, \sigma)$. By induction hypothesis, $W \sqsubseteq W^\sharp$, $P \sqsubseteq P^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let $t' = [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S$, $\alpha_{\mathcal{A}}\, t' = A'$, and

$$\begin{array}{rcl}
\beta\, t' & = & (W', P', \sigma') \text{ where} \\
W' & = & \{g \mapsto \{L_{t'}[\bar{v}']\} \mid g \in \mathcal{G}, (_, g = x', \bar{v}') = \mathsf{last_tl_write}_g\, t'\} \\
& & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t'\} \\
& = & W \\
P' & = & \{g \mapsto \mathsf{min_lockset_since}\, t'\, \bar{v}' \mid g \in \mathcal{G}, (_, g = x', \bar{v}') = \mathsf{last_tl_write}_g\, t'\} \\
& & \cup\, \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t'\} \\
& = & P
\end{array}$$

$$\begin{aligned}
\sigma' &= \{x' \mapsto \{t'(x')\} \mid x' \in \mathcal{X}\} \\
&\quad \cup \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\, t'\} \\
&\quad \cup \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t', \bot \neq \mathsf{last_write}_g\, t'\} \\
&\quad \cup \{g \mapsto \{\sigma_{j-1}\, x'\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x', _) = \mathsf{last_tl_write}_g\, t'\} \\
&= \sigma \oplus \{x \mapsto \{\nu(t)\}\}
\end{aligned}$$

Thus $W' = W \sqsubseteq W^\sharp \sqsubseteq W^{\sharp'}$, $P' = P \sqsubseteq P^\sharp \sqsubseteq P^{\sharp'}$, and as $\nu(t) \in \gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}(\nu^\sharp\, u\, \sigma^\sharp\, u_1)$ by (3.2),

$$\begin{aligned}
\sigma' &= \sigma \oplus \{x \mapsto \{\nu(t)\}\} \subseteq \bar{\gamma} \circ (\sigma^\sharp \oplus \{x \mapsto \nu^\sharp\, u\, \sigma^\sharp\, u_1\}) = \bar{\gamma} \circ \sigma^{\sharp''} \\
&\subseteq \bar{\gamma} \circ \sigma^{\sharp'}
\end{aligned}$$

Altogether, $t' \in \eta'_{\mathsf{wc}}\,[u', S, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], x = \mathsf{create}(u_1), u')]\!]_{\mathsf{wc}}\, \eta^{i-1}$ is subsumed by the value $\eta'_{\mathsf{wc}}\,[u', S, A']$.

Next, we consider the side-effects of the corresponding right-hand-side functions for a t as considered previously.

$$\begin{aligned}
\rho &= \{[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0] \mapsto \mathsf{new}\, u_1\, \{t\}\} \\
\rho^\sharp &= \{[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0] \mapsto (W^\sharp_\rho, P^\sharp_\rho, \sigma^\sharp_\rho)\}
\end{aligned}$$

Let $\eta^\sharp\,[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0] = (W^{\sharp'}_\rho, P^{\sharp'}_\rho, \sigma^{\sharp'}_\rho)$ the value provided by $\mathcal{C}^\sharp$ for the unknown receiving the side-effect. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $W^\sharp_\rho \sqsubseteq W^{\sharp'}_\rho$, $P^\sharp_\rho \sqsubseteq P^{\sharp'}_\rho$, and $\sigma^\sharp_\rho \sqsubseteq \sigma^{\sharp'}_\rho$ hold. By definition:

$$\begin{aligned}
&\eta'_{\mathsf{wc}}[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0] = \gamma_{u', S, \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0}(W^{\sharp'}, P^{\sharp'}, \sigma^{\sharp'}) \\
= \; &\{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = \emptyset, \alpha_{\mathcal{A}}\, t = \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0, \beta\, t = (W, P, \sigma), \\
&\qquad \sigma \subseteq \bar{\gamma} \circ \sigma^{\sharp'}_\rho, W \sqsubseteq W^{\sharp'}_\rho, P \sqsubseteq P^{\sharp'}_\rho\}
\end{aligned}$$

Let $t'' = \mathsf{new}\, u_1\, \{t\}$. Then, $\mathsf{loc}\, t'' = u_1$, $L_{t''} = \emptyset$, $\alpha_{\mathcal{A}}\, t'' = \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0$, and

$$\begin{aligned}
\beta\, t'' &= (W_\rho, P_\rho, \sigma_\rho) \text{ where} \\
W_\rho &= \{g \mapsto \{L_{t''}[\bar{v}']\} \mid g \in \mathcal{G}, (_, g = x', \bar{v}') = \mathsf{last_tl_write}_g\, t''\} \\
&\quad \cup \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t''\} \\
&= \{g \mapsto \emptyset \mid g \in \mathcal{G}\} \\
P_\rho &= \{g \mapsto \mathsf{min_lockset_since}\, t''\, \bar{v}' \mid g \in \mathcal{G}, (_, g = x', \bar{v}') = \mathsf{last_tl_write}_g\, t''\} \\
&\quad \cup \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t''\} \\
&= \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}\} \\
\sigma_\rho &= \{x' \mapsto \{t''(x')\} \mid x' \in \mathcal{X}\} \\
&\quad \cup \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\, t''\} \\
&\quad \cup \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t'', \bot \neq \mathsf{last_write}_g\, t''\} \\
&\quad \cup \{g \mapsto \{\sigma_{j-1}\, x'\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x', _) = \mathsf{last_tl_write}_g\, t''\} \\
&= \sigma \oplus (\{\mathsf{self} \mapsto \{\nu(t)\}\} \\
&\qquad \cup \{g \mapsto (\{0\} \cap \sigma\, g) \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\, t''\}) \\
&\qquad \cup \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot \neq \mathsf{last_write}_g\, t''\}
\end{aligned}$$

Therefore,

$$\begin{array}{lcl}
W_\rho & = & \{g \mapsto \emptyset \mid g \in \mathcal{G}\} = W_\rho^\sharp \sqsubseteq W_\rho^{\sharp'} \\
P_\rho & = & \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}\} = P_\rho^\sharp \sqsubseteq P_\rho^{\sharp'} \\
\sigma_\rho & = & \sigma \oplus (\{\mathsf{self} \mapsto \{\nu(t)\}\} \\
& & \quad \cup \{g \mapsto (\{0\} \cap \sigma\, g) \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\, t''\}) \\
& & \quad \cup \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot \neq \mathsf{last_write}_g\, t''\} \\
& \subseteq & (\bar\gamma \circ (\sigma^\sharp \oplus \{\mathsf{self} \mapsto \nu^\sharp\, u\, \sigma^\sharp\, u_1\})) \\
& & \oplus (\{g \mapsto (\{0\} \cap \sigma\, g) \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\, t''\} \\
& & \quad \cup \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot \neq \mathsf{last_write}_g\, t''\}) \\
& \subseteq & (\bar\gamma \circ (\sigma^\sharp \oplus \{\mathsf{self} \mapsto \nu^\sharp\, u\, \sigma^\sharp\, u_1\})) \oplus \{g \mapsto (\{0\} \cap \sigma\, g) \mid g \in \mathcal{G}\} \\
& \subseteq & \bar\gamma \circ \left(\sigma^\sharp \oplus \left(\{\mathsf{self} \mapsto \nu^\sharp\, u\, \sigma^\sharp\, u_1\} \cup \left\{g \mapsto \left(\sigma^\sharp\, g \sqcap [\![0]\!]^\sharp_{\mathcal{E}xp}\, \top\right) \mid g \in \mathcal{G}\right\}\right)\right) \\
& = & \bar\gamma \circ \sigma_\rho^\sharp \\
& \subseteq & \bar\gamma \circ \sigma_\rho^{\sharp'}
\end{array}$$

Altogether, $t'' \in \eta'_{\mathsf{wc}}\,[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0]$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. Hence, all side-effects for $x = \mathsf{create}(u_1)$ of $\mathcal{C}_{\mathsf{wc}}$ are accounted for in η'_{wc}, and the claim holds.

Next, consider the constraints corresponding to **returning** from a thread. Consider an edge $(u, \mathsf{return}, u') \in \mathcal{E}$ and digests A' and A_0 such that $A' \in [\![u, \mathsf{return}]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![([u, S, A_0], \mathsf{return}, u'), A']\!]_{\mathsf{wc}}\, \eta^{i-1} \quad \subseteq \quad (\eta'_{\mathsf{wc}}, \eta'_{\mathsf{wc}}\,[u', S, A'])$$

We have

$$\begin{array}{lcl}
[\![([u, S, A_0], \mathsf{return}, u'), A']\!]_{\mathsf{wc}}\, \eta & = & \textbf{let } T = [\![(u, \mathsf{return}, u')]\!]_{\mathcal{T}}(\eta\,[u, S, A_0]) \textbf{ in} \\
& & \textbf{let } \rho = \{[i, A'] \mapsto \{t \mid t \in T, \mathsf{id}\, t = i\} \mid i \in \mathcal{V}_{\mathbf{tid}}\} \textbf{ in} \\
& & (\rho, T) \\
[\![[u, S, A_0], \mathsf{return}, A']\!]^\sharp \eta^\sharp & = & \textbf{let } (W^\sharp, P^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\
& & \textbf{let } I = \sigma^\sharp\, \mathsf{self} \textbf{ in} \\
& & \textbf{let } v_\rho^\sharp = \sigma^\sharp\, \mathsf{ret} \textbf{ in} \\
& & \textbf{let } \rho^\sharp = \left\{[i, A'] \mapsto v_\rho^\sharp \mid i \in I\right\} \textbf{ in} \\
& & (\rho^\sharp, (W^\sharp, P^\sharp, \sigma^\sharp))
\end{array}$$

Let $\eta^\sharp\,[u, S, A_0] = (W^\sharp, P^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u', S, A'] = (W^{\sharp'}, P^{\sharp'}, \sigma^{\sharp'})$ the value provided by $\mathcal{C}^\sharp$ for the end point of the given control-flow edge and lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $W^\sharp \sqsubseteq W^{\sharp'}$, $P^\sharp \sqsubseteq P^{\sharp'}$, and $\sigma^\sharp \sqsubseteq \sigma^{\sharp'}$ hold. Then, by definition:

$$\begin{array}{lcl}
& & \eta'_{\mathsf{wc}}[u', S, A'] = \gamma_{u', S, A'}(W^{\sharp'}, P^{\sharp'}, \sigma^{\sharp'}) \\
& = & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = S, \alpha_{\mathcal{A}}\, t = A', \beta\, t = (W, P, \sigma), \\
& & \qquad \sigma \subseteq \bar\gamma \circ \sigma^{\sharp'}, W \sqsubseteq W^{\sharp'}, P \sqsubseteq P^{\sharp'}\}
\end{array}$$

For every trace $t \in \eta^{i-1}\,[u, S, A_0]$, let $\beta\, t = (W, P, \sigma)$. By induction hypothesis, $W \sqsubseteq W^\sharp$, $P \sqsubseteq P^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let $t' = [\![(u, \mathsf{return}, u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S$, $\alpha_{\mathcal{A}}\, t' = A'$, and

$$\begin{array}{lcl}
\beta\, t' & = & (W', P', \sigma') \text{ where} \\
W' & = & \{g \mapsto \{L_{t'}[\bar{v}']\} \mid g \in \mathcal{G}, (_, g = x, \bar{v}') = \mathsf{last_tl_write}_g\, t'\} \\
 & & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t'\} = W \\
P' & = & \{g \mapsto \mathsf{min_lockset_since}\, t'\, \bar{v}' \mid g \in \mathcal{G}, (_, g = x, \bar{v}') = \mathsf{last_tl_write}_g\, t'\} \\
 & & \cup\, \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t'\} = P \\
\sigma' & = & \{x \mapsto \{t'(x)\} \mid x \in \mathcal{X}\} \\
 & & \cup\, \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\, t'\} \\
 & & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t', \bot \neq \mathsf{last_write}_g\, t'\} \\
 & & \cup\, \{g \mapsto \{\sigma_{j-1}\, x'\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x', _) = \mathsf{last_tl_write}_g\, t'\} \\
 & = & \sigma
\end{array}$$

Thus $W' = W \sqsubseteq W^\sharp \sqsubseteq W^{\sharp'}$, $P' = P \sqsubseteq P^\sharp \sqsubseteq P^{\sharp'}$, and $\sigma' = \sigma \subseteq \bar{\gamma} \circ \sigma^\sharp \subseteq \bar{\gamma} \circ \sigma^{\sharp'}$. Altogether, $t' \in \eta'_{\mathsf{wc}}\,[u', S, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], \mathsf{return}, u'), A']\!]_{\mathsf{wc}}\, \eta^{i-1}$ is subsumed by the value $\eta'_{\mathsf{wc}}\,[u', S, A']$.

Next, we consider the side-effects of the corresponding right-hand-side functions for local traces t and t' as considered previously

$$\begin{array}{lcl}
\rho & = & \{[\mathsf{id}\, t', A'] \mapsto \{t'\}\} \\
\rho^\sharp & = & \left\{[i^\sharp, A'] \mapsto v^\sharp_\rho \mid i^\sharp \in \sigma^\sharp\, \mathsf{self}\right\}
\end{array}$$

where we use that $\mathsf{id}\, t = \mathsf{id}\, t'$. As $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$, we have $\{\mathsf{id}\, t'\} = \sigma\, \mathsf{self} \subseteq \gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}(\sigma^\sharp\, \mathsf{self})$. As $\mathcal{V}^\sharp_{\mathbf{tid}}$ is a powerset lattice, and the concretization is defined by the union of concretizations of the singleton sets, there is at least one $i^\sharp_\rho \in \sigma^\sharp\, \mathsf{self}$, such that $\mathsf{id}\, t' \in \gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}\{i^\sharp_\rho\}$. Consider one such $i^\sharp_\rho$, and let $\eta^\sharp\,[i^\sharp_\rho, A'] = v^{\sharp'}_\rho$ the value provided by $\mathcal{C}^\sharp$ for this unknown receiving the side-effect. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $v^\sharp_\rho \sqsubseteq v^{\sharp'}_\rho$ holds. By definition:

$$\eta'_{\mathsf{wc}}[\mathsf{id}\, t', A'] = \bigcup\{\gamma_{(\mathsf{id}\, t'), A'}(\eta^\sharp\,[i^\sharp, A']) \mid i^\sharp \in S_{\mathcal{V}^\sharp_{\mathbf{tid}}}, \mathsf{id}\, t' \in (\gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}\{i^\sharp\})\}$$

Now consider

$$\gamma_{(\mathsf{id}\, t'), A'}(\eta^\sharp\,[i^\sharp_\rho, A']) \quad \subseteq \quad \eta'_{\mathsf{wc}}[\mathsf{id}\, t', A']$$

Then, by definition:

$$\begin{array}{lcl}
 & & \gamma_{(\mathsf{id}\, t'), A'}(\eta^\sharp\,[i^\sharp_\rho, A']) = \gamma_{(\mathsf{id}\, t'), A'}\left(v^{\sharp'}_\rho\right) \\
 & = & \{t'' \in \mathcal{T} \mid \mathsf{last}\, t'' = \mathsf{return}, \alpha_{\mathcal{A}}\, t'' = A', \mathsf{id}\, t'' = \mathsf{id}\, t', t''(\mathsf{ret}) \in \gamma_{\mathcal{V}^\sharp}(v^{\sharp'}_\rho)\}
\end{array}$$

We have $\mathsf{last}\, t' = \mathsf{return}$, $\alpha_{\mathcal{A}}\, t' = A'$, (vacuously $\mathsf{id}\, t' = \mathsf{id}\, t'$), and

$$t'(\mathsf{ret}) \in \sigma\, \mathsf{ret} \subseteq (\bar{\gamma} \circ \sigma^\sharp)\, \mathsf{ret} \subseteq \gamma_{\mathcal{V}^\sharp}((\sigma^\sharp)\, \mathsf{ret}) = \gamma_{\mathcal{V}^\sharp}(v^\sharp_\rho) \subseteq \gamma_{\mathcal{V}^\sharp}(v^{\sharp'}_\rho)$$

Altogether, $t' \in \eta'_{\mathsf{wc}}\,[\mathsf{id}\,t, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. Hence, all side-effects for return of $\mathcal{C}_{\mathsf{wc}}$ are accounted for in η'_{wc}, and the claim holds.

Next, consider the constraints corresponding to calling **join**. Consider an edge $(u, x = \mathsf{join}(x'), u') \in \mathcal{E}$ and digests A', A_0 and all appropriate A_1 such that $A' \in [\![u, x = \mathsf{join}(x')]\!]^\sharp_{\mathcal{A}}(A_0, A_1)$. We verify that

$$[\![([u, S, A_0], x = \mathsf{join}(x'), u')]\!]_{\mathsf{wc}}\,\eta^{i-1} \quad \subseteq \quad (\eta'_{\mathsf{wc}}, \eta'_{\mathsf{wc}}\,[u', S, A'])$$

We have

$$\begin{array}{l}
[\![([u, S, A_0], x = \mathsf{join}(x'), u')]\!]_{\mathsf{wc}}\,\eta = \\
\quad \textbf{let } T_1 = \bigcup\{\eta\,[t(x'), A_1] \mid t \in \eta\,[u, S, A_0], A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1\} \textbf{ in} \\
\quad \textbf{let } T = [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\eta\,[u, S, A_0], T_1) \textbf{ in} \\
\quad (\emptyset, T) \\
[\![[u, S, A_0], x = \mathsf{join}(x')]\!]^\sharp \eta^\sharp = \\
\quad \textbf{let } (W^\sharp, P^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\
\quad \textbf{let } v^\sharp = \bigsqcup_{i' \in (\sigma^\sharp\,x')} \left(\bigsqcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1} (\eta^\sharp[i', A_1])\right) \textbf{ in} \\
\quad \textbf{if } v^\sharp = \bot \textbf{ then} \\
\qquad (\emptyset, \bot) \\
\quad \textbf{else} \\
\qquad \textbf{let } \sigma^{\sharp\prime\prime} = \sigma^\sharp \oplus \{x \mapsto v^\sharp\} \textbf{ in} \\
\qquad (\emptyset, (W^\sharp, P^\sharp, \sigma^{\sharp\prime\prime}))
\end{array}$$

Let $\eta^\sharp\,[u, S, A_0] = (W^\sharp, P^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u', S, A'] = (W^{\sharp\prime}, P^{\sharp\prime}, \sigma^{\sharp\prime})$ the value provided by $\mathcal{C}^\sharp$ for the end point and the given lockset and digest. As $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, either

(1) $v^\sharp = \bot$ holds; or

(2) $W^\sharp \sqsubseteq W^{\sharp\prime}$, $P^\sharp \sqsubseteq P^{\sharp\prime}$, and $\sigma^{\sharp\prime\prime} \sqsubseteq \sigma^{\sharp\prime}$ all hold.

Then, by definition:

$$\begin{array}{rcl}
 & & \eta'_{\mathsf{wc}}[u', S, A'] = \gamma_{u', S, A'}(W^{\sharp\prime}, P^{\sharp\prime}, \sigma^{\sharp\prime}) \\
 & = & \{t \in \mathcal{T} \mid \mathsf{loc}\,t = u', L_t = S, \alpha_{\mathcal{A}}\,t = A', \beta\,t = (W, P, \sigma), \\
 & & \qquad \sigma \subseteq \bar{\gamma} \circ \sigma^{\sharp\prime}, W \sqsubseteq W^{\sharp\prime}, P \sqsubseteq P^{\sharp\prime}\}
\end{array}$$

For every trace $t \in \eta^{i-1}\,[u, S, A_0]$, let $\beta\,t = (W, P, \sigma)$. By induction hypothesis, $W \sqsubseteq W^\sharp$, $P \sqsubseteq P^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let

$$\begin{array}{rcl}
T' & = & [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\{t\}, \\
 & & \qquad \bigcup\{\eta^{i-1}\,[(t'(x')), A_1] \mid t' \in \eta^{i-1}\,[u, S, A_0], A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1\}) \\
 & = & [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\{t\}, \bigcup\{\eta^{i-1}\,[(t(x')), A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1\})
\end{array}$$

where the equality exploits that for a given local trace t, $[\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\{t\}, \{t'\})$ only yields a non-empty set if the thread *id* of the thread being joined is the one stored

in x'. We distinguish the case where the resulting set of traces is non-empty and the case where it is empty. If T' is empty, it is subsumed by $\eta'_{\mathsf{wc}}\,[u', S, A']$ vacuously. Consider thus a $t' \in T'$ and $t'' \in \bigcup\{\eta^{i-1}\,[(t(x')), A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\}$ such that $\{t'\} = [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\{t\}, \{t''\})$ Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S$, $\alpha_{\mathcal{A}}\, t' = A'$, and

$$
\begin{array}{rcl}
\beta\, t' & = & (W', P', \sigma') \text{ where} \\
W' & = & \{g \mapsto \{L_{t'}[\bar{v}']\} \mid g \in \mathcal{G}, (\_, g = x'', \bar{v}') = \mathsf{last\_tl\_write}_g\, t'\} \\
 & & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\, t'\} = W \\
P' & = & \{g \mapsto \mathsf{min\_lockset\_since}\, t'\, \bar{v}' \mid g \in \mathcal{G}, (\_, g = x'', \bar{v}') = \mathsf{last\_tl\_write}_g\, t'\} \\
 & & \cup\, \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\, t'\} = P \\
\sigma' & = & \{x'' \mapsto \{t'(x'')\} \mid x'' \in \mathcal{X}\} \cup\ \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last\_write}_g\, t'\} \\
 & & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\, t', \bot \neq \mathsf{last\_write}_g\, t'\} \\
 & & \cup\, \{g \mapsto \{\sigma_{j-1}\, x''\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x'', \_) = \mathsf{last\_tl\_write}_g\, t'\} \\
 & = & \sigma \oplus (\{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\, t', \bot \neq \mathsf{last\_write}_g\, t'\} \\
 & & \cup \{x \mapsto t''(\mathsf{ret})\}) \\
 & \subseteq & \sigma \oplus \{x \mapsto t''(\mathsf{ret})\}
\end{array}
$$

Thus $W' = W \sqsubseteq W^\sharp$ and $P' = P \sqsubseteq P^\sharp$. To show that $t' \in \eta'_{\mathsf{wc}}\,[u', S, A']$, it thus remains to show that $v^\sharp \neq \bot$ holds (and thus $W^\sharp \sqsubseteq W^{\sharp'}$, $P^\sharp \sqsubseteq P^{\sharp'}$ also hold) and to relate $\sigma \oplus \{x \mapsto t''(\mathsf{ret})\}$ to $\bar{\gamma} \circ \sigma^{\sharp'}$. To this end, we first relate $t''(\mathsf{ret})$ to $v^\sharp$. We have

$$
\begin{array}{rcl}
t'' & \in & \bigcup\{\eta^{i-1}\,[t(x'), A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\} \\
 & \subseteq & \bigcup\{\eta'_{\mathsf{wc}}\,[t(x'), A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\} \\
 & = & \bigcup\{\bigcup\{\gamma_{t(x'),A_1}(\eta^\sharp\,[i^\sharp, A_1]) \mid t(x') \in (\gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}\,\{i^\sharp\})\} \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\} \\
 & \subseteq & \bigcup\{\bigcup\{\gamma_{t(x'),A_1}(\eta^\sharp\,[i^\sharp, A_1]) \mid i^\sharp \in \sigma^\sharp\, x'\} \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\}
\end{array}
$$

and therefore

$$
\begin{array}{rcl}
 & & \{t''(\mathsf{ret})\} \\
 & \subseteq & \{t'''(\mathsf{ret}) \mid t''' \in (\bigcup\{\bigcup\{\gamma_{t(x'),A_1}(\eta^\sharp\,[i^\sharp, A_1]) \mid i^\sharp \in \sigma^\sharp\, x'\} \\
 & & \qquad \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\})\} \\
 & \subseteq & \{t'''(\mathsf{ret}) \mid t''' \in (\bigcup\{\gamma_{t(x'),A_1} \bigsqcup\{(\eta^\sharp\,[i^\sharp, A_1]) \mid i^\sharp \in \sigma^\sharp\, x'\} \\
 & & \qquad \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\})\} \\
 & \subseteq & \{t'''(\mathsf{ret}) \mid t''' \in (\bigcup\{\{\bar{t} \in \mathcal{T} \mid \mathsf{last}\,(\bar{t}) = \mathsf{return}, \alpha_{\mathcal{A}}\,(\bar{t}) = A_1, \mathsf{id}\,\bar{t} = t(x'), \\
 & & \qquad \bar{t}(\mathsf{ret}) \in \gamma_{\mathcal{V}^\sharp}(\bigsqcup\{(\eta^\sharp\,[i^\sharp, A_1]) \mid i^\sharp \in \sigma^\sharp\, x'\})\} \\
 & & \qquad \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\})\} \qquad \text{(by def. of } \gamma_{t(x'),A_1}) \\
 & \subseteq & \bigcup\{\{\bar{t}(\mathsf{ret}) \mid \bar{t} \in \mathcal{T}, \mathsf{last}\,(\bar{t}) = \mathsf{return}, \alpha_{\mathcal{A}}\,(\bar{t}) = A_1, \mathsf{id}\,\bar{t} = t(x'), \\
 & & \qquad \bar{t}(\mathsf{ret}) \in \gamma_{\mathcal{V}^\sharp}(\bigsqcup\{(\eta^\sharp\,[i^\sharp, A_1]) \mid i^\sharp \in \sigma^\sharp\, x'\})\} \\
 & & \qquad \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\} \\
 & \subseteq & \bigcup\{\gamma_{\mathcal{V}^\sharp}(\bigsqcup\{(\eta^\sharp\,[i^\sharp, A_1]) \mid i^\sharp \in \sigma^\sharp\, x'\}) \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\} \\
 & \subseteq & \gamma_{\mathcal{V}^\sharp}(\bigsqcup\{\bigsqcup\{(\eta^\sharp\,[i^\sharp, A_1]) \mid i^\sharp \in \sigma^\sharp\, x'\} \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\}) \\
 & \subseteq & \gamma_{\mathcal{V}^\sharp}\left(\bigsqcup_{i^\sharp \in \sigma^\sharp\, x'}\left(\bigsqcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1}(\eta[i^\sharp, A_1])\right)\right) \\
 & = & \gamma_{\mathcal{V}^\sharp}(v^\sharp)
\end{array}
$$

As a consequence and as $\gamma_{\mathcal{V}^\sharp}(\bot) = \emptyset$, we obtain $v^\sharp \neq \bot$. Thus, we also have $W^\sharp \sqsubseteq W^{\sharp\prime}$, $P^\sharp \sqsubseteq P^{\sharp\prime}$, as well as $\sigma^{\sharp\prime\prime} \sqsubseteq \sigma^{\sharp\prime}$. Also

$$\sigma' \subseteq \sigma \oplus \{x \mapsto t''(\mathsf{ret})\} \subseteq (\bar{\gamma} \circ \sigma^\sharp) \oplus \{x \mapsto t''(\mathsf{ret})\} \subseteq \bar{\gamma} \circ (\sigma^\sharp \oplus \{x \mapsto v^\sharp\}) = \bar{\gamma} \circ \sigma^{\sharp\prime\prime}$$

Thus, $W' = W \sqsubseteq W^\sharp \sqsubseteq W^{\sharp\prime}$, $P' = P \sqsubseteq P^\sharp \sqsubseteq P^{\sharp\prime}$, and $\sigma' \subseteq \bar{\gamma} \circ \sigma^{\sharp\prime\prime} \subseteq \bar{\gamma} \circ \sigma^{\sharp\prime}$. Altogether, $t' \in \eta'_{\mathsf{wc}}\,[u', S, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], x = \mathsf{join}(x'), u'), A_1]\!]_{\mathsf{wc}}\,\eta^{i-1}$ is subsumed by the value $\eta'_{\mathsf{wc}}\,[u', S, A']$ in both cases. As neither constraint causes any side-effects, the statement holds.

Next, consider the constraints corresponding to calling **signal**. Consider an edge $(u, \mathsf{signal}(s), u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, \mathsf{signal}(s)]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![([u, S, A_0], \mathsf{signal}(s), u'), A']\!]_{\mathsf{wc}}\,\eta^{i-1} \quad \subseteq \quad (\eta'_{\mathsf{wc}}, \eta'_{\mathsf{wc}}\,[u', S, A'])$$

We have

$$\begin{array}{l}
[\![([u, S, A_0], \mathsf{signal}(s), u'), A']\!]_{\mathsf{wc}}\,\eta = \\
\quad \textbf{let } T = [\![(u, \mathsf{signal}(s), u')]\!]_{\mathcal{T}}(\eta\,[u, S, A_0]) \textbf{ in} \\
\quad \textbf{let } \rho = \{[s, A'] \mapsto T\} \textbf{ in} \\
\quad (\rho, T)
\end{array}$$

$$\begin{array}{l}
[\![[u, S, A_0], \mathsf{signal}(s), A']\!]^\sharp \eta^\sharp = \\
\quad \textbf{let } (W^\sharp, P^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\
\quad \textbf{let } \rho^\sharp = \left\{[s, A'] \mapsto (W^\sharp, P^\sharp, \sigma^\sharp)\right\} \textbf{ in} \\
\quad \left(\rho^\sharp, (W^\sharp, P^\sharp, \sigma^\sharp)\right)
\end{array}$$

Let $\eta^\sharp\,[u, S, A_0] = (W^\sharp, P^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u', S, A'] = (W^{\sharp\prime}, P^{\sharp\prime}, \sigma^{\sharp\prime})$ the value provided by $\mathcal{C}^\sharp$ for the end point of the given control-flow edge and lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $W^\sharp \sqsubseteq W^{\sharp\prime}$, $P^\sharp \sqsubseteq P^{\sharp\prime}$, and $\sigma^\sharp \sqsubseteq \sigma^{\sharp\prime}$ hold. Then, by definition:

$$\begin{array}{rl}
& \eta'_{\mathsf{wc}}[u', S, A'] = \gamma_{u', S, A'}(W^{\sharp\prime}, P^{\sharp\prime}, \sigma^{\sharp\prime}) \\
= & \{t \in \mathcal{T} \mid \mathsf{loc}\,t = u', L_t = S, \alpha_{\mathcal{A}}\,t = A', \beta\,t = (W, P, \sigma), \\
& \qquad \sigma \subseteq \bar{\gamma} \circ \sigma^{\sharp\prime}, W \sqsubseteq W^{\sharp\prime}, P \sqsubseteq P^{\sharp\prime}\}
\end{array}$$

For every trace $t \in \eta^{i-1}\,[u, S, A_0]$, let $\beta\,t = (W, P, \sigma)$. By induction hypothesis, $W \sqsubseteq W^\sharp$, $P \sqsubseteq P^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let $t' = [\![(u, \mathsf{signal}(s), u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\,t' = u'$, $L_{t'} = S$, $\alpha_{\mathcal{A}}\,t' = A'$, and

$$\begin{array}{rcl}
\beta\,t' & = & (W', P', \sigma') \text{ where} \\
W' & = & \{g \mapsto \{L_{t'}[\bar{v}']\} \mid g \in \mathcal{G}, (_, g = x, \bar{v}') = \mathsf{last_tl_write}_g\,t'\} \\
& & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\,t'\} \\
& = & W \\
P' & = & \{g \mapsto \mathsf{min_lockset_since}\,t'\,\bar{v}' \mid g \in \mathcal{G}, (_, g = x, \bar{v}') = \mathsf{last_tl_write}_g\,t'\} \\
& & \cup\, \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\,t'\} \\
& = & P
\end{array}$$

$$\begin{aligned}\sigma' &= \{x \mapsto \{t'(x)\} \mid x \in \mathcal{X}\} \\ &\quad \cup \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last\_write}_g\, t'\} \\ &\quad \cup \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\, t', \bot \neq \mathsf{last\_write}_g\, t'\} \\ &\quad \cup \{g \mapsto \{\sigma_{j-1}\, x'\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x', \_) = \mathsf{last\_tl\_write}_g\, t'\} \\ &= \sigma\end{aligned}$$

Thus $W' = W \sqsubseteq W^\sharp \sqsubseteq W^{\sharp'}$, $P' = P \sqsubseteq P^\sharp \sqsubseteq P^{\sharp'}$, and $\sigma' = \sigma \subseteq \bar{\gamma} \circ \sigma^\sharp \subseteq \bar{\gamma} \circ \sigma^{\sharp'}$. Altogether, $t' \in \eta'_{\mathsf{wc}}\,[u', S, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], \mathsf{signal}(s), u'), A']\!]_{\mathsf{wc}}\, \eta^{i-1}$ is subsumed by the value $\eta'_{\mathsf{wc}}\,[u', S, A']$.

Next, we consider the side-effects of the corresponding right-hand-side functions for local traces t and t' as considered previously.

$$\begin{aligned}\rho &= \{[s, A'] \mapsto T\} \\ \rho^\sharp &= \{[s, A'] \mapsto (W^\sharp, P^\sharp, \sigma^\sharp)\}\end{aligned}$$

Let $\eta^\sharp\,[s, A'] = (W^\sharp_\rho, P^\sharp_\rho, \sigma^\sharp_\rho)$ the value provided by $\mathcal{C}^\sharp$ for the unknown receiving the side-effect. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $W^\sharp \sqsubseteq W^\sharp_\rho$, $P^\sharp \sqsubseteq P^\sharp_\rho$, and $\sigma^\sharp \sqsubseteq \sigma^\sharp_\rho$ hold. By definition:

$$\begin{aligned}&\eta'_{\mathsf{wc}}[s, A'] = \gamma_{s,A'}(\eta^\sharp\,[s, A']) \\ = \;& \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{signal}(s), \alpha_{\mathcal{A}}\, t = A', \\ &\qquad \beta\, t = (W, P, \sigma), \sigma \subseteq \bar{\gamma} \circ \sigma^\sharp_\rho, W \sqsubseteq W^\sharp_\rho, P \sqsubseteq P^\sharp_\rho\}\end{aligned}$$

Consider a trace t' as above: Then $\mathsf{last}\, t' = \mathsf{signal}(s)$, $\alpha_{\mathcal{A}}\, t' = A'$, and $W' = W \sqsubseteq W^\sharp \sqsubseteq W^\sharp_\rho$, $P' = P \sqsubseteq P^\sharp \sqsubseteq P^\sharp_\rho$, and $\sigma' = \sigma \subseteq \bar{\gamma} \circ \sigma^\sharp \subseteq \bar{\gamma} \circ \sigma^\sharp_\rho$ Altogether, $t' \in \eta'_{\mathsf{wc}}\,[s, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. Hence, all side-effects for $\mathsf{signal}(s)$ of $\mathcal{C}_{\mathsf{wc}}$ are accounted for in η'_{wc}, and the claim holds.

Lastly, consider the constraints corresponding to a call of **wait**. Consider an edge $(u, \mathsf{wait}(s), u') \in \mathcal{E}$ and digests A', A_0 and all appropriate A_1 such that the resulting digest $A' \in [\![u, \mathsf{wait}(s)]\!]^\sharp_{\mathcal{A}}(A_0, A_1)$. We verify that

$$[\![([u, S, A_0], \mathsf{wait}(s), u')]\!]_{\mathsf{wc}}\, \eta^{i-1} \quad \subseteq \quad (\eta'_{\mathsf{wc}}, \eta'_{\mathsf{wc}}\,[u', S, A'])$$

We have

$$\begin{aligned}&[\![([u, S, A_0], \mathsf{wait}(s), u')]\!]_{\mathsf{wc}}\, \eta = \\ &\quad \textbf{let } T_1 = \bigcup\{\eta\,[s, A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\} \textbf{ in} \\ &\quad \textbf{let } T = [\![(u, \mathsf{wait}(s), u')]\!]_{\mathcal{T}}(\eta\,[u, S, A_0], T_1) \textbf{ in} \\ &\quad (\emptyset, T)\end{aligned}$$

$$\begin{aligned}&[\![[u, S, A_0], \mathsf{wait}(s)]\!]^\sharp \eta^\sharp = \\ &\quad \textbf{let } (W^\sharp, P^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\ &\quad \textbf{if } \left(\left(\bigsqcup_{A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'} \eta^\sharp\,[s, A']\right) = \underline{\bot}\right) \textbf{ then} \\ &\qquad (\emptyset, \underline{\bot}) \\ &\quad \textbf{else} \\ &\qquad (\emptyset, (W^\sharp, P^\sharp, \sigma^\sharp))\end{aligned}$$

Let $\eta^\sharp\,[u, S, A_0] = (W^\sharp, P^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u', S, A'] = (W^{\sharp'}, P^{\sharp'}, \sigma^{\sharp'})$ the value provided by $\mathcal{C}^\sharp$ for the end point of the given control-flow edge and lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, we either have

(1) $\left(\bigsqcup_{A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'} \eta^\sharp\,[s, A']\right) = \bot$; or

(2) $W^\sharp \sqsubseteq W^{\sharp'}$, $P^\sharp \sqsubseteq P^{\sharp'}$, and $\sigma^\sharp \sqsubseteq \sigma^{\sharp'}$ all hold.

By definition, we have:

$$\begin{array}{rcl} & \eta'_{\mathsf{wc}}[u', S, A'] = \gamma_{u', S, A'}(W^{\sharp'}, P^{\sharp'}, \sigma^{\sharp'}) \\ = & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = S, \alpha_{\mathcal{A}}\, t = A', \beta\, t = (W, P, \sigma), \\ & \qquad \sigma \subseteq \bar{\gamma} \circ \sigma^{\sharp'}, W \sqsubseteq W^{\sharp'}, P \sqsubseteq P^{\sharp'}\} \end{array}$$

For every trace $t \in \eta^{i-1}\,[u, S, A_0]$, let $\beta\, t = (W, P, \sigma)$. By induction hypothesis, $W \sqsubseteq W^\sharp$, $P \sqsubseteq P^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let

$$T' \quad = \quad [\![(u, \mathsf{wait}(s), u')]\!]_{\mathcal{T}}(\{t\}, \bigcup\{\eta^{i-1}\,[s, A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\})$$

We distinguish the case where the resulting set of traces is non-empty and the case where it is empty. If T' is empty, it is subsumed by $\eta'_{\mathsf{wc}}\,[u', S, A']$ vacuously. Consider thus a $t' \in T'$ and $t'' \in \bigcup\{\eta^{i-1}\,[s, A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\}$ such that $\{t'\} = [\![(u, \mathsf{wait}(s), u')]\!]_{\mathcal{T}}(\{t\}, \{t''\})$.

By induction hypothesis, we have

$$\begin{array}{rcl} t'' & \in & \bigcup\{\eta^{i-1}\,[s, A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\} \\ & \subseteq & \bigcup\{\eta'_{\mathsf{wc}}\,[s, A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\} \\ & = & \bigcup\{\gamma_{s, A_1}(\eta^\sharp\,[s, A_1]) \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\} \end{array}$$

and thus as $\gamma_{s, A'}(\bot) = \emptyset$ for any $A' \in \mathcal{A}$, there is at least one $A' \in \mathcal{A}$ for which $\mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'$ holds and where $\eta^\sharp\,[s, A'] \neq \bot$. Thus, $\left(\bigsqcup_{A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'} \eta^\sharp\,[s, A']\right) \neq \bot$ and (1) does not hold. We thus have that (2) $W^\sharp \sqsubseteq W^{\sharp'}$, $P^\sharp \sqsubseteq P^{\sharp'}$, and $\sigma^\sharp \sqsubseteq \sigma^{\sharp'}$ all hold.

Consider again the trace t'. Then, $\mathsf{loc}\, t' = u'$, $L_{t'} = S$, $\alpha_{\mathcal{A}}\, t' = A'$, and

$$\begin{array}{rcl} \beta\, t' & = & (W', P', \sigma') \text{ where} \\ W' & = & \{g \mapsto \{L_{t'}[\bar{v}']\} \mid g \in \mathcal{G}, (\_, g = x, \bar{v}') = \mathsf{last\_tl\_write}_g\, t'\} \\ & & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\, t'\} = W \\ P' & = & \{g \mapsto \mathsf{min\_lockset\_since}\, t'\, \bar{v}' \mid g \in \mathcal{G}, (\_, g = x, \bar{v}') = \mathsf{last\_tl\_write}_g\, t'\} \\ & & \cup\, \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\, t'\} = P \\ \sigma' & = & \{x \mapsto \{t'(x)\} \mid x \in \mathcal{X}\} \\ & & \cup\, \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last\_write}_g\, t'\} \\ & & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\, t', \bot \neq \mathsf{last\_write}_g\, t'\} \\ & & \cup\, \{g \mapsto \{\sigma_{j-1}\, x'\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x', \_) = \mathsf{last\_tl\_write}_g\, t'\} \\ & = & \sigma \oplus \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\, t', \bot \neq \mathsf{last\_write}_g\, t'\} \\ & \subseteq & \sigma \end{array}$$

Thus, $W = W' \sqsubseteq W^\sharp \sqsubseteq W^{\sharp\prime}$, $P = P' \sqsubseteq P^\sharp \sqsubseteq P^{\sharp\prime}$, and $\sigma' \subseteq \sigma \subseteq \bar{\gamma} \circ \sigma^\sharp \subseteq \bar{\gamma} \circ \sigma^{\sharp\prime}$. Altogether, $t' \in \eta'_{\mathsf{wc}}\,[u', S, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$ and $t'' \in \bigcup\{\eta^{i-1}\,[s, A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1\}$. As neither constraint causes any side-effects, the statement holds for edges corresponding to calls of $\mathsf{wait}(s)$.

This concludes the case distinction for the inductive step and thus the soundness proof for Write-Centered Reading with ego-lane digests. □

6.1.2 Lock-Centered Reading

Consider some ego-lane digest $\mathcal{A}$ and an appropriate definition of $\mathsf{compat}^\sharp_{\mathcal{A}}$. Let us refer to the constraint system of the analysis from Section 4.1.5 refined with $\mathcal{A}$ by $\mathcal{C}^\sharp_{\mathsf{lc}}$. As a first step, we construct a new constraint system $\mathcal{C}_{\mathsf{lc}}$ over sets of local traces for which the unknowns match those of $\mathcal{C}^\sharp_{\mathsf{lc}}$ — up to the unknowns used for thread returns. We then show the equivalence of $\mathcal{C}_{\mathsf{lc}}$ and $\mathcal{C}$ by showing $\mathcal{C}_{\mathsf{lc}}$ to be equivalent to $\mathcal{C}_{\mathsf{wc}}$ for which we have shown the equivalence in Proposition 16.

$\mathcal{C}_{\mathsf{lc}}$ then has the following unknowns:

- $[u, S, A]$ for $u \in \mathcal{N}$, $S \subseteq \mathcal{M}$ and $A \in \mathcal{A}$,
- $[\mathbf{g}, \mathbf{a}, \mathbf{S}, \mathbf{A}]$ for $g \in \mathcal{G}, a \in \mathcal{M}, S \subseteq \mathcal{M}$, and $A \in \mathcal{A}$,
- $[i, A]$ for $i \in \mathcal{V}_{\mathbf{tid}}$ and $A \in \mathcal{A}$, and
- $[s, A]$ for $s \in \mathcal{S}$ and $A \in \mathcal{A}$.

where unknowns different from the ones used in $\mathcal{C}_{\mathsf{wc}}$ are highlighted in bold. The constraints of $\mathcal{C}_{\mathsf{lc}}$ are then the same as in $\mathcal{C}_{\mathsf{wc}}$ with the following deviations for the right-hand sides:

$$
\begin{aligned}
&[\![([u, S, A_0], \mathsf{lock}(a), u')]\!]_{\mathsf{lc}}\,\eta_{\mathsf{lc}} = \\
&\quad \textbf{let } T_1 = \bigcup\{\eta_{\mathsf{lc}}\,[g, a, S', A_1] \mid g \in \mathcal{G}, S' \subseteq \mathcal{M}, A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1\} \textbf{ in} \\
&\quad \textbf{let } T_2 = \{t' \mid t \in \eta_{\mathsf{lc}}\,[u, S, A_0], \bar{i} = \mathsf{init\_v}\,t, \bar{i} \neq \bot, t' = \downarrow_{\bar{i}}\,(t), \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,(\alpha_{\mathcal{A}}\,t')\} \textbf{ in} \\
&\quad \textbf{let } T = [\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{lc}}\,[u, S, A_0], T_1 \cup T_2) \textbf{ in} \\
&\quad (\emptyset, T)
\end{aligned}
$$

$$
\begin{aligned}
&[\![([u, S, A_0], \mathsf{unlock}(a), u'), A']\!]_{\mathsf{lc}}\,\eta_{\mathsf{lc}} = \\
&\quad \textbf{let } T = [\![(u, \mathsf{unlock}(a), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{lc}}\,[u, S, A_0]) \textbf{ in} \\
&\quad \textbf{let } \rho = \{[g, a, S \setminus \{a\}, A'] \mapsto T \mid g \in \mathcal{G}\} \textbf{ in} \\
&\quad (\rho, T)
\end{aligned}
$$

Thus, the splitting according to w performed in $\mathcal{C}_{\mathsf{wc}}$ is not performed here. In order to relate least solutions of $\mathcal{C}_{\mathsf{lc}}$ to least solutions of $\mathcal{C}_{\mathsf{wc}}$, we first establish that a least solution of $\mathcal{C}_{\mathsf{lc}}$ exists and that it can be attained as the least upper bound of all Kleene iterates.

Proposition 19. *The right-hand side function of constraint system $\mathcal{C}_{\mathsf{lc}}$ over the lattice mapping (extended) unknowns to sets of local traces with the order as discussed in Section 2.2.2 is Scott-continuous.*

Proof. The proof here proceeds in the same matter as the proof of Proposition 14 in Section 6.1.1. The two differing right-hand sides are also compositions of Scott-continuous functions, and thus Scott-continuous. Then, the combined right-hand side collecting all constraints is given as the least upper bound of (compositions of) functions that are Scott-continuous, and is thus also Scott-continuous. □

Proposition 20. *The constraint system $\mathcal{C}_{lc}$ has a least solution which is obtained as the least-upper bound of all Kleene iterates.*

To relate solutions of both systems, we define for a mapping η_{wc} from the unknowns of $\mathcal{C}_{\mathsf{wc}}$ to sets of local traces, a mapping η_{lc} from the unknowns of $\mathcal{C}_{\mathsf{lc}}$ to sets of local traces by setting

$$\begin{array}{lcll} \eta_{\mathsf{lc}}\,[u,S,A] & = & \eta_{\mathsf{wc}}\,[u,S,A] & (u \in \mathcal{N}, S \subseteq \mathcal{M}, A \in \mathcal{A}) \\ \eta_{\mathsf{lc}}\,[g,a,S,A] & = & \bigcup_{w \subseteq \mathcal{M}} \eta_{\mathsf{wc}}\,[g,a,S,w,A] & (g \in \mathcal{G}, a \in \mathcal{M}, S \subseteq \mathcal{M}, A \in \mathcal{A}) \\ \eta_{\mathsf{lc}}\,[i,A] & = & \eta_{\mathsf{wc}}\,[i,A] & (i \in \mathcal{V}_{\mathbf{tid}}, A \in \mathcal{A}) \\ \eta_{\mathsf{lc}}\,[s,A] & = & \eta_{\mathsf{wc}}\,[s,A] & (s \in \mathcal{S}, A \in \mathcal{A}) \end{array} \tag{6.2}$$

Proposition 21. *The following two statements are equivalent:*

- *η_{wc} is the least solution of $\mathcal{C}_{wc}$;*
- *η_{lc} is the least solution of $\mathcal{C}_{lc}$.*

Proof. The proof is by fixpoint induction. Consider the $i-$th approximation η^i_{wc} to the least solution of $\mathcal{C}_{\mathsf{wc}}$, and the $i-$th approximation η^i_{lc} to the least solution of $\mathcal{C}_{\mathsf{lc}}$. Let us call property (1) that the relationship between η^i_{wc} and η^i_{lc} is as given in Eq. (6.2). For $i = 0$, the value of all unknowns in both constraint systems is $\emptyset$, and property (1) holds trivially. Evaluating constraints that are the same in $\mathcal{C}_{\mathsf{wc}}$ and $\mathcal{C}_{\mathsf{lc}}$ preserves property (1). Thus, the constraints corresponding to locking and unlocking mutexes remain to be considered.

First, consider the constraints corresponding to **lock** for some mutex a. Consider an edge $(u, \mathsf{lock}(a), u') \in \mathcal{E}$ and digests A', A_0, s.t. $\forall A_1 \in \mathcal{A}$, $[\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}(A_0, A_1) \in \{\{A'\}, \emptyset\}$, which exists as $\mathcal{A}$ here is ego-lane-derived and corresponding right-hand sides

$$\begin{array}{l} [\![([u,S,A_0], \mathsf{lock}(a), u')]\!]_{\mathsf{lc}}\,\eta_{\mathsf{lc}} = \\ \quad \textbf{let } T_1 = \bigcup\{\eta_{\mathsf{lc}}\,[g,a,S',A_1] \mid g \in \mathcal{G}, S' \subseteq \mathcal{M}, A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1\} \textbf{ in} \\ \quad \textbf{let } T_2 = \{t' \mid t \in \eta_{\mathsf{lc}}\,[u,S,A_0], \bar{i} = \mathsf{init\_v}\,t, \bar{i} \neq \bot, t' = \downarrow_{\bar{i}}(t), \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,(\alpha_{\mathcal{A}}\,t')\} \textbf{ in} \\ \quad \textbf{let } T = [\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{lc}}\,[u,S,A_0], T_1 \cup T_2) \textbf{ in} \\ \quad (\emptyset, T) \end{array}$$

$$\begin{array}{l} [\![([u,S,A_0], \mathsf{lock}(a), u')]\!]_{\mathsf{wc}}\,\eta_{\mathsf{wc}} = \\ \quad \textbf{let } T_1 = \bigcup\{\eta_{\mathsf{wc}}\,[g,a,S',w,A_1] \mid g \in \mathcal{G}, S' \subseteq \mathcal{M}, w \subseteq \mathcal{M}, A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1\} \textbf{ in} \\ \quad \textbf{let } T_2 = \{t' \mid t \in \eta_{\mathsf{wc}}\,[u,S,A_0], \bar{i} = \mathsf{init\_v}\,t, \bar{i} \neq \bot, t' = \downarrow_{\bar{i}}(t), \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,(\alpha_{\mathcal{A}}\,t')\} \textbf{ in} \\ \quad \textbf{let } T = [\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{wc}}\,[u,S,A_0], T_1 \cup T_2) \textbf{ in} \\ \quad (\emptyset, T) \end{array}$$

The difference between the two right-hand sides is in the computation of T_1. However, when property (1) holds, the resulting sets are identical. Thus, the contributions of $\mathcal{C}_{\mathsf{lc}}$ and $\mathcal{C}_{\mathsf{wc}}$ to the left-hand side are identical if property (1) holds for the i-th approximations and as neither right-hand side causes a side-effect, if property (1) holds for the i-th approximations, and constraints of this form are considered, it also holds for the $(i+1)$-th approximations.

Finally, consider the constraints corresponding to **unlock**, an edge $(u, \mathsf{unlock}(a), u') \in \mathcal{E}$, and digests A', A_0 such that $A' \in [\![u, \mathsf{unlock}(a)]\!]^\sharp_{\mathcal{A}}(A_0)$ and corresponding right-hand sides

$$
\begin{array}{l}
[\![([u, S, A_0], \mathsf{unlock}(a), u'), A']\!]_{\mathsf{lc}}\, \eta_{\mathsf{lc}} = \\
\quad \textbf{let } T = [\![(u, \mathsf{unlock}(a), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{lc}}\,[u, S, A_0]) \textbf{ in} \\
\quad \textbf{let } \rho = \{[g, a, S \setminus \{a\}, A'] \mapsto T \mid g \in \mathcal{G}\} \textbf{ in} \\
\quad (\rho, T)
\end{array}
$$

$$
\begin{array}{l}
[\![([u, S, A_0], \mathsf{unlock}(a), u'), A']\!]_{\mathsf{wc}}\, \eta_{\mathsf{wc}} = \\
\quad \textbf{let } T = [\![(u, \mathsf{unlock}(a), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{wc}}\,[u, S, A_0]) \textbf{ in} \\
\quad \textbf{let } \rho = \{[g, a, S \setminus \{a\}, w, A'] \mapsto \{t\} \mid t \in T, g \in \mathcal{G}, w \subseteq \mathcal{M}, \\
\qquad ((\mathsf{last\_tl\_write}_g\, t = (\bar{u}, g = x, \bar{u}') \wedge L_t[\bar{u}] \subseteq w) \vee (\mathsf{last\_tl\_write}_g\, t = \bot))\} \\
\quad \textbf{in} \\
\quad (\rho, T)
\end{array}
$$

When property (1) holds, the contributions to the left-hand sides are identical. It thus remains to consider the side-effects. With the observation that each $t \in T$ is side-effected to at least one $[g, a, S \setminus \{a\}, w, A']$ in $\mathcal{C}_{\mathsf{wc}}$, we conclude that if property (1) holds for the i-th approximations and constraints of this form are considered, it also holds for the $(i+1)$-th approximations.

This concludes the case distinction and thus the proof of Proposition 21. □

The next proposition needed for the soundness proof of Lock-Centered Reading indicates that the new unknown $[g, a, S, A]$ collects a superset of local traces whose last write to the global g can be read by a thread satisfying the specific assumptions (L0) through (L3) below.

Proposition 22. *Consider the i-th approximation η^i_{lc} to the least solution η_{lc} of constraint system $\mathcal{C}_{\mathsf{lc}}$, a control-flow edge $(u, x = g, u')$ of the program, and a local trace $t \in \eta^i_{\mathsf{lc}}[u', S, A]$ in which the last action is $x = g$, that ends in $\bar{u}' = (j, u', \sigma)$, i.e., $t = (\bar{u}') \downarrow_t$.*
For every mutex $a \in \mathcal{M}$, let $L\,a$ denote the singleton set containing the background lockset of the ego thread at the last thread-local lock of a, given that the ego thread has ever acquired a in t, and set $L\,a = \emptyset$ otherwise. Also, for every mutex a, let $V\,a$ the set of globals written by the ego thread since a was last acquired by it, or all globals written since the start of the ego thread in case it has never acquired a.
Then, the value $d = \sigma\, x$ that is read for g,

- *is the initial value* 0 *of g; or*

- *is produced by a write to g that is the last thread-local write to g in t; or*
- *is produced by a write to g that is the last* thread-local *write to g in some local trace stored at $\eta^{i'}\,[g, a, S', A']$ for some $i' < i$, i.e.,*

$$d \in \mathit{eval\_tl}_g\,(\eta^{i'}_{\mathsf{lc}}\,[g, a, S', A'])$$

where

(L0) a has been acquired by the ego thread, i.e., $L\,a \neq \emptyset$,

(L1) $L\,a = \{B\}$ such that $B \cap S' = \emptyset$,

(L2) $g \notin V\,a$,

(L3) $\mathit{compat}^{\sharp}_{\mathcal{A}}\;A\;A'$

Proof. The proof is by fixpoint induction. We prove that the values (that are not the initial values of globals) read non-thread-locally for a global g at some $(u, x = g, u')$ during the computation of η^{i}_{lc} are the last *thread-local* writes of a local trace t'' ending in an unlock operation that is side-effected to an appropriate $\eta^{i'}_{\mathsf{lc}}\,[g, a, S', A']$ in some prior iteration $i' < i$ for some a, S', and A' satisfying (L0) through (L3).

This property holds for $i = 0$, as in η^{0}_{lc}, all unknowns have the value $\emptyset$, and, therefore, no reads from globals or unlocks can happen.

For the induction step $i > 0$, there are two proof obligations: First, that the property holds for all reads from a global, and, additionally, that all traces ending in an unlock operation are once more side-effected to appropriate unknowns in this iteration.

For the first obligation, consider a local trace $t \in \eta^{i}_{\mathsf{lc}}\,[u', S, A]$ where the last action is $x = g$ and the result of $\mathsf{last\_write}_g\,t$. Recall that $\mathsf{last\_write}_g\,t$ returns either the edge along which the last write to g happened or the edge corresponding to initMT in case g has not been written. In the latter case, the value read for g is the initial value 0 of g, and the proposition holds. Now consider the case where there is a write to g in t:

$$\mathsf{last\_write}_g\,t = ((j' - 1, u_{j'-1}, \sigma_{j'-1}), g = x', \bar{u}'') = l.$$

Let $i_0 = \mathsf{id}\,t$ and $i_1 = \sigma_{j-1}\,\mathsf{self}$ the thread *id*s of the reading ego thread and the thread performing the last write, respectively. We distinguish two cases:
Case 1: $i_0 = i_1$. The last write is thread-local to t (and l is therefore also the last thread-local write to g in t) and the proposition holds.
Case 2: $i_0 \neq i_1$. The last write is not thread-local. Consider the maximal sub-trace t' of t with $\mathsf{id}\,(t') = i_1$, i.e., the maximal sub-trace of the thread performing the last write to g appearing in t. Let a denote the last (w.r.t. the program order) mutex unlocked by i_1 in t' for which the following additional conditions hold:

- the mutex a is unlocked in t' by i_1 after the last write to g (l)
- the mutex a has also been locked by i_0 in t

- the last lock of the mutex a by i_0 succeeds the unlock of a by i_1 w.r.t. the causality order $\leq$ of t.

We observe that there is at least one mutex, namely m_g, which is unlocked by i_1 after its last write to g and subsequently locked by i_0 before g is read. Let S' denote the background lockset held at the last action $\mathsf{unlock}(a)$ by i_1. Let B denote the background lockset at the last $\mathsf{lock}(a)$ of the ego thread i_0, i.e., $L\,a = \{B\}$.

First, assume that property (L1) is violated for a. Then there is some $c \in B \cap S'$, leading to a contradiction. To see this, assume first that c later is unlocked by i_1 so that i_0 can acquire c. Then, however, the conditions are also fulfilled for c, meaning that a is not the *last* such mutex. If on the other hand, c is never unlocked by i_1 in t', thread i_0 will not be able to acquire c before its last operation $\mathsf{lock}(a)$, also yielding a contradiction.

Accordingly, now assume that $B \cap S' = \emptyset$. We claim that then also $g \notin V\,a$ must hold (L2). If this were not the case, some thread-local write to g by i_0 has happened after the last operation $\mathsf{lock}(a)$. Then, however, the write in t' happens before this write to g by i_0, and is thus not the last write, yielding once again a contradiction.

Lastly, for (L3): The argument is the same as given for (W5) in Proposition 17.

The local trace t'' which is the sub-trace of t' ending in this $\mathsf{unlock}(a)$ of i_1 thus contains the last write to g in t. It was constructed during some earlier iteration $i' < i$ and, by induction hypothesis, added to $\eta^{i'}_{\mathsf{lc}}[g, a, S', A']$ during the i'-th iteration. We conclude that the value d read from g by i_0 is given by $d = \sigma_{j'-1}\,x' \in \mathsf{eval\_tl}_g\,\eta^{i'}_{\mathsf{lc}}\,[g, a, S', A'] \subseteq \mathsf{eval\_tl}_g\,(\eta^{i}_{\mathsf{lc}}\,[g, a, S', A'])$.

It now remains to show that any trace t with $\mathsf{last}(t) = \mathsf{unlock}(a)$, $a \in \mathcal{M}$ ending in (j, u, σ), i.e., $t = (j, u, \sigma)\downarrow_t$, produced in this iteration i is side-effected to $\eta^{i}_{\mathsf{lc}}\,[g, a, S, \alpha_{\mathcal{A}}(t)]$ for $S = L_t[(j, u, \sigma)]$. This, however, follows directly from the construction of $\mathcal{C}_{\mathsf{lc}}$. □

Now it remains to relate solutions of $\mathcal{C}_{\mathsf{lc}}$ and solutions of $\mathcal{C}^{\sharp}_{\mathsf{lc}}$ to each other. Let us refer to $\mathcal{C}^{\sharp}_{\mathsf{lc}}$ by $\mathcal{C}^{\sharp}$ in the remainder of this section.

As a first step, we define a function β that extracts from some local trace t for each mutex a

- the set $V\,a$ of global variables that were written by the ego thread since a was last acquired by it, or all global variables written since the start of the ego thread in case it has never acquired a; and

- the set $L\,a$ containing the background lockset when a was acquired by the ego thread last.

Additionally, β extracts a map σ that contains the values of the locals at the sink of t as

well as the last-written thread-local values of globals. Thus, we define

$$\begin{array}{lcl}
\beta\, t & = & (V, L, \sigma) \qquad \text{where} \\
V & = & \{a \mapsto \{g \mid g \in \mathcal{G}, (_, g = x, \bar{u}') = \mathsf{last_tl_write}_g\, t, \bar{u} \le \bar{u}'\} \mid \\
 & & \quad a \in \mathcal{M}, (_, \mathsf{lock}(a), \bar{u}) = \mathsf{last_tl_lock}_a\, t\} \cup \\
 & & \{a \mapsto \{g \mid g \in \mathcal{G}, (_, g = x, _) = \mathsf{last_tl_write}_g\, t\} \mid \\
 & & \quad a \in \mathcal{M}, \bot = \mathsf{last_tl_lock}_a\, t\} \\
L & = & \{a \mapsto \{L_t[\bar{u}]\} \mid a \in \mathcal{M}, (\bar{u}, \mathsf{lock}(a), _) = \mathsf{last_tl_lock}_a\, t\} \cup \\
 & & \{a \mapsto \emptyset \mid a \in \mathcal{M}, \bot = \mathsf{last_tl_lock}_a\, t\} \\
\sigma & = & \{x \mapsto \{t(x)\} \mid x \in \mathcal{X}\} \\
 & & \cup\, \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\, t\} \\
 & & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t, \bot \neq \mathsf{last_write}_g\, t\} \\
 & & \cup\, \{g \mapsto \{\sigma_{j-1}\, x\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x, _) = \mathsf{last_tl_write}_g\, t\}
\end{array}$$

and remark that the definition of σ is the same as in Eq. (6.1). The abstraction function β then is once more used to specify concretization functions for the values of unknowns $[u, S, A]$ for program points, currently held locksets, and digests as well as for the other unknowns.

$$\begin{array}{lcl}
\gamma_{u,S,A}(V^\sharp, L^\sharp, \sigma^\sharp) & = & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u, L_t = S, \alpha_{\mathcal{A}}\, t = A, \beta\, t = (V, L, \sigma), \\
 & & \qquad \sigma \subseteq \bar{\gamma}_{\bar{\mathcal{V}}^\sharp} \circ \sigma^\sharp, V \sqsubseteq V^\sharp, L \sqsubseteq L^\sharp\}
\end{array}$$

where $\subseteq$ on maps is understood to be defined point-wise, and $\sqsubseteq$ for L and V is defined point-wise as well. We will also once more abbreviate $\bar{\gamma}_{\bar{\mathcal{V}}^\sharp}$ by $\bar{\gamma}$ for the remainder of this section. Furthermore, we have

$$\begin{array}{lcl}
\gamma_{g,a,S,A}(v) & = & \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{unlock}(a), L_t = S, \alpha_{\mathcal{A}}\, t = A, \\
 & & \qquad ((_, _, \sigma_{j-1}), g = x, \bar{u}') = \mathsf{last_tl_write}_g\, t, \sigma_{j-1}\, x \in \gamma_{\mathcal{V}^\sharp}(v)\} \\
 & & \cup\, \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{unlock}(a), L_t = S, \alpha_{\mathcal{A}}\, t = A, \\
 & & \qquad \mathsf{last_tl_write}_g\, t = \bot\} \\
\gamma_{i,A}(v) & = & \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{return}, \alpha_{\mathcal{A}}\, t = A, \mathsf{id}\, t = i, t(\mathsf{ret}) \in \gamma_{\mathcal{V}^\sharp}(v)\} \\
\gamma_{s,A}(V^\sharp, L^\sharp, \sigma^\sharp) & = & \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{signal}(s), \alpha_{\mathcal{A}}\, t = A, \beta\, t = (V, L, \sigma), \\
 & & \qquad \sigma \subseteq \bar{\gamma} \circ \sigma^\sharp, V \sqsubseteq V^\sharp, L \sqsubseteq L^\sharp\}
\end{array}$$

Let $\eta^\sharp$ be a solution of $\mathcal{C}^\sharp$. We then construct from it a mapping η_{lc} by:

$$\begin{array}{lcll}
\eta_{\mathsf{lc}}[u, S, A] & = & \gamma_{u,S,A}(\eta^\sharp\,[u, S, A]) & u \in \mathcal{N}, S \subseteq \mathcal{M}, A \in \mathcal{A} \\
\eta_{\mathsf{lc}}[g, a, S, A] & = & \gamma_{g,a,S,A}(\eta^\sharp\,[g, a, S, A]) & g \in \mathcal{G}, a \in \mathcal{M}, \\
 & & & S \subseteq \mathcal{M}, A \in \mathcal{A} \\
\eta_{\mathsf{lc}}[i, A] & = & \bigcup\{\gamma_{i,A}(\eta^\sharp\,[i^\sharp, A]) \mid i^\sharp \in S_{\mathcal{V}^\sharp_{\mathbf{tid}}}, i \in \gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}\{i^\sharp\}\} & i \in \mathcal{V}_{\mathbf{tid}}, A \in \mathcal{A} \\
\eta_{\mathsf{lc}}[s, A] & = & \gamma_{s,A}(\eta^\sharp\,[s, A]) & s \in \mathcal{S}, A \in \mathcal{A}
\end{array}$$

Altogether, correctness of $\mathcal{C}^\sharp_{\mathsf{lc}}$ follows from the following theorem:

Theorem 18. *Every solution of $\mathcal{C}_{lc}^{\sharp}$ is sound w.r.t. the local trace semantics.*

Proof. Recall from Proposition 21 that the least solution of $\mathcal{C}_{\mathsf{lc}}$ is sound w.r.t. the local trace semantic (where the proof proceeds via $\mathcal{C}_{\mathsf{wc}}$ and Proposition 16). It thus suffices to show that the mapping η_{lc} as constructed above, is a solution of the constraint system $\mathcal{C}_{\mathsf{lc}}$. For that, we verify by fixpoint induction that for the i-th approximation η^i to the least solution η of $\mathcal{C}_{\mathsf{lc}}$, $\eta^i \subseteq \eta_{\mathsf{lc}}$ holds.

To this end, we first consider the constraints for **initialization**, the start point u_0 and the empty lockset. We verify that for all $A \in \mathsf{init}_{\mathcal{A}}^{\sharp}$:

$$(\emptyset, \{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\}) \quad \subseteq \quad (\eta_{\mathsf{lc}}, \eta_{\mathsf{lc}}\,[u_0, \emptyset, A])$$

As no side-effects are triggered, it suffices to check that $\{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\} \subseteq \eta_{\mathsf{lc}}\,[u_0, \emptyset, A]$ holds.

$$\begin{array}{rcl}
\mathsf{init}(A)^{\sharp}\,_ & = & \textbf{let } V^{\sharp} = \{a \mapsto \emptyset \mid a \in \mathcal{M}\} \textbf{ in} \\
& & \textbf{let } L^{\sharp} = \{a \mapsto \emptyset \mid a \in \mathcal{M}\} \textbf{ in} \\
& & \textbf{let } \sigma^{\sharp} = \{x \mapsto \top \mid x \in \mathcal{X} \setminus \{\mathsf{self}\}\} \cup \{\mathsf{self} \mapsto [\![i_0]\!]^{\sharp}_{\mathcal{E}xp}\,\top\} \\
& & \qquad\qquad \cup \{g \mapsto [\![0]\!]^{\sharp}_{\mathcal{E}xp}\,\top \mid g \in \mathcal{G}\} \\
& & \textbf{in} \\
& & (\emptyset, (V^{\sharp}, L^{\sharp}, \sigma^{\sharp}))
\end{array}$$

Let $\eta^{\sharp}\,[u_0, \emptyset, A] = (V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'})$ the value provided by $\eta^{\sharp}$ for the start point and the empty lockset. Since $\eta^{\sharp}$ is a solution of $\mathcal{C}^{\sharp}$, $V^{\sharp} \sqsubseteq V^{\sharp'}$, $L^{\sharp} \sqsubseteq L^{\sharp'}$, and $\sigma^{\sharp} \sqsubseteq \sigma^{\sharp'}$ all hold. Then, by definition:

$$\begin{array}{rcl}
\eta_{\mathsf{lc}}[u_0, \emptyset, A] & = & \gamma_{u_0,\emptyset,A}(V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'}) \\
& = & \{t \in \mathcal{T} \mid \mathsf{loc}\,t = u_0, L_t = \emptyset, \alpha_{\mathcal{A}}\,t = A, \beta\,t = (V, L, \sigma), \\
& & \qquad \sigma \subseteq \bar{\gamma}_{\bar{\mathcal{V}}^{\sharp}} \circ \sigma^{\sharp'}, V \sqsubseteq V^{\sharp'}, L \sqsubseteq L^{\sharp'}\}
\end{array}$$

For every trace $t \in \{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\}$, let

$$\begin{array}{rcl}
\beta\,t & = & (V, L, \sigma) \qquad \text{where} \\
V & = & \{a \mapsto \{g \mid g \in \mathcal{G}, (_, g = x, \bar{u}') = \mathsf{last_tl_write}_g\,t, \bar{u} \leq \bar{u}'\} \mid \\
& & \quad a \in \mathcal{M}, (_, \mathsf{lock}(a), \bar{u}) = \mathsf{last_tl_lock}_a\,t\} \cup \\
& & \{a \mapsto \{g \mid g \in \mathcal{G}, (_, g = x, _) = \mathsf{last_tl_write}_g\,t\} \mid \\
& & \quad a \in \mathcal{M}, \bot = \mathsf{last_tl_lock}_a\,t\} \\
& = & \{a \mapsto \emptyset \mid a \in \mathcal{M}\} \\
L & = & \{a \mapsto \{L_t[\bar{u}]\} \mid a \in \mathcal{M}, (\bar{u}, \mathsf{lock}(a), _) = \mathsf{last_tl_lock}_a\,t\} \cup \\
& & \{a \mapsto \emptyset \mid a \in \mathcal{M}, \bot = \mathsf{last_tl_lock}_a\,t\} \\
& = & \{a \mapsto \emptyset \mid a \in \mathcal{M}\}
\end{array}$$

$$\begin{array}{rcl}
\sigma & = & \{x \mapsto \{t(x)\} \mid x \in \mathcal{X}\} \\
& & \cup\, \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\, t\} \\
& & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t, \bot \neq \mathsf{last_write}_g\, t\} \\
& & \cup\, \{g \mapsto \{\sigma_{j-1}\, x\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x, _) = \mathsf{last_tl_write}_g\, t\} \\
& = & \{x \mapsto \{t(x)\} \mid x \in \mathcal{X}\} \cup \{g \mapsto \{0\} \mid g \in \mathcal{G}\}
\end{array}$$

Thus, $V = V^\sharp \sqsubseteq V^{\sharp'}$, $L = L^\sharp \sqsubseteq L^{\sharp'}$, and

$$\begin{array}{rcl}
\sigma & = & \{x \mapsto \{t(x)\} \mid x \in \mathcal{X}\} \cup \{g \mapsto \{0\} \mid g \in \mathcal{G}\} \\
& \subseteq & \bar{\gamma} \circ (\{x \mapsto \top \mid x \in \mathcal{X} \setminus \{\mathsf{self}\}\} \cup \{\mathsf{self} \mapsto [\![i_0]\!]^\sharp_{\mathcal{E}xp}\, \top\} \cup \{g \mapsto [\![0]\!]^\sharp_{\mathcal{E}xp}\, \top \mid g \in \mathcal{G}\}) \\
& = & \bar{\gamma} \circ \sigma^\sharp \\
& \subseteq & \bar{\gamma} \circ \sigma^{\sharp'}
\end{array}$$

Altogether $t \in \eta_{\mathsf{lc}}\, [u_0, \emptyset, A]$ holds for all $t \in \{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\}$.

Next, consider the constraints for **initMT**. Consider an edge $(u, \mathsf{initMT}, u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, \mathsf{initMT}]\!]^\sharp_{\mathcal{A}}(A_0)$. We remark that, by construction, the lockset is empty when executing initMT. We verify that

$$[\![([u, \emptyset, A_0], \mathsf{initMT}, u')]\!]_{\mathsf{lc}}\, \eta^{i-1} \quad \subseteq \quad (\eta_{\mathsf{lc}}, \eta_{\mathsf{lc}}\, [u', \emptyset, A'])$$

We have

$$\begin{array}{rcl}
[\![([u, \emptyset, A_0], \mathsf{initMT}, u')]\!]_{\mathsf{lc}}\, \eta & = & (\emptyset, [\![(u, \mathsf{initMT}, u')]\!]_{\mathcal{T}}(\eta\, [u, \emptyset, A_0])) \\
[\![[u, \emptyset, A_0], \mathsf{initMT}]\!]^\sharp \eta^\sharp & = & \big(\emptyset, \eta^\sharp\, [u, \emptyset, A_0]\big)
\end{array}$$

Let $\eta^\sharp\, [u, \emptyset, A_0] = (V^\sharp, L^\sharp, \sigma^\sharp)$ and $\eta^\sharp\, [u', \emptyset, A'] = (V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'})$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge, the empty lockset, and the resulting digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $V^\sharp \sqsubseteq V^{\sharp'}$, $L^\sharp \sqsubseteq L^{\sharp'}$, and $\sigma^\sharp \sqsubseteq \sigma^{\sharp'}$ all hold. Then, by definition:

$$\begin{array}{rcl}
\eta_{\mathsf{lc}}[u', \emptyset, A'] & = & \gamma_{u', \emptyset, A'}(V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'}) \\
& = & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = \emptyset, \alpha_{\mathcal{A}}\, t = A', \beta\, t = (V, L, \sigma), \\
& & \qquad \sigma \subseteq \bar{\gamma}_{\bar{\mathcal{V}}^\sharp} \circ \sigma^{\sharp'}, V \sqsubseteq V^{\sharp'}, L \sqsubseteq L^{\sharp'}\}
\end{array}$$

For every trace $t \in \eta^{i-1}\, [u, \emptyset, A_0]$, let $\beta\, t = (V, L, \sigma)$. By induction hypothesis, $V \sqsubseteq V^\sharp$, $L \sqsubseteq L^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let $t' = [\![(u, \mathsf{initMT}, u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = \emptyset$, $\alpha_{\mathcal{A}}\, t' = A'$, and

$$\begin{array}{rcl}
\beta\, t' & = & (V', L', \sigma') \qquad \text{where} \\
V' & = & \{a \mapsto \{g \mid g \in \mathcal{G}, (_, g = x, \bar{u}') = \mathsf{last_tl_write}_g\, t', \bar{u} \leq \bar{u}'\} \mid \\
& & \qquad a \in \mathcal{M}, (_, \mathsf{lock}(a), \bar{u}) = \mathsf{last_tl_lock}_a\, t'\} \cup \\
& & \{a \mapsto \{g \mid g \in \mathcal{G}, (_, g = x, _) = \mathsf{last_tl_write}_g\, t'\} \mid \\
& & \qquad a \in \mathcal{M}, \bot = \mathsf{last_tl_lock}_a\, t'\} \\
& = & V \\
L' & = & \{a \mapsto \{L_{t'}[\bar{u}]\} \mid a \in \mathcal{M}, (\bar{u}, \mathsf{lock}(a), _) = \mathsf{last_tl_lock}_a\, t'\} \cup \\
& & \{a \mapsto \emptyset \mid a \in \mathcal{M}, \bot = \mathsf{last_tl_lock}_a\, t'\} \\
& = & L
\end{array}$$

$$\begin{array}{rcl}\sigma' & = & \{x \mapsto \{t'(x)\} \mid x \in \mathcal{X}\} \\ & & \cup\, \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last\_write}_g\, t'\} \\ & & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\, t', \bot \neq \mathsf{last\_write}_g\, t'\} \\ & & \cup\, \{g \mapsto \{\sigma_{j-1}\, x\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x, \_) = \mathsf{last\_tl\_write}_g\, t'\} \\ & = & \sigma \end{array}$$

Thus, $V' = V \sqsubseteq V^\sharp \sqsubseteq V^{\sharp'}$, $L' = L \sqsubseteq L^\sharp \sqsubseteq L^{\sharp'}$ and $\sigma' = \sigma \subseteq \bar{\gamma} \circ \sigma^\sharp \subseteq \bar{\gamma} \circ \sigma^{\sharp'}$. Altogether, $t' \in \eta_{\mathsf{lc}}\,[u', \emptyset, A']$ holds for all $t \in \eta^{i-1}\,[u, \emptyset, A_0]$. We conclude that the return value of $[\![([u, \emptyset, A_0], \mathsf{initMT}, u')]\!]_{\mathsf{lc}}\, \eta^{i-1}$ is subsumed by the value $\eta_{\mathsf{lc}}\,[u', \emptyset, A']$ and since the constraint causes no side-effects, the claim holds.

Next, consider the constraints for a **read** from a global $x = g$. Consider an edge $(u, x = g, u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, x = g]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![([u, S, A_0], x = g, u')]\!]_{\mathsf{lc}}\, \eta^{i-1} \quad \subseteq \quad (\eta_{\mathsf{lc}}, \eta_{\mathsf{lc}}\,[u', S, A'])$$

We have

$$\begin{array}{rcl} [\![([u, S, A_0], x = g, u')]\!]_{\mathsf{lc}}\, \eta & = & (\emptyset, [\![(u, x = g, u')]\!]_{\mathcal{T}}(\eta\,[u, S, A_0])) \\ [\![[u, S, A_0], x = g]\!]^\sharp \eta^\sharp & = & \textbf{let } (V^\sharp, L^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\ & & \textbf{let } d = \sigma^\sharp\, g \sqcup \bigsqcup \{\eta^\sharp[g, a, S', A'] \mid a \in \mathcal{M}, g \notin V^\sharp\, a, \\ & & \qquad \exists B \in L^\sharp\, a, B \cap S' = \emptyset, A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \\ & & \textbf{in} \\ & & \textbf{let } \sigma^{\sharp''} = \sigma^\sharp \oplus \{x \mapsto d\} \textbf{ in} \\ & & (\emptyset, (V^\sharp, L^\sharp, \sigma^{\sharp''})) \end{array}$$

Let $\eta^\sharp\,[u, S, A_0] = (V^\sharp, L^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u', S, A'] = (V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'})$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge, the appropriate lockset, and the resulting digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $V^\sharp \sqsubseteq V^{\sharp'}$, $L^\sharp \sqsubseteq L^{\sharp'}$, and $\sigma^{\sharp''} \sqsubseteq \sigma^{\sharp'}$ all hold. Then, by definition:

$$\begin{array}{rcl} \eta_{\mathsf{lc}}[u', S, A'] & = & \gamma_{u', S, A'}(V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'}) \\ & = & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = S, \alpha_{\mathcal{A}}\, t = A', \beta\, t = (V, L, \sigma), \\ & & \qquad \sigma \subseteq \bar{\gamma}_{\mathcal{V}^\sharp} \circ \sigma^{\sharp'}, V \sqsubseteq V^{\sharp'}, L \sqsubseteq L^{\sharp'}\} \end{array}$$

For every trace $t \in \eta^{i-1}\,[u, S, A_0]$, let $\beta\, t = (V, L, \sigma)$. By induction hypothesis, $V \sqsubseteq V^\sharp$, $L \sqsubseteq L^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let $t' = [\![(u, x = g, u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S$, $\alpha_{\mathcal{A}}\, t' = A'$, and

$$\begin{array}{rcll} \beta\, t' & = & (V', L', \sigma') & \text{where} \\ V' & = & \{a \mapsto \{g' \mid g' \in \mathcal{G}, (\_, g' = x', \bar{u}') = \mathsf{last\_tl\_write}_{g'}\, t', \bar{u} \leq \bar{u}'\} \mid \\ & & \quad a \in \mathcal{M}, (\_, \mathsf{lock}(a), \bar{u}) = \mathsf{last\_tl\_lock}_a\, t'\} \cup \\ & & \{a \mapsto \{g' \mid g' \in \mathcal{G}, (\_, g' = x', \_) = \mathsf{last\_tl\_write}_{g'}\, t'\} \mid \\ & & \quad a \in \mathcal{M}, \bot = \mathsf{last\_tl\_lock}_a\, t'\} = V \\ L' & = & \{a \mapsto \{L_{t'}[\bar{u}]\} \mid a \in \mathcal{M}, (\bar{u}, \mathsf{lock}(a), \_) = \mathsf{last\_tl\_lock}_a\, t'\} \cup \\ & & \{a \mapsto \emptyset \mid a \in \mathcal{M}, \bot = \mathsf{last\_tl\_lock}_a\, t'\} = L \end{array}$$

$$
\begin{aligned}
\sigma' &= \{x' \mapsto \{t'(x')\} \mid x' \in \mathcal{X}\} \\
&\quad \cup \{g' \mapsto \{0\} \mid g' \in \mathcal{G}, \bot = \mathsf{last\_write}_{g'}\, t'\} \\
&\quad \cup \{g' \mapsto \emptyset \mid g' \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_{g'}\, t', \bot \neq \mathsf{last\_write}_{g'}\, t'\} \\
&\quad \cup \{g' \mapsto \{\sigma_{j-1}\, x'\} \mid g' \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g' = x', \_) = \mathsf{last\_tl\_write}_{g'}\, t'\} \\
&= \sigma \oplus \{x \mapsto \{t'(x)\}\} \\
&= \sigma \oplus \begin{cases} \{x \mapsto \{\sigma_{j'-1}\, x'\}\} & \text{if } \mathsf{last\_write}_g\, t' = ((j'-1, u_{j'-1}, \sigma_{j'-1}), g = x', \_) \\ \{x \mapsto \{0\}\} & \text{if } \mathsf{last\_write}_g\, t' = \bot \end{cases} \\
&= \sigma \oplus \begin{cases} \{x \mapsto \{\sigma_{j'-1}\, x'\}\} & \text{if } \mathsf{last\_write}_g\, t = ((j'-1, u_{j'-1}, \sigma_{j'-1}), g = x', \_) \\ \{x \mapsto \{0\}\} & \text{if } \mathsf{last\_write}_g\, t = \bot \end{cases}
\end{aligned}
$$

Thus, $L' = L \sqsubseteq L^\sharp \sqsubseteq L^{\sharp'}$ and $V' = V \sqsubseteq V^\sharp \sqsubseteq V^{\sharp'}$. Also, $\sigma\, y = \sigma'\, y$ and therefore, $\sigma'\, y \subseteq (\bar{\gamma} \circ \sigma^{\sharp'})\, y$ for $y \not\equiv x$. For $y \equiv x$, we consider three cases:

- There is no write to g ($\mathsf{last\_write}_g\, t = \bot$): Then $\sigma\, g = \{0\} \subseteq (\bar{\gamma}\, \sigma^\sharp)\, g$, thus $\sigma'\, x \subseteq (\bar{\gamma} \circ \sigma^{\sharp''})\, x$ and accordingly, $\sigma' \subseteq \bar{\gamma} \circ \sigma^{\sharp'}$.
- The last write to g is thread-local ($\mathsf{last\_write}_g\, t = \mathsf{last\_tl\_write}_g\, t$): Then $\sigma\, g = \{\sigma_{j'-1}\, x'\} \subseteq (\bar{\gamma} \circ \sigma^\sharp)\, g$, thus $\sigma'\, x \subseteq (\bar{\gamma} \circ \sigma^{\sharp''})\, x$ and accordingly, $\sigma' \subseteq \bar{\gamma} \circ \sigma^{\sharp'}$.
- The last write to g is non-thread-local.

$$
\begin{aligned}
\sigma'\, x &\subseteq \bigcup \{\mathsf{eval\_tl}_g\, (\eta_{\mathsf{lc}}^{i-1}\, [g, a, S', A']) \mid a \in \mathcal{M}, g \notin V^\sharp\, a, \\
&\qquad \exists B \in L^\sharp\, a, B \cap S' = \emptyset, \\
&\qquad \mathsf{compat}_{\mathcal{A}}^\sharp\, A_0\, A'\} \qquad \text{(By Proposition 22)} \\
&\subseteq \bigcup \{\mathsf{eval\_tl}_g\, (\eta_{\mathsf{lc}}\, [g, a, S', A']) \mid a \in \mathcal{M}, g \notin V^\sharp\, a, \\
&\qquad \exists B \in L^\sharp\, a, B \cap S' = \emptyset, \\
&\qquad \mathsf{compat}_{\mathcal{A}}^\sharp\, A_0\, A'\} \qquad \text{(By Induction Hypothesis)} \\
&\subseteq \bigcup \{\mathsf{eval\_tl}_g\, (\gamma_{g,a,S',A'}(\eta^\sharp\, [g, a, S', A'])) \mid a \in \mathcal{M}, g \notin V^\sharp\, a, \\
&\qquad \exists B \in L^\sharp\, a, B \cap S' = \emptyset, \\
&\qquad \mathsf{compat}_{\mathcal{A}}^\sharp\, A_0\, A'\} \\
&\subseteq \bigcup \{\gamma_{\mathcal{V}^\sharp}(\eta^\sharp\, [g, a, S', A']) \mid a \in \mathcal{M}, g \notin V^\sharp\, a, \\
&\qquad \exists B \in L^\sharp\, a, B \cap S' = \emptyset, \\
&\qquad \mathsf{compat}_{\mathcal{A}}^\sharp\, A_0\, A'\} \\
&\subseteq \gamma_{\mathcal{V}^\sharp}(\bigsqcup \{(\eta^\sharp\, [g, a, S', A']) \mid a \in \mathcal{M}, g \notin V^\sharp\, a, \\
&\qquad \exists B \in L^\sharp\, a, B \cap S' = \emptyset, \\
&\qquad \mathsf{compat}_{\mathcal{A}}^\sharp\, A_0\, A'\}) \\
&\subseteq \gamma_{\mathcal{V}^\sharp}(\sigma^\sharp\, g \sqcup \bigsqcup \{(\eta^\sharp\, [g, a, S', A']) \mid a \in \mathcal{M}, g \notin V^\sharp\, a, \\
&\qquad \exists B \in L^\sharp\, a, B \cap S' = \emptyset, \\
&\qquad \mathsf{compat}_{\mathcal{A}}^\sharp\, A_0\, A'\}) \\
&= \gamma_{\mathcal{V}^\sharp}(\sigma^{\sharp''}\, g) = (\bar{\gamma} \circ \sigma^{\sharp''})\, x \\
&\subseteq (\bar{\gamma} \circ \sigma^{\sharp'})\, x
\end{aligned}
$$

and thus $\sigma' \subseteq \bar{\gamma} \circ \sigma^{\sharp''} \subseteq \bar{\gamma} \circ \sigma^{\sharp'}$.

Altogether, $t' \in \eta_{\mathsf{lc}}\,[u', S, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], x = g, u')]\!]_{\mathsf{lc}}\,\eta^{i-1}$ is subsumed by the value $\eta_{\mathsf{lc}}\,[u', S, A']$ and since the constraint causes no side-effects, the claim holds.

Next, consider the constraints for a **write** to a global $g = x$. Consider an edge $(u, g = x, u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, g = x]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![([u, S, A_0], g = x, u')]\!]_{\mathsf{lc}}\,\eta^{i-1} \quad \subseteq \quad (\eta_{\mathsf{lc}}, \eta_{\mathsf{lc}}\,[u', S, A'])$$

We have

$$\begin{array}{lcl}
[\![([u, S, A_0], g = x, u')]\!]_{\mathsf{lc}}\,\eta & = & (\emptyset, [\![(u, g = x, u')]\!]_{\mathcal{T}}(\eta\,[u, S, A_0])) \\
[\![[u, S, A_0], g = x]\!]^\sharp\eta^\sharp & = & \textbf{let } (V^\sharp, L^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\
 & & \textbf{let } V^{\sharp\prime\prime} = \{a \mapsto (V^\sharp\,a \cup \{g\}) \mid a \in \mathcal{M}\} \textbf{ in} \\
 & & \textbf{let } \sigma^{\sharp\prime\prime} = \sigma^\sharp \oplus \{g \mapsto (\sigma^\sharp\,x)\} \textbf{ in} \\
 & & (\emptyset, (V^{\sharp\prime\prime}, L^\sharp, \sigma^{\sharp\prime\prime}))
\end{array}$$

Let $\eta^\sharp\,[u, S, A_0] = (V^\sharp, L^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u', S, A'] = (V^{\sharp\prime}, L^{\sharp\prime}, \sigma^{\sharp\prime})$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge, the appropriate lockset, and the resulting digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $V^{\sharp\prime\prime} \sqsubseteq V^{\sharp\prime}$, $L^\sharp \sqsubseteq L^{\sharp\prime}$, and $\sigma^{\sharp\prime\prime} \sqsubseteq \sigma^{\sharp\prime}$ all hold. Then, by definition:

$$\begin{array}{lcl}
\eta_{\mathsf{lc}}[u', S, A'] & = & \gamma_{u', S, A'}(V^{\sharp\prime}, L^{\sharp\prime}, \sigma^{\sharp\prime}) \\
 & = & \{t \in \mathcal{T} \mid \mathsf{loc}\,t = u', L_t = S, \alpha_{\mathcal{A}}\,t = A', \beta\,t = (V, L, \sigma), \\
 & & \qquad \sigma \subseteq \bar{\gamma}_{\bar{\mathcal{V}}^\sharp} \circ \sigma^{\sharp\prime}, V \sqsubseteq V^{\sharp\prime}, L \sqsubseteq L^{\sharp\prime}\}
\end{array}$$

For every trace $t \in \eta^{i-1}\,[u, S, A_0]$, let $\beta\,t = (V, L, \sigma)$. By induction hypothesis, $V \sqsubseteq V^\sharp$, $L \sqsubseteq L^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let $t' = [\![(u, g = x, u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\,t' = u'$, $L_{t'} = S$, $\alpha_{\mathcal{A}}\,t' = A'$, and

$$\begin{array}{lcl}
\beta\,t' & = & (V', L', \sigma') \qquad \text{where} \\
V' & = & \{a \mapsto \{g' \mid g' \in \mathcal{G}, (_, g' = x', \bar{u}') = \mathsf{last_tl_write}_{g'}\,t', \bar{u} \leq \bar{u}'\} \mid \\
 & & \qquad a \in \mathcal{M}, (_, \mathsf{lock}(a), \bar{u}) = \mathsf{last_tl_lock}_a\,t'\} \cup \\
 & & \{a \mapsto \{g' \mid g' \in \mathcal{G}, (_, g' = x', _) = \mathsf{last_tl_write}_{g'}\,t'\} \mid \\
 & & \qquad a \in \mathcal{M}, \bot = \mathsf{last_tl_lock}_a\,t'\} \\
 & = & \{a \mapsto Va \cup \{g\} \mid a \in \mathcal{M}\} \\
L' & = & \{a \mapsto \{L_{t'}[\bar{u}]\} \mid a \in \mathcal{M}, (\bar{u}, \mathsf{lock}(a), _) = \mathsf{last_tl_lock}_a\,t'\} \cup \\
 & & \{a \mapsto \emptyset \mid a \in \mathcal{M}, \bot = \mathsf{last_tl_lock}_a\,t'\} \\
 & = & L \\
\sigma' & = & \{x' \mapsto \{t'(x')\} \mid x' \in \mathcal{X}\} \\
 & & \cup \{g' \mapsto \{0\} \mid g' \in \mathcal{G}, \bot = \mathsf{last_write}_{g'}\,t'\} \\
 & & \cup \{g' \mapsto \emptyset \mid g' \in \mathcal{G}, \bot = \mathsf{last_tl_write}_{g'}\,t', \bot \neq \mathsf{last_write}_{g'}\,t'\} \\
 & & \cup \{g' \mapsto \{\sigma_{j-1}\,x'\} \mid g' \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g' = x', _) = \mathsf{last_tl_write}_{g'}\,t'\} \\
 & = & \sigma \oplus \{g \mapsto \sigma x\}
\end{array}$$

Thus, $L' = L \sqsubseteq L^\sharp \sqsubseteq L^{\sharp'}$, and

$$\begin{array}{rcl} V' & = & \{V\,a \cup \{g\} \mid a \in \mathcal{M}\} \sqsubseteq \{a \mapsto V^\sharp\,a \cup \{g\} \mid a \in \mathcal{M}\} = V^{\sharp''} \sqsubseteq V^{\sharp'} \\ \sigma' & = & \sigma \oplus \{g \mapsto \sigma\,x\} \subseteq \bar{\gamma} \circ (\sigma^\sharp \oplus \{g \mapsto \sigma^\sharp\,x\}) = \bar{\gamma} \circ \sigma^{\sharp''} \subseteq \bar{\gamma} \circ \sigma^{\sharp'} \end{array}$$

Altogether, $t' \in \eta_{\mathsf{lc}}\,[u', S, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], g = x, u')]\!]_{\mathsf{lc}}\,\eta^{i-1}$ is subsumed by the value $\eta_{\mathsf{lc}}\,[u', S, A']$ and since the constraint causes no side-effects, the claim holds.

For constraints corresponding to **computations on locals** and corresponding to **guards**, the right-hand sides for the analysis change the σ components of the abstract states only. Furthermore, for such actions act, $t' \in [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}\{t\}$, $\beta\,t = (L, V, \sigma)$, and $\beta\,t' = (L', V', \sigma')$, $L = L'$ and $V = V'$ both hold. It thus suffices to consider the third component here. As the effect on the σ components of the abstract states in this analysis is the same as for Write-Centered-Reading, and the definition of the σ-component of β in Section 6.1.1 is identical to the definition here, the arguments given for these actions in Section 6.1.1 apply here as well. We conclude that for such actions act and $A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0)$, the return value of $[\![([u, S, A_0], \mathsf{act}, u')]\!]_{\mathsf{lc}}\,\eta^{i-1}$ is subsumed by the value $\eta_{\mathsf{lc}}\,[u', S, A']$ and since the constraints causes no side-effects, the claim holds.

Next, consider the constraints corresponding to **locking** a mutex a. Consider an edge $(u, \mathsf{lock}(a), u') \in \mathcal{E}$ and digests A', A_0 and all appropriate A_1 such that $A' \in [\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}(A_0, A_1)$. We verify that

$$[\![([u, S, A_0], \mathsf{lock}(a), u')]\!]_{\mathsf{lc}}\,\eta^{i-1} \quad \subseteq \quad (\eta_{\mathsf{lc}}, \eta_{\mathsf{lc}}\,[u', S \cup \{a\}, A'])$$

We have

$[\![([u, S, A_0], \mathsf{lock}(a), u')]\!]_{\mathsf{lc}}\,\eta =$
$\quad$ **let** $T_1 = \bigcup\{\eta\,[g, a, S', A_1] \mid g \in \mathcal{G}, S' \subseteq \mathcal{M}, A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1\}$ **in**
$\quad$ **let** $T_2 = \{t' \mid t \in \eta\,[u, S, A_0], \bar{i} = \mathsf{init\_v}\,t, \bar{i} \neq \bot, t' = \downarrow_{\bar{i}}(t), \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,(\alpha_{\mathcal{A}}\,t')\}$ **in**
$\quad$ **let** $T = [\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}(\eta\,[u, S, A_0], T_1 \cup T_2)$ **in**
$\quad$ $(\emptyset, T)$

$[\![[u, S, A_0], \mathsf{lock}(a)]\!]^\sharp \eta^\sharp =$
$\quad$ **let** $(V^\sharp, L^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0]$ **in**
$\quad$ **let** $V^{\sharp''} = V^\sharp \oplus \{a \mapsto \emptyset\}$ **in**
$\quad$ **let** $L^{\sharp''} = L^\sharp \oplus \{a \mapsto \{S\}\}$ **in**
$\quad$ $(\emptyset, (V^{\sharp''}, L^{\sharp''}, \sigma^\sharp))$

Let $\eta^\sharp\,[u, S, A_0] = (V^\sharp, L^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u', S \cup \{a\}, A'] = (V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'})$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge, the appropriate lockset, and the resulting digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $V^{\sharp''} \sqsubseteq V^{\sharp'}$, $L^{\sharp''} \sqsubseteq L^{\sharp'}$, and $\sigma^\sharp \sqsubseteq \sigma^{\sharp'}$ all hold. Then, by definition:

$$\begin{array}{rcl} \eta_{\mathsf{lc}}[u', S \cup \{a\}, A'] & = & \gamma_{u', S \cup \{a\}, A'}(V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'}) \\ & = & \{t \in \mathcal{T} \mid \mathsf{loc}\,t = u', L_t = S \cup \{a\}, \alpha_{\mathcal{A}}\,t = A', \beta\,t = (V, L, \sigma), \\ & & \qquad \sigma \subseteq \bar{\gamma}_{\bar{\mathcal{V}}^\sharp} \circ \sigma^{\sharp'}, V \sqsubseteq V^{\sharp'}, L \sqsubseteq L^{\sharp'}\} \end{array}$$

For every trace $t \in \eta^{i-1}\,[u, S, A_0]$, let $\beta\, t = (V, L, \sigma)$. By induction hypothesis, $V \sqsubseteq V^\sharp$, $L \sqsubseteq L^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let

$$\begin{array}{rcl} t' & \in & [\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}(\{t\}, \\ & & \quad \bigcup\{\eta\,[g, a, S', A_1] \mid g \in \mathcal{G}, S' \subseteq \mathcal{M}, A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\} \\ & & \quad \cup\{t'' \mid t' \in \eta\,[u, S, A_0], \bar{i} = \mathsf{init\_v}\, t', \bar{i} \neq \bot, t'' = \downarrow_{\bar{i}}(t'), \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, (\alpha_{\mathcal{A}}\, t'')\}) \\ & = & [\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}(\{t\}, \\ & & \quad \bigcup\{\eta\,[g, a, S', A_1] \mid g \in \mathcal{G}, S' \subseteq \mathcal{M}, A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\} \\ & & \quad \cup\{t' \mid \bar{i} = \mathsf{init\_v}\, t, \bar{i} \neq \bot, t' = \downarrow_{\bar{i}}(t)\}) \end{array}$$

where the equality exploits that for a given local trace t and a first lock of a, the second trace which contains the observed initMT must be a sub-trace of t. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S \cup \{a\}$, $\alpha_{\mathcal{A}}\, t' = A'$, and

$$\begin{array}{rcl} \beta\, t' & = & (V', L', \sigma') \qquad \text{where} \\ V' & = & \{a \mapsto \{g \mid g \in \mathcal{G}, (\_, g = x, \bar{u}') = \mathsf{last\_tl\_write}_g\, t', \bar{u} \leq \bar{u}'\} \mid \\ & & \qquad a \in \mathcal{M}, (\_, \mathsf{lock}(a), \bar{u}) = \mathsf{last\_tl\_lock}_a\, t'\} \cup \\ & & \{a \mapsto \{g \mid g \in \mathcal{G}, (\_, g = x, \_) = \mathsf{last\_tl\_write}_g\, t'\} \mid \\ & & \qquad a \in \mathcal{M}, \bot = \mathsf{last\_tl\_lock}_a\, t'\} \\ & = & V \oplus \{a \mapsto \emptyset\} \\ L' & = & \{a \mapsto \{L_{t'}[\bar{u}]\} \mid a \in \mathcal{M}, (\bar{u}, \mathsf{lock}(a), \_) = \mathsf{last\_tl\_lock}_a\, t'\} \cup \\ & & \{a \mapsto \emptyset \mid a \in \mathcal{M}, \bot = \mathsf{last\_tl\_lock}_a\, t'\} \\ & = & L \oplus \{a \mapsto \{S\}\} \\ \sigma' & = & \{x \mapsto \{t'(x)\} \mid x \in \mathcal{X}\} \\ & & \cup\, \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last\_write}_g\, t'\} \\ & & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\, t', \bot \neq \mathsf{last\_write}_g\, t'\} \\ & & \cup\, \{g \mapsto \{\sigma_{j-1}\, x\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x, \_) = \mathsf{last\_tl\_write}_g\, t'\} \\ & = & \sigma \oplus \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\, t', \bot \neq \mathsf{last\_write}_g\, t'\} \\ & \subseteq & \sigma \end{array}$$

Therefore,

$$\begin{array}{rcl} V' & = & V \oplus \{a \mapsto \emptyset\} \sqsubseteq V^\sharp \oplus \{a \mapsto \emptyset\} = V^{\sharp''} \sqsubseteq V^{\sharp'} \\ L' & = & L \oplus \{a \mapsto \{S\}\} \sqsubseteq L^\sharp \oplus \{a \mapsto \{S\}\} = L^{\sharp''} \sqsubseteq L^{\sharp'} \\ \sigma' & \subseteq & \sigma \subseteq \bar{\gamma} \circ \sigma^\sharp \subseteq \bar{\gamma} \circ \sigma^{\sharp'} \end{array}$$

Altogether, $t' \in \eta_{\mathsf{lc}}\,[u', S \cup \{a\}, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], \mathsf{lock}(a), u')]\!]_{\mathsf{lc}}\, \eta^{i-1}$ is subsumed by the value $\eta_{\mathsf{lc}}\,[u', S \cup \{a\}, A']$ and since the constraint causes no side-effects, the claim holds.

Next, consider the constraints corresponding to **unlocking** some mutex a. Consider an edge $(u, \mathsf{unlock}(a), u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, \mathsf{unlock}(a)]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![([u, S, A_0], \mathsf{unlock}(a), u'), A']\!]_{\mathsf{lc}}\, \eta^{i-1} \quad \subseteq \quad (\eta_{\mathsf{lc}}, \eta_{\mathsf{lc}}\,[u', S \setminus \{a\}, A'])$$

We have

$$
\begin{array}{l}
[\![([u, S, A_0], \mathsf{unlock}(a), u'), A']\!]_{\mathsf{lc}}\,\eta = \\
\quad \textbf{let } T = [\![(u, \mathsf{unlock}(a), u')]\!]_{\mathcal{T}}(\eta\,[u, S, A_0]) \textbf{ in} \\
\quad \textbf{let } \rho = \{[g, a, S \setminus \{a\}, A'] \mapsto T \mid g \in \mathcal{G}\} \textbf{ in} \\
\quad (\rho, T)
\end{array}
$$

$$
\begin{array}{l}
[\![[u, S, A_0], \mathsf{unlock}(a), A']\!]^\sharp \eta^\sharp \\
\quad \textbf{let } (V^\sharp, L^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\
\quad \textbf{let } \rho^\sharp = \{[g, a, S \setminus \{a\}, A'] \mapsto \sigma^\sharp\, g \mid g \in \mathcal{G}\} \textbf{ in} \\
\quad (\rho^\sharp, (V^\sharp, L^\sharp, \sigma^\sharp))
\end{array}
$$

Let $\eta^\sharp\,[u, S, A_0] = (V^\sharp, L^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u', S \setminus \{a\}, A'] = (V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'})$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge, the appropriate lockset, and the resulting digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $V^\sharp \sqsubseteq V^{\sharp'}$, $L^\sharp \sqsubseteq L^{\sharp'}$, and $\sigma^\sharp \sqsubseteq \sigma^{\sharp'}$ all hold. Then, by definition:

$$
\begin{array}{rcl}
\eta_{\mathsf{lc}}[u', S \setminus \{a\}, A'] & = & \gamma_{u', S\setminus\{a\}, A'}(V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'}) \\
& = & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = S \setminus \{a\}, \alpha_{\mathcal{A}}\, t = A', \beta\, t = (V, L, \sigma), \\
& & \qquad \sigma \subseteq \bar{\gamma}_{\mathcal{V}^\sharp} \circ \sigma^{\sharp'}, V \sqsubseteq V^{\sharp'}, L \sqsubseteq L^{\sharp'}\}
\end{array}
$$

For every trace $t \in \eta^{i-1}\,[u, S, A_0]$, let $\beta\, t = (V, L, \sigma)$. By induction hypothesis, $V \sqsubseteq V^\sharp$, $L \sqsubseteq L^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let $t' = [\![(u, \mathsf{unlock}(a), u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S \setminus \{a\}$, $\alpha_{\mathcal{A}}\, t' = A'$, and

$$
\begin{array}{rcl}
\beta\, t' & = & (V', L', \sigma') \qquad \text{where} \\
V' & = & \{a \mapsto \{g \mid g \in \mathcal{G}, (\_, g = x, \bar{u}') = \mathsf{last\_tl\_write}_g\, t', \bar{u} \le \bar{u}'\} \mid \\
& & \qquad a \in \mathcal{M}, (\_, \mathsf{lock}(a), \bar{u}) = \mathsf{last\_tl\_lock}_a\, t'\} \cup \\
& & \{a \mapsto \{g \mid g \in \mathcal{G}, (\_, g = x, \_) = \mathsf{last\_tl\_write}_g\, t'\} \mid \\
& & \qquad a \in \mathcal{M}, \bot = \mathsf{last\_tl\_lock}_a\, t'\} \\
& = & V \\
L' & = & \{a \mapsto \{L_{t'}[\bar{u}]\} \mid a \in \mathcal{M}, (\bar{u}, \mathsf{lock}(a), \_) = \mathsf{last\_tl\_lock}_a\, t'\} \cup \\
& & \{a \mapsto \emptyset \mid a \in \mathcal{M}, \bot = \mathsf{last\_tl\_lock}_a\, t'\} \\
& = & L \\
\sigma' & = & \{x \mapsto \{t'(x)\} \mid x \in \mathcal{X}\} \\
& & \cup \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last\_write}_g\, t'\} \\
& & \cup \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\, t', \bot \neq \mathsf{last\_write}_g\, t'\} \\
& & \cup \{g \mapsto \{\sigma_{j-1}\, x\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x, \_) = \mathsf{last\_tl\_write}_g\, t'\} \\
& = & \sigma
\end{array}
$$

Thus, $V = V' \sqsubseteq V^\sharp \sqsubseteq V^{\sharp'}$, and $L = L' \sqsubseteq L^\sharp \sqsubseteq L^{\sharp'}$, and $\sigma' = \sigma \subseteq \bar{\gamma} \circ \sigma^\sharp \subseteq \bar{\gamma} \circ \sigma^{\sharp'}$. Altogether, $t' \in \eta_{\mathsf{lc}}\,[u', S \setminus \{a\}, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], \mathsf{unlock}(a), u'), A']\!]_{\mathsf{lc}}\,\eta^{i-1}$ is subsumed by the value of the unknown $\eta_{\mathsf{lc}}\,[u', S \cup \{a\}, A']$. Next, we consider the side-effects of the corresponding right-hand-side functions. For each $g \in \mathcal{G}$, we distinguish two cases for t':

- $\mathsf{last\_tl\_write}_g\, t' = \bot$: Then, a side-effect $\{[g, a, S \setminus \{a\}, A'] \mapsto \{t'\}\}$ is caused. This side-effect is accounted for by construction of η_{lc}:

$$\begin{array}{rcl} t' & \in & \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{unlock}(a), L_t = S \setminus \{a\}, \alpha_{\mathcal{A}}\, t = A, \mathsf{last\_tl\_write}_g\, t = \bot\} \\ & \subseteq & \eta_{\mathsf{lc}}\, [g, a, S \setminus \{a\}, A] \end{array}$$

- $\mathsf{last\_tl\_write}_g\, t' = ((j-1, u_{j-1}, \sigma_{j-1}), g = x, \bar{u}')$ with $(j-1, u_{j-1}, \sigma_{j-1}) = \bar{u}$: Then the side-effects caused to unknowns associated with g in $\mathcal{C}^\sharp_{\mathsf{lc}}$ and $\mathcal{C}^\sharp$, respectively, are given by

$$\begin{array}{rcl} \rho' & = & \{[g, a, S \setminus \{a\}, A'] \mapsto \{t'\}\} \\ \rho^{\sharp'} & = & \{[g, a, S \setminus \{a\}, A'] \mapsto \sigma^\sharp\, g\} \end{array}$$

  We have $\sigma\, g = \{\sigma_{j-1}\, x\} \subseteq (\bar{\gamma} \circ \sigma^\sharp) g$. Furthermore, as $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, we have $\sigma^\sharp\, g \sqsubseteq \eta^\sharp\, [g, a, S \setminus \{a\}, A']$. Thus, we have

$$t' \in \gamma_{g,a,S\setminus\{a\},A'}(\sigma^\sharp\, g) \subseteq \eta_{\mathsf{lc}}\, [g, a, S \setminus \{a\}, A']$$

Hence, all side-effects for $\mathsf{unlock}(a)$ of $\mathcal{C}_{\mathsf{lc}}$ are accounted for in η_{lc}, and the claim holds.

Next, consider the constraints corresponding to **starting a new thread**. Consider an edge $(u, x = \mathsf{create}(u_1), u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, x = \mathsf{create}(u_1)]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![([u, S, A_0], x = \mathsf{create}(u_1), u')]\!]_{\mathsf{lc}}\, \eta^{i-1} \quad \subseteq \quad (\eta_{\mathsf{lc}}, \eta_{\mathsf{lc}}\, [u', S, A'])$$

We have

$$\begin{array}{l} [\![([u, S, A_0], x = \mathsf{create}(u_1), u')]\!]_{\mathsf{lc}}\, \eta = \\ \quad \textbf{let } T = [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\eta\, [u, S, A_0]) \textbf{ in} \\ \quad \textbf{let } \rho = \{[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0] \mapsto \mathsf{new}\, u_1\, (\eta\, [u, S, A_0])\} \textbf{ in} \\ \quad (\rho, T) \end{array}$$

$$\begin{array}{l} [\![[u, S, A_0], x = \mathsf{create}(u_1)]\!]^\sharp \eta^\sharp = \\ \quad \textbf{let } (V^\sharp, L^\sharp, \sigma^\sharp) = \eta^\sharp\, [u, S, A_0] \textbf{ in} \\ \quad \textbf{let } V^\sharp_\rho = \{a \mapsto \emptyset \mid a \in \mathcal{M}\} \textbf{ in} \\ \quad \textbf{let } L^\sharp_\rho = \{a \mapsto \emptyset \mid a \in \mathcal{M}\} \textbf{ in} \\ \quad \textbf{let } i = \nu^\sharp\, u\, \sigma\, u_1 \textbf{ in} \\ \quad \textbf{let } \sigma^\sharp_\rho = \sigma^\sharp \oplus \left(\{\mathsf{self} \mapsto i\} \cup \left\{g \mapsto \left(\sigma\, g \sqcap [\![0]\!]^\sharp_{\mathcal{E}xp}\, \top\right) \mid g \in \mathcal{G}\right\}\right) \textbf{ in} \\ \quad \textbf{let } \rho^\sharp = \{[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0] \mapsto (V^\sharp_\rho, L^\sharp_\rho, \sigma^\sharp_\rho)\} \textbf{ in} \\ \quad \textbf{let } \sigma^{\sharp''} = \sigma^\sharp \oplus \{x \mapsto i\} \textbf{ in} \\ \quad (\rho^\sharp, (V^\sharp, L^\sharp, \sigma^{\sharp''})) \end{array}$$

where we, for notational convenience, denote by $\mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0$ the only element of this singleton set. Let $\eta^\sharp\, [u, S, A_0] = (V^\sharp, L^\sharp, \sigma^\sharp)$ and $\eta^\sharp\, [u', S, A'] = (V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'})$ the value

provided by $\eta^\sharp$ for the endpoint of the given control-flow edge, the appropriate lockset, and the resulting digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $V^\sharp \sqsubseteq V^{\sharp'}$, $L^\sharp \sqsubseteq L^{\sharp'}$, and $\sigma^{\sharp''} \sqsubseteq \sigma^{\sharp'}$ all hold. Then, by definition:

$$\begin{array}{rcl} \eta_{\mathsf{lc}}[u', S, A'] & = & \gamma_{u',S,A'}(V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'}) \\ & = & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = S, \alpha_{\mathcal{A}}\, t = A', \beta\, t = (V, L, \sigma), \\ & & \qquad \sigma \subseteq \bar{\gamma}_{\bar{\mathcal{V}}^\sharp} \circ \sigma^{\sharp'}, V \sqsubseteq V^{\sharp'}, L \sqsubseteq L^{\sharp'}\} \end{array}$$

For every trace $t \in \eta^{i-1}\,[u, S, A_0]$, let $\beta\, t = (V, L, \sigma)$. By induction hypothesis, $V \sqsubseteq V^\sharp$, $L \sqsubseteq L^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let $t' = [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S$, $\alpha_{\mathcal{A}}\, t' = A'$, and

$$\begin{array}{rcl} \beta\, t' & = & (V', L', \sigma') \qquad \text{where} \\ V' & = & \{a \mapsto \{g \mid g \in \mathcal{G}, (\_, g = x, \bar{u}') = \mathsf{last\_tl\_write}_g\, t', \bar{u} \le \bar{u}'\} \mid \\ & & \qquad a \in \mathcal{M}, (\_, \mathsf{lock}(a), \bar{u}) = \mathsf{last\_tl\_lock}_a\, t'\} \cup \\ & & \{a \mapsto \{g \mid g \in \mathcal{G}, (\_, g = x, \_) = \mathsf{last\_tl\_write}_g\, t'\} \mid \\ & & \qquad a \in \mathcal{M}, \bot = \mathsf{last\_tl\_lock}_a\, t'\} \\ & = & V \\ L' & = & \{a \mapsto \{L_{t'}[\bar{u}]\} \mid a \in \mathcal{M}, (\bar{u}, \mathsf{lock}(a), \_) = \mathsf{last\_tl\_lock}_a\, t'\} \cup \\ & & \{a \mapsto \emptyset \mid a \in \mathcal{M}, \bot = \mathsf{last\_tl\_lock}_a\, t'\} \\ & = & L \\ \sigma' & = & \{x' \mapsto \{t'(x')\} \mid x' \in \mathcal{X}\} \\ & & \cup\, \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last\_write}_g\, t'\} \\ & & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\, t', \bot \neq \mathsf{last\_write}_g\, t'\} \\ & & \cup\, \{g \mapsto \{\sigma_{j-1}\, x'\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x', \_) = \mathsf{last\_tl\_write}_g\, t'\} \\ & = & \sigma \oplus \{x \mapsto \{\nu(t)\}\} \end{array}$$

Thus $V' = V \sqsubseteq V^\sharp \sqsubseteq V^{\sharp'}$, $L' = L \sqsubseteq L^\sharp \sqsubseteq L^{\sharp'}$, and as $\nu(t) \in \gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}(\nu^\sharp\, u\, \sigma^\sharp\, u_1)$ by (3.2),

$$\begin{array}{rcl} \sigma' & = & \sigma \oplus \{x \mapsto \{\nu(t)\}\} \subseteq \bar{\gamma} \circ (\sigma^\sharp \oplus \{x \mapsto \nu^\sharp\, u\, \sigma^\sharp\, u_1\}) = \bar{\gamma} \circ \sigma^{\sharp''} \\ & \subseteq & \bar{\gamma} \circ \sigma^{\sharp'} \end{array}$$

Altogether, $t' \in \eta_{\mathsf{lc}}\,[u', S, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], x = \mathsf{create}(u_1), u')]\!]_{\mathsf{lc}}\, \eta^{i-1}$ is subsumed by the value $\eta_{\mathsf{lc}}\,[u', S, A']$.

Next, we consider the side-effects of the corresponding right-hand-side functions for a t as considered previously.

$$\begin{array}{rcl} \rho & = & \{[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0] \mapsto \mathsf{new}\, u_1\, t\} \\ \rho^\sharp & = & \{[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0] \mapsto (V^\sharp_\rho, L^\sharp_\rho, \sigma^\sharp_\rho)\} \end{array}$$

Let $\eta^\sharp\,[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0] = (V^{\sharp'}_\rho, L^{\sharp'}_\rho, \sigma^{\sharp'}_\rho)$ the value provided by $\mathcal{C}^\sharp$ for the unknown receiving the side-effect. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $V^\sharp_\rho \sqsubseteq V^{\sharp'}_\rho$, $L^\sharp_\rho \sqsubseteq L^{\sharp'}_\rho$, and $\sigma^\sharp_\rho \sqsubseteq \sigma^{\sharp'}_\rho$

hold. By definition:

$$\begin{aligned}
& \eta_{\mathsf{lc}}[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0] \\
= \;& \gamma_{u_1,\emptyset,\mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0}(V^{\sharp'}_\rho, L^{\sharp'}_\rho, \sigma^{\sharp'}_\rho) \\
= \;& \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = \emptyset, \alpha_{\mathcal{A}}\, t = \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0, \beta\, t = (V, L, \sigma), \\
& \qquad \sigma \subseteq \bar{\gamma}_{\bar{\mathcal{V}}^\sharp} \circ \sigma^{\sharp'}_\rho, V \sqsubseteq V^{\sharp'}_\rho, L \sqsubseteq L^{\sharp'}_\rho\}
\end{aligned}$$

Let $t'' = \mathsf{new}\, u_1\, \{t\}$. Then, $\mathsf{loc}\, t'' = u_1$, $L_{t''} = \emptyset$, $\alpha_{\mathcal{A}}\, t'' = \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0$, and

$$\begin{aligned}
\beta\, t'' &= (V_\rho, L_\rho, \sigma_\rho) \qquad \text{where} \\
V_\rho &= \{a \mapsto \{g \mid g \in \mathcal{G}, (_, g = x, \bar{u}') = \mathsf{last_tl_write}_g\, t'', \bar{u} \leq \bar{u}'\} \mid \\
&\qquad a \in \mathcal{M}, (_, \mathsf{lock}(a), \bar{u}) = \mathsf{last_tl_lock}_a\, t''\} \cup \\
&\quad \{a \mapsto \{g \mid g \in \mathcal{G}, (_, g = x, _) = \mathsf{last_tl_write}_g\, t''\} \mid \\
&\qquad a \in \mathcal{M}, \bot = \mathsf{last_tl_lock}_a\, t''\} \\
&= \{a \mapsto \emptyset \mid a \in \mathcal{M}\} \\
L_\rho &= \{a \mapsto \{L_{t''}[\bar{u}]\} \mid a \in \mathcal{M}, (\bar{u}, \mathsf{lock}(a), _) = \mathsf{last_tl_lock}_a\, t''\} \cup \\
&\quad \{a \mapsto \emptyset \mid a \in \mathcal{M}, \bot = \mathsf{last_tl_lock}_a\, t''\} \\
&= \{a \mapsto \emptyset \mid a \in \mathcal{M}\} \\
\sigma_\rho &= \{x' \mapsto \{t''(x')\} \mid x' \in \mathcal{X}\} \\
&\quad \cup \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\, t''\} \\
&\quad \cup \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t'', \bot \neq \mathsf{last_write}_g\, t''\} \\
&\quad \cup \{g \mapsto \{\sigma_{j-1}\, x'\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x', _) = \mathsf{last_tl_write}_g\, t''\} \\
&= \sigma \oplus (\{\mathsf{self} \mapsto \{\nu(t)\}\} \\
&\qquad \cup \{g \mapsto (\{0\} \cap \sigma\, g) \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\, t''\}) \\
&\qquad \cup \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot \neq \mathsf{last_write}_g\, t''\}
\end{aligned}$$

Therefore,

$$\begin{aligned}
V_\rho &= \{a \mapsto \emptyset \mid a \in \mathcal{M}\} = V^\sharp_\rho \sqsubseteq V^{\sharp'}_\rho \\
L_\rho &= \{a \mapsto \emptyset \mid a \in \mathcal{M}\} = L^\sharp_\rho \sqsubseteq L^{\sharp'}_\rho \\
\sigma_\rho &= \sigma \oplus (\{\mathsf{self} \mapsto \{\nu(t)\}\} \\
&\qquad \cup \{g \mapsto (\{0\} \cap \sigma\, g) \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\, t''\}) \\
&\qquad \cup \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot \neq \mathsf{last_write}_g\, t''\} \\
&\subseteq (\bar{\gamma} \circ (\sigma^\sharp \oplus \{\mathsf{self} \mapsto \nu^\sharp\, u\, \sigma^\sharp\, u_1\})) \\
&\quad \oplus (\{g \mapsto (\{0\} \cap \sigma\, g) \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\, t''\} \\
&\qquad \cup \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot \neq \mathsf{last_write}_g\, t''\}) \\
&\subseteq (\bar{\gamma} \circ (\sigma^\sharp \oplus \{\mathsf{self} \mapsto \nu^\sharp\, u\, \sigma^\sharp\, u_1\})) \oplus \{g \mapsto (\{0\} \cap \sigma\, g) \mid g \in \mathcal{G}\} \\
&\subseteq \bar{\gamma} \circ \left(\sigma^\sharp \oplus \left(\{\mathsf{self} \mapsto \nu^\sharp\, u\, \sigma^\sharp\, u_1\} \cup \left\{g \mapsto \left(\sigma^\sharp\, g \sqcap [\![0]\!]^\sharp_{\mathcal{E}xp}\, \top\right) \mid g \in \mathcal{G}\right\}\right)\right) \\
&= \bar{\gamma} \circ \sigma^\sharp_\rho \subseteq \bar{\gamma} \circ \sigma^{\sharp'}_\rho
\end{aligned}$$

Altogether, $t'' \in \eta_{\mathsf{lc}}\, [u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0]$ holds for all $t \in \eta^{i-1}\, [u, S, A_0]$. Hence, all side-effects for $x = \mathsf{create}(u_1)$ of $\mathcal{C}_{\mathsf{lc}}$ are accounted for in η_{lc} and the claim holds.

Next, consider the constraints corresponding to **returning** from a thread. Consider an edge $(u, \mathsf{return}, u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, \mathsf{return}]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![([u, S, A_0], \mathsf{return}, u')]\!]_{\mathsf{lc}}\, \eta^{i-1} \quad \subseteq \quad (\eta_{\mathsf{lc}}, \eta_{\mathsf{lc}}\,[u', S, A'])$$

We have

$$\begin{array}{lcl}
[\![([u, S, A_0], \mathsf{return}, u'), A']\!]_{\mathsf{lc}}\, \eta & = & \textbf{let } T = [\![(u, \mathsf{return}, u')]\!]_{\mathcal{T}}(\eta\,[u, S, A_0]) \textbf{ in} \\
& & \textbf{let } \rho = \{[i, A'] \mapsto \{t \mid t \in T, \mathsf{id}\, t = i\} \mid i \in \mathcal{V}_{\mathbf{tid}}\} \textbf{ in} \\
& & (\rho, T) \\
[\![[u, S, A_0], \mathsf{return}, A']\!]^\sharp \eta^\sharp & = & \textbf{let } (V^\sharp, L^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\
& & \textbf{let } I = \sigma^\sharp\, \mathsf{self} \textbf{ in} \\
& & \textbf{let } v^\sharp_\rho = \sigma^\sharp\, \mathsf{ret} \textbf{ in} \\
& & \textbf{let } \rho^\sharp = \left\{[i, A'] \mapsto v^\sharp_\rho \mid i \in I\right\} \textbf{ in} \\
& & (\rho^\sharp, (V^\sharp, L^\sharp, \sigma^\sharp))
\end{array}$$

Let $\eta^\sharp\,[u, S, A_0] = (V^\sharp, L^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u', S, A'] = (V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'})$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge, the appropriate lockset, and the resulting digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $V^\sharp \sqsubseteq V^{\sharp'}$, $L^\sharp \sqsubseteq L^{\sharp'}$, and $\sigma^\sharp \sqsubseteq \sigma^{\sharp'}$ all hold. Then, by definition:

$$\begin{array}{lcl}
\eta_{\mathsf{lc}}[u', S, A'] & = & \gamma_{u', S, A'}(V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'}) \\
& = & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = S, \alpha_{\mathcal{A}}\, t = A', \beta\, t = (V, L, \sigma), \\
& & \qquad \sigma \subseteq \bar{\gamma}_{\mathcal{V}^\sharp} \circ \sigma^{\sharp'}, V \sqsubseteq V^{\sharp'}, L \sqsubseteq L^{\sharp'}\}
\end{array}$$

For every trace $t \in \eta^{i-1}\,[u, S, A_0]$, let $\beta\, t = (V, L, \sigma)$. By induction hypothesis, $V \sqsubseteq V^\sharp$, $L \sqsubseteq L^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let $t' = [\![(u, \mathsf{return}, u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S$, $\alpha_{\mathcal{A}}\, t' = A'$, and

$$\begin{array}{lcl}
\beta\, t' & = & (V', L', \sigma') \qquad \text{where} \\
V' & = & \{a \mapsto \{g \mid g \in \mathcal{G}, (_, g = x, \bar{u}') = \mathsf{last_tl_write}_g\, t', \bar{u} \leq \bar{u}'\} \mid \\
& & \qquad a \in \mathcal{M}, (_, \mathsf{lock}(a), \bar{u}) = \mathsf{last_tl_lock}_a\, t'\} \cup \\
& & \{a \mapsto \{g \mid g \in \mathcal{G}, (_, g = x, _) = \mathsf{last_tl_write}_g\, t'\} \mid \\
& & \qquad a \in \mathcal{M}, \bot = \mathsf{last_tl_lock}_a\, t'\} \\
& = & V \\
L' & = & \{a \mapsto \{L_{t'}[\bar{u}]\} \mid a \in \mathcal{M}, (\bar{u}, \mathsf{lock}(a), _) = \mathsf{last_tl_lock}_a\, t'\} \cup \\
& & \{a \mapsto \emptyset \mid a \in \mathcal{M}, \bot = \mathsf{last_tl_lock}_a\, t'\} \\
& = & L \\
\sigma' & = & \{x' \mapsto \{t'(x')\} \mid x' \in \mathcal{X}\} \\
& & \cup\, \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\, t'\} \\
& & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t', \bot \neq \mathsf{last_write}_g\, t'\} \\
& & \cup\, \{g \mapsto \{\sigma_{j-1}\, x'\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x', _) = \mathsf{last_tl_write}_g\, t'\} \\
& = & \sigma
\end{array}$$

Thus $V' = V \sqsubseteq V^\sharp \sqsubseteq V^{\sharp'}$, $L' = L \sqsubseteq L^\sharp \sqsubseteq L^{\sharp'}$, and $\sigma' = \sigma \subseteq \bar{\gamma} \circ \sigma^\sharp \subseteq \bar{\gamma} \circ \sigma^{\sharp'}$. Altogether, $t' \in \eta_{\mathsf{lc}}\,[u', S, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], \mathsf{return}, u')]\!]_{\mathsf{lc}}\,\eta^{i-1}$ is subsumed by the value $\eta_{\mathsf{lc}}\,[u', S, A']$.

Next, we consider the side-effects of the corresponding right-hand-side functions for local traces t and t' as considered previously.

$$\begin{array}{lcl} \rho & = & \{[\mathsf{id}\,t', A'] \mapsto \{t'\}\} \\ \rho^\sharp & = & \left\{[i^\sharp, A'] \mapsto v_\rho^\sharp \mid i^\sharp \in \sigma^\sharp\,\mathsf{self}\right\} \end{array}$$

where we use that $\mathsf{id}\,t = \mathsf{id}\,t'$. As $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$, we have $\{\mathsf{id}\,t'\} = \sigma\,\mathsf{self} \subseteq \gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}(\sigma^\sharp\,\mathsf{self})$. As $\mathcal{V}^\sharp_{\mathbf{tid}}$ is a powerset lattice, and the concretization is defined by the union of concretizations of the singleton sets, there is at least one $i_\rho^\sharp \in \sigma^\sharp\,\mathsf{self}$, such that $\mathsf{id}\,t' \in \gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}\{i_\rho^\sharp\}$. Consider one such $i_\rho^\sharp$, and let $\eta^\sharp\,[i_\rho^\sharp, A'] = v_\rho^{\sharp'}$ the value provided by $\mathcal{C}^\sharp$ for this unknown receiving the side-effect. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $v_\rho^\sharp \sqsubseteq v_\rho^{\sharp'}$ holds. By definition:

$$\eta_{\mathsf{lc}}[\mathsf{id}\,t', A'] = \bigcup\{\gamma_{(\mathsf{id}\,t'),A'}(\eta^\sharp\,[i^\sharp, A']) \mid i^\sharp \in S_{\mathcal{V}^\sharp_{\mathbf{tid}}}, \mathsf{id}\,t' \in (\gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}\{i^\sharp\})\}$$

Now consider

$$\gamma_{(\mathsf{id}\,t'),A'}(\eta^\sharp\,[i_\rho^\sharp, A']) \qusubseteq \quad \eta_{\mathsf{lc}}[\mathsf{id}\,t', A']$$

Then, by definition:

$$\begin{array}{lcl} & & \gamma_{(\mathsf{id}\,t'),A'}(\eta^\sharp\,[i_\rho^\sharp, A']) = \gamma_{(\mathsf{id}\,t'),A'}\left(v_\rho^{\sharp'}\right) \\ & = & \{t'' \in \mathcal{T} \mid \mathsf{last}\,t'' = \mathsf{return}, \alpha_{\mathcal{A}}\,t'' = A', \mathsf{id}\,t'' = \mathsf{id}\,t', t''(\mathsf{ret}) \in \gamma_{\mathcal{V}^\sharp}(v_\rho^{\sharp'})\} \end{array}$$

Then $\mathsf{last}\,t' = \mathsf{return}$, $\alpha_{\mathcal{A}}\,t' = A'$, (vacuously $\mathsf{id}\,t' = \mathsf{id}\,t'$), and

$$t'(\mathsf{ret}) \in \sigma\,\mathsf{ret} \subseteq (\bar{\gamma} \circ \sigma^\sharp)\,\mathsf{ret} \subseteq \gamma_{\mathcal{V}^\sharp}((\sigma^\sharp)\,\mathsf{ret}) = \gamma_{\mathcal{V}^\sharp}(v_\rho^\sharp) \subseteq \gamma_{\mathcal{V}^\sharp}(v_\rho^{\sharp'})$$

Altogether, $t' \in \eta_{\mathsf{lc}}\,[\mathsf{id}\,t, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. Hence, all side-effects for return of $\mathcal{C}_{\mathsf{lc}}$ are accounted for in η_{lc}, and the claim holds.

Next, consider the constraints corresponding to calling **join**. Consider an edge $(u, x = \mathsf{join}(x'), u') \in \mathcal{E}$ and digests A', A_0 and all appropriate A_1 such that $A' \in [\![u, x = \mathsf{join}(x')]\!]^\sharp_{\mathcal{A}}(A_0, A_1)$. We verify that

$$[\![([u, S, A_0], x = \mathsf{join}(x'), u')]\!]_{\mathsf{lc}}\,\eta^{i-1} \quad \subseteq \quad (\eta_{\mathsf{lc}}, \eta_{\mathsf{lc}}\,[u', S, A'])$$

We have

$$\begin{array}{l} [\![([u, S, A_0], x = \mathsf{join}(x'), u')]\!]_{\mathsf{lc}}\,\eta = \\ \quad \textbf{let}\ T_1 = \bigcup\{\eta\,[t(x'), A_1] \mid t \in \eta\,[u, S, A_0], A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1\}\ \textbf{in} \\ \quad \textbf{let}\ T = [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\eta\,[u, S, A_0], T_1)\ \textbf{in} \\ \quad (\emptyset, T) \end{array}$$

$$
\begin{array}{l}
[\![[u, S, A_0], x = \mathsf{join}(x')]\!]^\sharp \eta^\sharp = \\
\quad \textbf{let } (V^\sharp, L^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\
\quad \textbf{let } v^\sharp = \bigsqcup_{i' \in (\sigma^\sharp\, x')} \left(\bigsqcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1} (\eta^\sharp[i', A_1]) \right) \textbf{ in} \\
\quad \textbf{if } v^\sharp = \bot \textbf{ then} \\
\qquad (\emptyset, \bot) \\
\quad \textbf{else} \\
\qquad \textbf{let } \sigma^{\sharp\prime\prime} = \sigma^\sharp \oplus \{x \mapsto v^\sharp\} \textbf{ in} \\
\qquad (\emptyset, (V^\sharp, L^\sharp, \sigma^{\sharp\prime\prime}))
\end{array}
$$

Let $\eta^\sharp\,[u, S, A_0] = (V^\sharp, L^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u', S, A'] = (V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'})$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge, the appropriate lockset, and the resulting digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, we either have

(1) $v^\sharp = \bot$; or

(2) $V^\sharp \sqsubseteq V^{\sharp'}$, $L^\sharp \sqsubseteq L^{\sharp'}$, and $\sigma^{\sharp\prime\prime} \sqsubseteq \sigma^{\sharp'}$ all hold.

Then, by definition:

$$
\begin{array}{rcl}
\eta_{\mathsf{lc}}[u', S, A'] & = & \gamma_{u', S, A'}(V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'}) \\
& = & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = S, \alpha_{\mathcal{A}}\, t = A', \beta\, t = (V, L, \sigma), \\
& & \qquad \sigma \subseteq \bar{\gamma}_{\bar{\mathcal{V}}^\sharp} \circ \sigma^{\sharp'}, V \sqsubseteq V^{\sharp'}, L \sqsubseteq L^{\sharp'}\}
\end{array}
$$

For every trace $t \in \eta^{i-1}\,[u, S, A_0]$, let $\beta\, t = (V, L, \sigma)$. By induction hypothesis, $V \sqsubseteq V^\sharp$, $L \sqsubseteq L^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let

$$
\begin{array}{rcl}
T' & = & [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\{t\}, \\
& & \quad \bigcup\{\eta^{i-1}\,[(t'(x')), A_1] \mid t' \in \eta^{i-1}\,[u, S, A_0], A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\}) \\
& = & [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\{t\}, \bigcup\{\eta^{i-1}\,[(t(x')), A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\})
\end{array}
$$

where the equality exploits that for a given local trace t, $[\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\{t\}, \{t'\})$ only yields a non-empty set if the thread *id* of the thread being joined is the one stored in x'. We distinguish the case where the resulting set of traces is non-empty and the case where it is empty. If T' is empty, it is subsumed by $\eta_{\mathsf{lc}}\,[u', S, A']$ vacuously.

Consider thus a $t' \in T'$ and $t'' \in \bigcup\{\eta^{i-1}\,[(t(x')), A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\}$ such that $\{t'\} = [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\{t\}, \{t''\})$ Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S$, $\alpha_{\mathcal{A}}\, t' = A'$, and

$$
\begin{array}{rcl}
\beta\, t' & = & (V', L', \sigma') \qquad \text{where} \\
V' & = & \{a \mapsto \{g \mid g \in \mathcal{G}, (\_, g = x, \bar{u}') = \mathsf{last\_tl\_write}_g\, t', \bar{u} \leq \bar{u}'\} \mid \\
& & \qquad a \in \mathcal{M}, (\_, \mathsf{lock}(a), \bar{u}) = \mathsf{last\_tl\_lock}_a\, t'\} \cup \\
& & \{a \mapsto \{g \mid g \in \mathcal{G}, (\_, g = x, \_) = \mathsf{last\_tl\_write}_g\, t'\} \mid \\
& & \qquad a \in \mathcal{M}, \bot = \mathsf{last\_tl\_lock}_a\, t'\} \\
& = & V \\
L' & = & \{a \mapsto \{L_{t'}[\bar{u}]\} \mid a \in \mathcal{M}, (\bar{u}, \mathsf{lock}(a), \_) = \mathsf{last\_tl\_lock}_a\, t'\} \cup \\
& & \{a \mapsto \emptyset \mid a \in \mathcal{M}, \bot = \mathsf{last\_tl\_lock}_a\, t'\} \\
& = & L
\end{array}
$$

$$\begin{array}{rcl}
\sigma' & = & \{x'' \mapsto \{t'(x'')\} \mid x'' \in \mathcal{X}\} \\
& & \cup\, \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\, t'\} \\
& & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t', \bot \neq \mathsf{last_write}_g\, t'\} \\
& & \cup\, \{g \mapsto \{\sigma_{j-1}\, x''\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x'', _) = \mathsf{last_tl_write}_g\, t'\} \\
& = & \sigma \oplus (\{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t', \bot \neq \mathsf{last_write}_g\, t'\} \\
& & \cup \{x \mapsto t''(\mathsf{ret})\}) \\
& \subseteq & \sigma \oplus \{x \mapsto t''(\mathsf{ret})\}
\end{array}$$

Thus $V' = V \sqsubseteq V^\sharp$ and $L' = L \sqsubseteq L^\sharp$. To show that $t' \in \eta_{\mathsf{lc}}\,[u', S, A']$, it thus remains to show that $v^\sharp \neq \bot$ holds (and thus $V^\sharp \sqsubseteq V^{\sharp'}$, $L^\sharp \sqsubseteq L^{\sharp'}$ also hold) and to relate $\sigma \oplus \{x \mapsto t''(\mathsf{ret})\}$ to $\bar{\gamma} \circ \sigma^{\sharp'}$. To this end, we first relate $t''(\mathsf{ret})$ to $v^\sharp$. We have

$$\begin{array}{rcl}
t'' & \in & \bigcup\{\eta^{i-1}\,[t(x'), A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\} \\
& \subseteq & \bigcup\{\eta_{\mathsf{lc}}\,[t(x'), A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\} \\
& = & \bigcup\{\bigcup\{\gamma_{t(x'),A_1}(\eta^\sharp\,[i^\sharp, A_1]) \mid t(x') \in (\gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}\,\{i^\sharp\})\} \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\} \\
& \subseteq & \bigcup\{\bigcup\{\gamma_{t(x'),A_1}(\eta^\sharp\,[i^\sharp, A_1]) \mid i^\sharp \in \sigma^\sharp\, x'\} \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\}
\end{array}$$

and therefore

$$\begin{array}{rcl}
& & \{t''(\mathsf{ret})\} \\
& \subseteq & \{t'''(\mathsf{ret}) \mid t''' \in (\bigcup\{\bigcup\{\gamma_{t(x'),A_1}(\eta^\sharp\,[i^\sharp, A_1]) \mid i^\sharp \in \sigma^\sharp\, x'\} \\
& & \qquad\qquad \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\})\} \\
& \subseteq & \{t'''(\mathsf{ret}) \mid t''' \in (\bigcup\{\gamma_{t(x'),A_1} \bigsqcup\{(\eta^\sharp\,[i^\sharp, A_1]) \mid i^\sharp \in \sigma^\sharp\, x'\} \\
& & \qquad\qquad \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\})\} \\
& \subseteq & \{t'''(\mathsf{ret}) \mid t''' \in (\bigcup\{\{\bar{t} \in \mathcal{T} \mid \mathsf{last}\,(\bar{t}) = \mathsf{return}, \alpha_{\mathcal{A}}\,(\bar{t}) = A_1, \mathsf{id}\,\bar{t} = t(x'), \\
& & \qquad\qquad\qquad \bar{t}(\mathsf{ret}) \in \gamma_{\mathcal{V}^\sharp}(\bigsqcup\{(\eta^\sharp\,[i^\sharp, A_1]) \mid i^\sharp \in \sigma^\sharp\, x'\})\} \\
& & \qquad\qquad \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\})\} \qquad \text{(by def. of } \gamma_{t(x'),A_1}) \\
& \subseteq & \bigcup\{\{\bar{t}(\mathsf{ret}) \mid \bar{t} \in \mathcal{T}, \mathsf{last}\,(\bar{t}) = \mathsf{return}, \alpha_{\mathcal{A}}\,(\bar{t}) = A_1, \mathsf{id}\,\bar{t} = t(x'), \\
& & \qquad\qquad\qquad \bar{t}(\mathsf{ret}) \in \gamma_{\mathcal{V}^\sharp}(\bigsqcup\{(\eta^\sharp\,[i^\sharp, A_1]) \mid i^\sharp \in \sigma^\sharp\, x'\})\} \\
& & \qquad\qquad \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\} \\
& \subseteq & \bigcup\{\gamma_{\mathcal{V}^\sharp}(\bigsqcup\{(\eta^\sharp\,[i^\sharp, A_1]) \mid i^\sharp \in \sigma^\sharp\, x'\}) \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\} \\
& \subseteq & \gamma_{\mathcal{V}^\sharp}(\bigsqcup\{\bigsqcup\{(\eta^\sharp\,[i^\sharp, A_1]) \mid i^\sharp \in \sigma^\sharp\, x'\} \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\}) \\
& \subseteq & \gamma_{\mathcal{V}^\sharp}\left(\bigsqcup_{i^\sharp \in \sigma^\sharp\, x'}\left(\bigsqcup_{A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1}(\eta[i^\sharp, A_1])\right)\right) \\
& = & \gamma_{\mathcal{V}^\sharp}(v^\sharp)
\end{array}$$

As a consequence and as $\gamma_{\mathcal{V}^\sharp}(\bot) = \emptyset$, we obtain $v^\sharp \neq \bot$. Thus, we also have $V^\sharp \sqsubseteq V^{\sharp'}$, $L^\sharp \sqsubseteq L^{\sharp'}$, and $\sigma^{\sharp''} \sqsubseteq \sigma^{\sharp'}$. Also,

$$\sigma' \subseteq \sigma \oplus \{x \mapsto t''(\mathsf{ret})\} \subseteq (\bar{\gamma} \circ \sigma^\sharp) \oplus \{x \mapsto t''(\mathsf{ret})\} \subseteq \bar{\gamma} \circ (\sigma^\sharp \oplus \{x \mapsto v^\sharp\}) = \bar{\gamma} \circ \sigma^{\sharp''}$$

Thus, $V' = V \sqsubseteq V^\sharp \sqsubseteq V^{\sharp'}$, $L' = L \sqsubseteq L^\sharp \sqsubseteq L^{\sharp'}$, and $\sigma' \subseteq \bar{\gamma} \circ \sigma^{\sharp''} \subseteq \bar{\gamma} \circ \sigma^{\sharp'}$. Altogether, $t' \in \eta_{\mathsf{lc}}\,[u', S, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], x = \mathsf{join}(x'), u'), A']\!]_{\mathsf{lc}}\, \eta^{i-1}$ is subsumed by the value $\eta_{\mathsf{lc}}\,[u', S, A']$ in both cases. As neither constraint causes any side-effects, the statement holds.

Next, consider the constraints corresponding to calling **signal**. Consider an edge $(u, \mathsf{signal}(s), u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, \mathsf{signal}(s)]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![([u, S, A_0], \mathsf{signal}(s), u'), A']\!]_{\mathsf{lc}}\, \eta^{i-1} \quad \subseteq \quad (\eta_{\mathsf{lc}}, \eta_{\mathsf{lc}}\,[u', S, A'])$$

We have

$$\begin{array}{l}
[\![([u, S, A_0], \mathsf{signal}(s), u'), A']\!]_{\mathsf{lc}}\, \eta = \\
\quad \textbf{let } T = [\![(u, \mathsf{signal}(s), u')]\!]_{\mathcal{T}}(\eta\,[u, S, A_0]) \textbf{ in} \\
\quad \textbf{let } \rho = \{[s, A'] \mapsto T\} \textbf{ in} \\
\quad (\rho, T)
\end{array}$$

$$\begin{array}{l}
[\![[u, S, A_0], \mathsf{signal}(s), A']\!]^\sharp \eta^\sharp = \\
\quad \textbf{let } (V^\sharp, L^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\
\quad \textbf{let } \rho^\sharp = \{[s, A'] \mapsto (V^\sharp, L^\sharp, \sigma^\sharp)\} \textbf{ in} \\
\quad (\rho^\sharp, (V^\sharp, L^\sharp, \sigma^\sharp))
\end{array}$$

Let $\eta^\sharp\,[u, S, A_0] = (V^\sharp, L^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u', S, A'] = (V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'})$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge, the appropriate lockset, and the resulting digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, we have $V^\sharp \sqsubseteq V^{\sharp'}$, $L^\sharp \sqsubseteq L^{\sharp'}$, and $\sigma^\sharp \sqsubseteq \sigma^{\sharp'}$. Then, by definition:

$$\begin{array}{rcl}
\eta_{\mathsf{lc}}[u', S, A'] & = & \gamma_{u', S, A'}(V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'}) \\
& = & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = S, \alpha_{\mathcal{A}}\, t = A', \beta\, t = (V, L, \sigma), \\
& & \qquad \sigma \subseteq \bar{\gamma}_{\mathcal{V}^\sharp} \circ \sigma^{\sharp'}, V \sqsubseteq V^{\sharp'}, L \sqsubseteq L^{\sharp'}\}
\end{array}$$

For every trace $t \in \eta^{i-1}\,[u, S, A_0]$, let $\beta\, t = (V, L, \sigma)$. By induction hypothesis, $V \sqsubseteq V^\sharp$, $L \sqsubseteq L^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let $t' = [\![(u, \mathsf{signal}(s), u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S$, $\alpha_{\mathcal{A}}\, t' = A'$, and

$$\begin{array}{rcll}
\beta\, t' & = & (V', L', \sigma') & \text{where} \\
V' & = & \{a \mapsto \{g \mid g \in \mathcal{G}, (_, g = x, \bar{u}') = \mathsf{last_tl_write}_g\, t', \bar{u} \le \bar{u}'\} \mid & \\
& & \qquad a \in \mathcal{M}, (_, \mathsf{lock}(a), \bar{u}) = \mathsf{last_tl_lock}_a\, t'\} \cup & \\
& & \{a \mapsto \{g \mid g \in \mathcal{G}, (_, g = x, _) = \mathsf{last_tl_write}_g\, t'\} \mid & \\
& & \qquad a \in \mathcal{M}, \bot = \mathsf{last_tl_lock}_a\, t'\} & \\
& = & V & \\
L' & = & \{a \mapsto \{L_{t'}[\bar{u}]\} \mid a \in \mathcal{M}, (\bar{u}, \mathsf{lock}(a), _) = \mathsf{last_tl_lock}_a\, t'\} \cup & \\
& & \{a \mapsto \emptyset \mid a \in \mathcal{M}, \bot = \mathsf{last_tl_lock}_a\, t'\} & \\
& = & L & \\
\sigma' & = & \{x' \mapsto \{t'(x')\} \mid x' \in \mathcal{X}\} & \\
& & \cup\, \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\, t'\} & \\
& & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last_tl_write}_g\, t', \bot \neq \mathsf{last_write}_g\, t'\} & \\
& & \cup\, \{g \mapsto \{\sigma_{j-1}\, x'\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x', _) = \mathsf{last_tl_write}_g\, t'\} & \\
& = & \sigma &
\end{array}$$

Thus $V' = V \sqsubseteq V^\sharp \sqsubseteq V^{\sharp'}$, $L' = L \sqsubseteq L^\sharp \sqsubseteq L^{\sharp'}$, and $\sigma' = \sigma \subseteq \bar{\gamma} \circ \sigma^\sharp \subseteq \bar{\gamma} \circ \sigma^{\sharp'}$. Altogether, $t' \in \eta_{\mathsf{lc}}\,[u', S, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], \mathsf{signal}(s), u'), A']\!]_{\mathsf{lc}}\,\eta^{i-1}$ is subsumed by the value $\eta_{\mathsf{lc}}\,[u', S, A']$.

Next, we consider the side-effects of the corresponding right-hand-side functions for local traces t and t' as considered previously.

$$\begin{array}{lcl} \rho & = & \{[s, A'] \mapsto T\} \\ \rho^\sharp & = & \{[s, A'] \mapsto (V^\sharp, L^\sharp, \sigma^\sharp)\} \end{array}$$

Let $\eta^\sharp\,[s, A'] = (V^\sharp_\rho, L^\sharp_\rho, \sigma^\sharp_\rho)$ the value provided by $\mathcal{C}^\sharp$ for the unknown receiving the side-effect. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $V^\sharp \sqsubseteq V^\sharp_\rho$, $L^\sharp \sqsubseteq L^\sharp_\rho$, and $\sigma^\sharp \sqsubseteq \sigma^\sharp_\rho$ hold. By definition:

$$\begin{array}{ll} & \eta_{\mathsf{lc}}[s, A'] = \gamma_{s,A'}(\eta^\sharp\,[s, A']) \\ = & \{t \in \mathcal{T} \mid \mathsf{last}\,t = \mathsf{signal}(s), \alpha_{\mathcal{A}}\,t = A', \beta\,t = (V, L, \sigma), \sigma \subseteq \bar{\gamma} \circ \sigma^\sharp_\rho, V \sqsubseteq V^\sharp_\rho, L \sqsubseteq L^\sharp_\rho\} \end{array}$$

Consider a trace t' as above: Then $\mathsf{last}\,t' = \mathsf{signal}(s)$, $\alpha_{\mathcal{A}}\,t' = A'$, and $V' = V \sqsubseteq V^\sharp \sqsubseteq V^\sharp_\rho$, $L' = L \sqsubseteq L^\sharp \sqsubseteq L^\sharp_\rho$, and $\sigma' = \sigma \subseteq \bar{\gamma} \circ \sigma^\sharp \subseteq \bar{\gamma} \circ \sigma^\sharp_\rho$ Altogether, $t' \in \eta_{\mathsf{lc}}\,[s, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$. Hence, all side-effects for $\mathsf{signal}(s)$ of $\mathcal{C}_{\mathsf{lc}}$ are accounted for in η_{lc}, and the claim holds.

Lastly, consider the constraints corresponding to a call of **wait**. Consider an edge $(u, \mathsf{wait}(s), u') \in \mathcal{E}$ and digests A', A_0 and all appropriate A_1 such that the resulting digest $A' \in [\![u, \mathsf{wait}(s)]\!]^\sharp_{\mathcal{A}}(A_0, A_1)$. We verify that

$$[\![([u, S, A_0], \mathsf{wait}(s), u')]\!]_{\mathsf{lc}}\,\eta^{i-1} \quad \subseteq \quad (\eta_{\mathsf{lc}}, \eta_{\mathsf{lc}}\,[u', S, A'])$$

We have

$$\begin{array}{l} [\![([u, S, A_0], \mathsf{wait}(s), u')]\!]_{\mathsf{lc}}\,\eta = \\ \quad \textbf{let } T_1 = \bigcup\{\eta\,[s, A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A_1\} \textbf{ in} \\ \quad \textbf{let } T = [\![(u, \mathsf{wait}(s), u')]\!]_{\mathcal{T}}(\eta\,[u, S, A_0], T_1) \textbf{ in} \\ \quad (\emptyset, T) \end{array}$$

$$\begin{array}{l} [\![[u, S, A_0], \mathsf{wait}(s)]\!]^\sharp \eta^\sharp = \\ \quad \textbf{let } (V^\sharp, L^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\ \quad \textbf{if } \left(\left(\bigsqcup_{A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A'} \eta^\sharp\,[s, A']\right) = \bot\right) \textbf{ then} \\ \qquad (\emptyset, \bot) \\ \quad \textbf{else} \\ \qquad (\emptyset, (V^\sharp, L^\sharp, \sigma^\sharp)) \end{array}$$

Let $\eta^\sharp\,[u, S, A_0] = (V^\sharp, L^\sharp, \sigma^\sharp)$ and $\eta^\sharp\,[u', S, A'] = (V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'})$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge, the appropriate lockset, and the resulting digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, we either have

(1) $\left(\bigsqcup_{A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\,A_0\,A'} \eta^\sharp\,[s, A']\right) = \bot$; or

(2) $V^\sharp \sqsubseteq V^{\sharp'}$, $L^\sharp \sqsubseteq L^{\sharp'}$, and $\sigma^\sharp \sqsubseteq \sigma^{\sharp'}$ all hold.

Then, by definition:

$$\begin{array}{rcl}\eta_{\mathsf{lc}}[u', S, A'] & = & \gamma_{u',S,A'}(V^{\sharp'}, L^{\sharp'}, \sigma^{\sharp'}) \\ & = & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = S, \alpha_{\mathcal{A}}\, t = A', \beta\, t = (V, L, \sigma), \\ & & \qquad \sigma \subseteq \bar{\gamma}_{\bar{\mathcal{V}}^\sharp} \circ \sigma^{\sharp'}, V \sqsubseteq V^{\sharp'}, L \sqsubseteq L^{\sharp'}\}\end{array}$$

For every trace $t \in \eta^{i-1}\,[u, S, A_0]$, let $\beta\, t = (V, L, \sigma)$. By induction hypothesis, $V \sqsubseteq V^\sharp$, $L \sqsubseteq L^\sharp$, and $\sigma \subseteq \bar{\gamma} \circ \sigma^\sharp$. Let

$$T' \quad = \quad [\![(u, \mathsf{wait}(s), u')]\!]_{\mathcal{T}}(\{t\}, \bigcup\{\eta^{i-1}\,[s, A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\})$$

We distinguish the case where the resulting set of traces is non-empty, and the case where it is. If T' is empty, it is subsumed by $\eta_{\mathsf{lc}}\,[u', S, A']$ vacuously. Consider thus a $t' \in T'$ and $t'' \in \bigcup\{\eta^{i-1}\,[s, A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\}$ such that $\{t'\} = [\![(u, \mathsf{wait}(s), u')]\!]_{\mathcal{T}}(\{t\}, \{t''\})$.

By induction hypothesis, we have

$$\begin{array}{rcl} t'' & \in & \bigcup\{\eta^{i-1}\,[s, A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\} \\ & \subseteq & \bigcup\{\eta_{\mathsf{lc}}\,[s, A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\} \\ & = & \bigcup\{\gamma_{s,A_1}(\eta^\sharp\,[s, A_1]) \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\}\end{array}$$

and thus as $\gamma_{s,A_1}(\bot) = \emptyset$ for any $A_1 \in \mathcal{A}$, there is at least one $A_1 \in \mathcal{A}$ for which $\mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1$ holds and where $\eta^\sharp\,[s, A_1] \neq \bot$. Thus, $\left(\bigsqcup_{A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'} \eta^\sharp\,[s, A']\right) \neq \bot$ and (1) does not hold. We thus have that (2) $V^\sharp \sqsubseteq V^{\sharp'}$, $L^\sharp \sqsubseteq L^{\sharp'}$, and $\sigma^\sharp \sqsubseteq \sigma^{\sharp'}$ all hold.

Consider again the trace t'. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S$, $\alpha_{\mathcal{A}}\, t' = A'$, and

$$\begin{array}{rcll}\beta\, t' & = & (V', L', \sigma') & \text{where} \\ V' & = & \{a \mapsto \{g \mid g \in \mathcal{G}, (\_, g = x, \bar{u}') = \mathsf{last\_tl\_write}_g\, t', \bar{u} \leq \bar{u}'\} \mid & \\ & & \quad a \in \mathcal{M}, (\_, \mathsf{lock}(a), \bar{u}) = \mathsf{last\_tl\_lock}_a\, t'\} \cup & \\ & & \{a \mapsto \{g \mid g \in \mathcal{G}, (\_, g = x, \_) = \mathsf{last\_tl\_write}_g\, t'\} \mid & \\ & & \quad a \in \mathcal{M}, \bot = \mathsf{last\_tl\_lock}_a\, t'\} & \\ & = & V & \\ L' & = & \{a \mapsto \{L_{t'}[\bar{u}]\} \mid a \in \mathcal{M}, (\bar{u}, \mathsf{lock}(a), \_) = \mathsf{last\_tl\_lock}_a\, t'\} \cup & \\ & & \{a \mapsto \emptyset \mid a \in \mathcal{M}, \bot = \mathsf{last\_tl\_lock}_a\, t'\} & \\ & = & L & \\ \sigma' & = & \{x' \mapsto \{t'(x')\} \mid x' \in \mathcal{X}\} & \\ & & \cup\, \{g \mapsto \{0\} \mid g \in \mathcal{G}, \bot = \mathsf{last\_write}_g\, t'\} & \\ & & \cup\, \{g \mapsto \emptyset \mid g \in \mathcal{G}, \bot = \mathsf{last\_tl\_write}_g\, t', \bot \neq \mathsf{last\_write}_g\, t'\} & \\ & & \cup\, \{g \mapsto \{\sigma_{j-1}\, x'\} \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x', \_) = \mathsf{last\_tl\_write}_g\, t'\} & \\ & = & \sigma & \end{array}$$

Thus $V' = V \sqsubseteq V^\sharp \sqsubseteq V^{\sharp\prime}$, $L' = L \sqsubseteq L^\sharp \sqsubseteq L^{\sharp\prime}$, and $\sigma' = \sigma \subseteq \bar{\gamma} \circ \sigma^\sharp \subseteq \bar{\gamma} \circ \sigma^{\sharp\prime}$. Altogether, $t' \in \eta_{\mathsf{lc}}\,[u', S, A']$ holds for all $t \in \eta^{i-1}\,[u, S, A_0]$ and $t'' \in \bigcup\{\eta^{i-1}\,[s, A_1] \mid A_1 \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1\}$.

As neither constraint causes any side-effects, the statement holds for edges corresponding to calls of $\mathsf{wait}(s)$.

This concludes the case distinction for the inductive step and thus the soundness proof for Lock-Centered Reading with ego-lane digests. □

6.1.3 Protection-Based Reading

To prove the *Protection-Based Reading with Ego-Lane Digests* analysis sound, we show that we can construct from its solution a solution of the constraint system for the *Write-Centered Reading with Ego-Lane Digests* analysis described in Section 4.1.7. Consider some ego-lane digest $\mathcal{A}$ and an appropriate definition of $\mathsf{compat}^\sharp_{\mathcal{A}}$. Let us refer to the constraint system of the analysis from Section 4.1.2 refined with $\mathcal{A}$ by $\mathcal{C}^\sharp_{\mathsf{pb}}$.

To relate the constraint system $\mathcal{C}^\sharp_{\mathsf{pb}}$ to the constraint system for *Write-Centered Reading with Ego-Lane Digests* we introduce some additional side-effects which we will argue are benign for least solutions.

Thus, we first need to establish that the constraint system $\mathcal{C}^\sharp_{\mathsf{pb}}$ has a least solution as observed in Remark 6. As we will later use a fixpoint induction in one of the proof steps, we establish the stronger property that this least solution is in fact attained as the least-upper bound of the Kleene iterates.

We first observe that, the set of all (type-preserving) maps from unknowns used by $\mathcal{C}^\sharp_{\mathsf{pb}}$ to values from their respective abstract domains forms a complete lattice with least element $\bot = \{x \mapsto \bot_x \mid x \in \mathsf{X}\}$, greatest element $\top = \{x \mapsto \top_x \mid x \in \mathsf{X}\}$, and the least upper bound defined point-wise. $\bot_x$ and $\top_x$ here refers to the $\bot$ and $\top$ value for the lattice associated with unknowns of the same type as x, respectively.

Proposition 23. *The right-hand side function of constraint system $\mathcal{C}^\sharp_{pb}$ over the lattice given above is Scott-continuous.*

Proof. We first re-write the right-hand sides and collect all of them in one constraint with the help of a function $\mathsf{flat}_{[x]}(S, V) = S \cup \{[x] \mapsto V\}$ akin to the one introduced in Section 2.2.2 for tuples of concrete values.

As the function $\mathsf{init}(A)^\sharp$ is constant, it also is Scott-continuous. As $\mathsf{flat}_{[x]}$ is Scott-continuous for all $x \in \mathsf{X}$, it thus suffices to check that all $[\![[u, S, A_0,], \mathsf{act}]\!]^\sharp$ respectively $[\![[u, S, A_0,], \mathsf{act}, A']\!]^\sharp$ are Scott-continuous for all possible actions. For create edges, this follows from the Scott-continuity of $\nu^\sharp$, $\pi_{[x]}$ for $[x] \in \mathsf{X}$, and of the map replacement operation $\oplus$. initMT and locking return the unmodified state at their predecessor and are thus Scott-continuous. For assignment to a global variable, Scott-continuity follows from the Scott-continuity of adding an element to a set, the operator $\oplus$ and the access of a value in the map. Similarly, for guards. For unlocking of a mutex, we observe that the unknowns receiving side-effects are fixed. Scott-continuity then follows from the

Scott-continuity of accessing elements in the map. For reading from a global at a given constraint, it is fixed whether the primed or unprimed versions of a value will be read. As the order on P is reversed, the resulting right-hand side is also Scott-continuous. For return, Scott-continuity is a result of the definition of the thread *id* domain — the same holds for join. For signal, the argument is distributed to two different unknowns, which also is Scott-continuous. Lastly, wait returns its argument provided its other argument is non-$\underline{\bot}$ which once more is Scott-continuous.

Thus, the right-hand side is given as the least upper bound of (compositions of) functions that are Scott-continuous, and is thus also Scott-continuous. □

Proposition 24. *The constraint system $\mathcal{C}^\sharp_{pb}$ has a least solution which is obtained as the least-upper bound of all Kleene iterates.*

To simplify the next steps of the proof, we make some adjustments to this constraint system and call the new constraint system ${\mathcal{C}^\sharp}'_{\mathsf{pb}}$. In particular, we modify $[\![[u, S, A_0], \mathsf{unlock}(m_g), A']\!]^\sharp_{\mathsf{pb}}$ and $[\![[u, S, A_0], \mathsf{unlock}(a), A']\!]^\sharp_{\mathsf{pb}}$ by introducing some additional side-effects (highlighted in color below). First, we consider unlocking a mutex m_g for some $g \in \mathcal{G}$.

$$
\begin{aligned}
&[\![[u, S, A_0], \mathsf{unlock}(m_g), A']\!]^{\sharp\prime}_{\mathsf{pb}}\eta = \\
&\quad \textbf{let } (P, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
&\quad \textbf{let } P' = \{h \in P \mid ((S \setminus \{m_g\}) \cap \bar{\mathcal{M}}[h]) \neq \emptyset\} \textbf{ in} \\
&\quad \textbf{let } \rho_0 = \{[g, A']' \mapsto \sigma\, g\} \cup \{[g, A'] \mapsto \sigma\, g \mid \bar{\mathcal{M}}[g] = \{m_g\}\} \textbf{ in} \\
&\quad \textbf{let } \rho_1 = \{[h, A']' \mapsto \sigma\, h \mid h \in \mathcal{G} \setminus \{g\}\} \textbf{ in} \\
&\quad \textbf{let } \rho_2 = \{[h, A'] \mapsto \sigma\, h \mid h \in \mathcal{G}, (\bar{\mathcal{M}}[h] \setminus \{m_h\}) \not\subseteq S \setminus \{m_g\}\} \textbf{ in} \\
&\quad (\rho_0 \cup \rho_1 \cup \rho_2, (P', \sigma))
\end{aligned}
$$

Next, for a mutex $a \notin \{m_g \mid g \in \mathcal{G}\}$:

$$
\begin{aligned}
&[\![[u, S, A_0], \mathsf{unlock}(a), A']\!]^{\sharp\prime}_{\mathsf{pb}}\eta = \\
&\quad \textbf{let } (P, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
&\quad \textbf{let } P' = \{g \in P \mid ((S \setminus \{a\}) \cap \bar{\mathcal{M}}[g]) \neq \emptyset\} \textbf{ in} \\
&\quad \textbf{let } \rho_0 = \{[g, A'] \mapsto \sigma\, g \mid a \in \bar{\mathcal{M}}[g]\} \textbf{ in} \\
&\quad \textbf{let } \rho_1 = \{[g, A']' \mapsto \sigma\, g \mid g \in \mathcal{G}\} \textbf{ in} \\
&\quad \textbf{let } \rho_2 = \{[g, A'] \mapsto \sigma\, g \mid a \notin \bar{\mathcal{M}}[g], (\bar{\mathcal{M}}[g] \setminus \{m_g\}) \not\subseteq S \setminus \{a\}\} \textbf{ in} \\
&\quad (\rho_0 \cup \rho_1 \cup \rho_2, (P', \sigma))
\end{aligned}
$$

The side-effects ρ_1 in both cases ensure to publish the local values of all global variables to the unknown $[g, A']'$. The side-effects ρ_2, on the other hand, publish the values of globals that are no longer totally protected at the end point of the edge, i.e., where at least one of the mutexes $\bar{\mathcal{M}}[g] \setminus \{m_g\}$ is no longer held, to $[g, A']$.

We will argue that for ego-lane digests, these additional side-effects do not change the values that are read for global variables at any program point for least solutions of $\mathcal{C}^\sharp_{\mathsf{pb}}$. To this end, we show that all values that are additionally read from one of the

unknowns that now receive an additional side-effect, are already read from another unknown that also receives a side-effect in the original formulation.

First consider the values contained in $\sigma\, g$. For the least solution of this system, one can write $\sigma\, g$ as a join of all abstract values w_g written to g along some path to the start point of the thread template to which the current program point belongs and the initial value $[\![0]\!]^\sharp_{\mathcal{E}xp}\, \top$.

$$\sigma\, g = ([\![0]\!]^\sharp_{\mathcal{E}xp}\, \top) \sqcup \bigsqcup_{w_g \text{ written to } g} w_g$$

This can be verified by fixpoint induction.

Now, consider a global g and an additional side-effect $\{[g, A']' \mapsto \sigma\, g\}$ from some ρ_1, and a right-hand side for a read from g with associated digest A_0 at which this unknown is consulted. Then, the value read from globals is given by

$$r = \bigsqcup_{A'' \in \mathcal{A},\, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A''} \eta\, [g, A'']'$$

and $\mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'$ holds. We then argue that the contribution from $[g, A']'$ to this join is subsumed by contributions of unknowns to which side-effects were already caused in the original formulation. First, we consider the initial value $([\![0]\!]^\sharp_{\mathcal{E}xp}\, \top)$. As the considered digest is ego-lane-derived, we have by (2.19) that $\mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1$ holds for all $A_1 \in \mathsf{init}^\sharp_{\mathcal{A}}$. Furthermore, the unknowns $[g, A_1]'$ receive an initial side-effect of $([\![0]\!]^\sharp_{\mathcal{E}xp}\, \top)$ in the constraint for initialization. Thus, the initial value is already read from an unknown $[g, A_1]'$. Next, consider some written value w_g: The write of this value was, by construction, immediately followed by an unlock of mutex m_g with associated digest A_1. That constraint causes a side-effect to $[g, A_1]'$. As the digest is ego-lane-derived, we have — by (2.18) — $\mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1$, and the contribution w_g to r is subsumed by contributions of unknowns to which side-effects were already caused in the original formulation. We conclude that the additional side-effects do not change the values read when unknowns of the form $[g, A]'$ are consulted.

Next, for a global g and an additional side-effect $\{[g, A'] \mapsto \sigma\, g\}$ from some ρ_2, and a right-hand side for a read from g with associated digest A_0 at which this unknown is consulted. Then, the value read from globals is given by

$$r = \bigsqcup_{A'' \in \mathcal{A},\, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A''} \eta\, [g, A'']$$

and $\mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'$ holds. We then once more argue that the contribution from $[g, A']$ to this join is subsumed by contributions of unknowns to which side-effects were already caused in the original formulation. For the initial value $([\![0]\!]^\sharp_{\mathcal{E}xp}\, \top)$, the argument is the same as for the side-effects from some ρ_1. Next, consider some written value w_g: By construction, this write happened while holding all mutexes $\bar{\mathcal{M}}[g]$. For each additional side-effect, we know that the global g is not only protected by m_g, is not protected by

the mutex currently being unlocked, and that after the unlock at least one protecting mutex is not held anymore. Since the unlocked mutex was not protecting, and the write happened while all protecting mutexes were held, there must be an unlock of a protecting mutex between the write and the unlock of the current mutex. At this program point, the original formulation causes a side-effect to $[g, A_1]$ for some A_1 that accounts for the written value w_g. As the digest is ego-lane-derived, we have — by (2.18) — $\mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A_1$, and the contribution w_g to r is subsumed by contributions of unknowns to which side-effects were already caused in the original formulation.

We thus have that the additional side-effects do not change the values read from unknowns for global variables and thus obtain:

Proposition 25. *A solution of the modified constraint system ${\mathcal{C}^\sharp}'_{pb}$ can be constructed from the solution of the original constraint system $\mathcal{C}^\sharp_{pb}$ where both coincide on all unknowns except for those corresponding to global variables.*

Proof. By verifying that, after evaluating all right-hand sides once to trigger any additional side-effects, all constraints in ${\mathcal{C}^\sharp}'_{\mathsf{pb}}$ are satisfied and that the mappings still agree on the value of all unknowns except for those corresponding to global variables. □

With this observation, one can now re-write the two modified right-hand sides of the constraint system ${\mathcal{C}^\sharp}'_{\mathsf{pb}}$ to an equivalent but more concise form that gives up the strict distinction between newly added side-effects and those already present in $\mathcal{C}^\sharp_{\mathsf{pb}}$.

$$
\begin{aligned}
&[\![[u, S, A_0], \mathsf{unlock}(m_g), A']\!]^{\sharp\prime}_{\mathsf{pb}}\eta = \\
&\quad \textbf{let } (P, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
&\quad \textbf{let } P' = \{h \in P \mid ((S \setminus \{m_g\}) \cap \bar{\mathcal{M}}[h]) \neq \emptyset\} \textbf{ in} \\
&\quad \textbf{let } \rho_0 = \{[g, A'] \mapsto \sigma\, g \mid \bar{\mathcal{M}}[g] = \{m_g\}\} \textbf{ in} \\
&\quad \textbf{let } \rho_1 = \{[h, A']' \mapsto \sigma\, h \mid h \in \mathcal{G}\} \textbf{ in} \\
&\quad \textbf{let } \rho_2 = \{[h, A'] \mapsto \sigma\, h \mid h \in \mathcal{G}, (\bar{\mathcal{M}}[h] \setminus \{m_h\}) \not\subseteq S \setminus \{m_g\}\} \textbf{ in} \\
&\quad (\rho_0 \cup \rho_1 \cup \rho_2, (P', \sigma))
\end{aligned}
$$

$$
\begin{aligned}
&[\![[u, S, A_0], \mathsf{unlock}(a), A']\!]^{\sharp\prime}_{\mathsf{pb}}\eta = \\
&\quad \textbf{let } (P, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
&\quad \textbf{let } P' = \{g \in P \mid ((S \setminus \{a\}) \cap \bar{\mathcal{M}}[g]) \neq \emptyset\} \textbf{ in} \\
&\quad \textbf{let } \rho_0 = \{[g, A'] \mapsto \sigma\, g \mid g \in \mathcal{G}, (\bar{\mathcal{M}}[g] \setminus \{m_g\}) \not\subseteq S \setminus \{a\}\} \textbf{ in} \\
&\quad \textbf{let } \rho_1 = \{[g, A']' \mapsto \sigma\, g \mid g \in \mathcal{G}\} \textbf{ in} \\
&\quad (\rho_0 \cup \rho_1, (P', \sigma))
\end{aligned}
$$

It thus remains to relate solutions of the modified constraint system ${\mathcal{C}^\sharp}'_{\mathsf{pb}}$ which coincide with the least solution of $\mathcal{C}^\sharp_{\mathsf{pb}}$ on all unknowns except for those corresponding to global variables to solutions of the constraint system $\mathcal{C}^\sharp_{\mathsf{wc}}$ for the *Write-Centered Reading with Ego-Lane Digests* analysis. By abuse of notation, in this section we locally refer to the *modified* constraint system ${\mathcal{C}^\sharp}'_{\mathsf{pb}}$ by $\mathcal{C}^\sharp$ (with right-hand sides referred to by $[\![\cdot]\!]^\sharp_{\mathsf{pb}}$), and

to the constraint system $\mathcal{C}^\sharp_{\mathsf{wc}}$ by $\mathcal{C}$ (with right-hand sides $[\![\cdot]\!]^\sharp_{\mathsf{wc}}$) to avoid obscuring key ideas by heavy notation.

We introduce a concretization mapping γ defined by

$$\begin{array}{lcll} \gamma\,(P^\sharp, \sigma^\sharp) & = & (W, P, \sigma) & \text{where} \\ W & = & \{g \mapsto \{\bar{\mathcal{M}}[g]\} \mid g \in \mathcal{G}\} & \\ P & = & \{g \mapsto \textbf{if}\, g \in P^\sharp\, \textbf{then}\, \{\{a\} \mid a \in \bar{\mathcal{M}}[g]\}\, \textbf{else}\, \{\emptyset\} \mid g \in \mathcal{G}\} & \\ \sigma & = & \sigma^\sharp & \end{array}$$

Moreover, we introduce a *description relation* Δ between the sets of unknowns of $\mathcal{C}$ and those of $\mathcal{C}^\sharp$. It is given by

$$\begin{array}{rcll} [u, S, A] & \Delta & [u, S, A] & \text{for } u \in \mathcal{N} \\ [i, A] & \Delta & [i, A] & \text{for } i \in S_{\mathcal{V}^\sharp_{\mathbf{tid}}} \\ [s, A] & \Delta & [s, A] & \text{for } s \in \mathcal{S} \\ [g, a, S, w, A] & \Delta & [g, A] & \text{for } a \in (\bar{\mathcal{M}}[g] \setminus \{m_g\}), \bar{\mathcal{M}}[g] \subseteq w \\ [g, a, S, w, A] & \Delta & [g, A] & \text{for } a \in ((\mathcal{M} \setminus \bar{\mathcal{M}}[g]) \cup \{m_g\}), \\ & & & \quad (\bar{\mathcal{M}}[g] \setminus \{m_g\}) \not\subseteq S, \bar{\mathcal{M}}[g] \subseteq w \\ [g, a, S, w, A] & \Delta & [g, A]' & \text{for } a \in ((\mathcal{M} \setminus \bar{\mathcal{M}}[g]) \cup \{m_g\}), \\ & & & \quad (\bar{\mathcal{M}}[g] \setminus \{m_g\}) \subseteq S, \bar{\mathcal{M}}[g] \subseteq w \end{array}$$

where additionally $A \in \mathcal{A}$, $g \in \mathcal{G}$, $w \subseteq \mathcal{M}$, and $S \subseteq \mathcal{M}$.

Let $\eta^\sharp$ be the least solution of $\mathcal{C}^\sharp$, which is known to exist as all right-hand sides of $\mathcal{C}^\sharp$ are monotonic, both in the returned values and the side-effects. We construct a mapping η for the constraint system $\mathcal{C}$ from $\eta^\sharp$ by setting $\eta\,[u, S, A] = \gamma\,(\eta^\sharp\,[u, S, A])$, setting $\eta\,[i, A] = \eta^\sharp\,[i, A]$, setting $\eta\,[s, A] = \gamma\,(\eta^\sharp\,[s, A])$, and finally setting

$$\eta\,[g, a, S, w, A] = \begin{cases} \eta^\sharp\,[g, A]' & \text{if } a \in \{m_g\} \cup (\mathcal{M} \setminus \bar{\mathcal{M}}[g]), \\ & \quad (\bar{\mathcal{M}}[g] \setminus \{m_g\}) \subseteq S, \bar{\mathcal{M}}[g] \subseteq w \\ \bot & \text{if } \bar{\mathcal{M}}[g] \not\subseteq w \\ \eta^\sharp\,[g, A] & \text{otherwise} \end{cases}$$

Theorem 19. *Then, we have:*

- *The mapping η, as constructed above, is a solution of the constraint system $\mathcal{C}$.*
- *$\eta^\sharp[g, A] \sqsubseteq \eta^\sharp[g, A]'$ holds for all $g \in \mathcal{G}, A \in \mathcal{A}$.*
- *Whenever $[g, a, S, w, A]\,\Delta\,[g, A]'$, then $\eta\,[g, a, S, w, A] \sqsubseteq \eta^\sharp\,[g, A]'$;*
- *Whenever $[g, a, S, w, A]\,\Delta\,[g, A]$, then $\eta\,[g, a, S, w, A] \sqsubseteq \eta^\sharp\,[g, A]$.*

As, per Theorem 17, solutions of $\mathcal{C}$ are sound w.r.t. the local trace semantics, so are solutions of $\mathcal{C}$ (as they are greater than the *least* solution of $\mathcal{C}$). By Proposition 25, solutions to the constraint system $\mathcal{C}^\sharp_{\mathsf{pb}}$ are then also sound w.r.t. the local trace semantics.

Proof. The proof is by verifying for each edge (u, act, u') of the control-flow graph, appropriate locksets S and S', suitable digests A and A', η constructed above that $[\![[u, S, A], \mathsf{act}]\!]^\sharp_{\mathsf{wc}} \eta \sqsubseteq (\eta, \eta\,[u', S', A'])$ holds.

Consider the constraints corresponding to **unlocking a mutex** a. Consider an edge $(u, \mathsf{unlock}(a), u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, \mathsf{unlock}(a)]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![[u, S, A_0], \mathsf{unlock}(a), A']\!]^\sharp_{\mathsf{wc}}\,\eta \quad \sqsubseteq \quad (\eta, \eta\,[u', S \setminus \{a\}, A'])$$

holds. We first consider **the case where a** $\equiv \mathbf{m_g}$ for some $m_g \in \{m_{g'} \mid g' \in \mathcal{G}\}$. Then,

$$\begin{array}{l}
[\![[u, S, A_0], \mathsf{unlock}(m_g), A']\!]^\sharp_{\mathsf{pb}}\eta^\sharp = \\
\quad \textbf{let } (P^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\
\quad \textbf{let } P^{\sharp\prime\prime} = \{h \in P^\sharp \mid ((S \setminus \{m_g\}) \cap \bar{\mathcal{M}}[h]) \neq \emptyset\} \textbf{ in} \\
\quad \textbf{let } \rho^\sharp_0 = \{[g, A'] \mapsto \sigma^\sharp\, g \mid \bar{\mathcal{M}}[g] = \{m_g\}\} \textbf{ in} \\
\quad \textbf{let } \rho^\sharp_1 = \{[h, A']' \mapsto \sigma^\sharp\, h \mid h \in \mathcal{G}\} \textbf{ in} \\
\quad \textbf{let } \rho^\sharp_2 = \{[h, A'] \mapsto \sigma\sharp\, h \mid h \in \mathcal{G}, (\bar{\mathcal{M}}[h] \setminus \{m_h\}) \not\subseteq S \setminus \{m_g\}\} \textbf{ in} \\
\quad (\rho^\sharp_0 \cup \rho^\sharp_1 \cup \rho^\sharp_2, (P^{\sharp\prime\prime}, \sigma^\sharp)) \\
[\![[u, S, A_0], \mathsf{unlock}(m_g), A']\!]^\sharp_{\mathsf{wc}}\eta = \\
\quad \textbf{let } (W, P, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
\quad \textbf{let } P'' = \{h \mapsto P\,h \sqcup \{S \setminus \{m_g\}\} \mid h \in \mathcal{G}\} \textbf{ in} \\
\quad \textbf{let } \rho = \{[h, m_g, S \setminus \{m_g\}, w, A'] \mapsto \sigma\,h \mid h \in \mathcal{G}, w \in W\,h\} \textbf{ in} \\
\quad (\rho, (W, P'', \sigma))
\end{array}$$

By construction of η, we have

$$\eta\,[u, S, A_0] = (W, P, \sigma) = \gamma(P^\sharp, \sigma^\sharp) = \gamma(\eta^\sharp\,[u, S, A_0])$$

We first show that, in this case,

$$(W, P'', \sigma) \sqsubseteq \eta\,[u', S \setminus \{m_g\}, A'] = (W', P', \sigma') = \gamma(\eta^\sharp\,[u', S \setminus \{m_g\}, A'])$$

holds. Let $\eta^\sharp\,[u', S \setminus \{m_g\}, A'] = (P^{\sharp\prime}, \sigma^{\sharp\prime})$ be the value provided by $\eta^\sharp$ for the endpoint of the edge for the respective digest and lockset. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, we have $P^{\sharp\prime\prime} \sqsubseteq P^{\sharp\prime}$, and $\sigma^\sharp \sqsubseteq \sigma^{\sharp\prime}$. Consequently, we have $\sigma \sqsubseteq \sigma'$ and $W \sqsubseteq W'$. Now consider

$$\begin{array}{ll}
& P'' = \{h \mapsto P\,h \sqcup \{S \setminus \{m_g\}\} \mid h \in \mathcal{G}\} \\
= & \{h \mapsto (\textbf{if } h \in P^\sharp \textbf{ then } \{\{a\} \mid a \in \bar{\mathcal{M}}[h]\} \textbf{ else } \{\emptyset\}) \sqcup \{S \setminus \{m_g\}\} \mid h \in \mathcal{G}\} \\
= & \{h \mapsto (\textbf{if } h \in P^\sharp \textbf{ then } (\{\{a\} \mid a \in \bar{\mathcal{M}}[h]\} \sqcup \{S \setminus \{m_g\}\}) \textbf{ else } (\{\emptyset\})) \mid h \in \mathcal{G}\} \\
\sqsubseteq & \{h \mapsto (\textbf{if } h \in P^{\sharp\prime\prime} \textbf{ then } (\{\{a\} \mid a \in \bar{\mathcal{M}}[h]\} \sqcup \{S \setminus \{m_g\}\}) \textbf{ else } (\{\emptyset\})) \mid h \in \mathcal{G}\} \\
= & \{h \mapsto (\textbf{if } h \in P^{\sharp\prime\prime} \textbf{ then } (\{\{a\} \mid a \in \bar{\mathcal{M}}[h]\}) \textbf{ else } (\{\emptyset\})) \mid h \in \mathcal{G}\} \\
\sqsubseteq & \{h \mapsto (\textbf{if } h \in P^{\sharp\prime} \textbf{ then } (\{\{a\} \mid a \in \bar{\mathcal{M}}[h]\}) \textbf{ else } (\{\emptyset\})) \mid h \in \mathcal{G}\} \\
= & P'
\end{array}$$

where we exploit that for $h \in P^{\sharp\prime\prime}$, by construction $(S \setminus \{m_g\}) \cap \bar{\mathcal{M}}[h] \neq \emptyset$ and thus $\{\{a\} \mid a \in \bar{\mathcal{M}}[h]\} \sqcup \{S \setminus \{m_g\}\} = \{\{a\} \mid a \in \bar{\mathcal{M}}[h]\}$.

We conclude that the return value of $[\![[u, S, A_0], \mathsf{unlock}(m_g), A']\!]^\sharp_{\mathsf{wc}}\eta$ is subsumed by the value $\eta\,[u', S \setminus \{m_g\}, A']$. It remains to consider the side-effects. The side-effects caused by $\mathcal{C}^\sharp$ and by $\mathcal{C}$, are given by

$$\begin{array}{lcl} \rho_0^\sharp & = & \{[g, A'] \mapsto \sigma\, g \mid \bar{\mathcal{M}}[g] = \{m_g\}\} \\ \rho_1^\sharp & = & \{[h, A']' \mapsto \sigma\, h \mid h \in \mathcal{G}\} \\ \rho_2^\sharp & = & \{[h, A'] \mapsto \sigma\, h \mid h \in \mathcal{G}, (\bar{\mathcal{M}}[h] \setminus \{m_h\}) \not\subseteq S \setminus \{m_g\}\} \end{array}$$

and

$$\rho \;=\; \{[h, m_g, S \setminus \{m_g\}, w, A'] \mapsto \sigma\, h \mid h \in \mathcal{G}, w \in W\, h\}$$

respectively. We remark that, by construction of η, we have $W\, h = \{\bar{\mathcal{M}}[h]\}$. The proof obligation then is to show that any side-effected caused by $\mathcal{C}$ is already subsumed by η by construction.

Consider the side-effect in ρ for a global variable h. We distinguish three cases:

- If $m_g \in \{m_h\} \cup (\mathcal{M} \setminus \bar{\mathcal{M}}[h])$, and $\bar{\mathcal{M}}[h] \setminus \{m_h\} \not\subseteq S \setminus \{m_g\}$, then the relationship $[h, m_g, S \setminus \{m_g\}, \bar{\mathcal{M}}[h], A']\Delta[h, A']$ holds, and the side-effect is accounted for by the corresponding side-effect from $\rho_2^\sharp$ to $[h, A']$ in $\mathcal{C}^\sharp$.
- If $m_g \in \{m_h\} \cup (\mathcal{M} \setminus \bar{\mathcal{M}}[h])$, and $\bar{\mathcal{M}}[h] \setminus \{m_h\} \subseteq S \setminus \{m_g\}$, then the relationship $[h, m_g, S \setminus \{m_g\}, \bar{\mathcal{M}}[h], A']\Delta[h, A']'$ holds and the side-effect is accounted for by the corresponding side-effect from $\rho_1^\sharp$ to $[h, A']'$ in $\mathcal{C}^\sharp$.
- $m_g \notin \{m_h\} \cup (\mathcal{M} \setminus \bar{\mathcal{M}}[h])$ implies that $h \neq g$ and $m_g \in \bar{\mathcal{M}}[h]$, which contradicts the construction of $\bar{\mathcal{M}}$ and thus need not be considered.

As we have $\sigma^\sharp = \sigma$, all side-effects from ρ are accounted for, and the claim holds.

Now consider **the case where** $a \in \mathcal{M} \setminus \{m_g \mid g \in \mathcal{G}\}$. Then,

$$\begin{array}{l} [\![[u, S, A_0], \mathsf{unlock}(a), A']\!]^\sharp_{\mathsf{pb}}\eta^\sharp = \\ \quad \textbf{let } (P^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\ \quad \textbf{let } P^{\sharp\prime\prime} = \{g \in P^\sharp \mid ((S \setminus \{a\}) \cap \bar{\mathcal{M}}[g]) \neq \emptyset\} \textbf{ in} \\ \quad \textbf{let } \rho_0^\sharp = \{[g, A'] \mapsto \sigma^\sharp\, g \mid g \in \mathcal{G}(\bar{\mathcal{M}}[g] \setminus \{m_g\}) \not\subseteq S \setminus \{a\}\} \textbf{ in} \\ \quad \textbf{let } \rho_1^\sharp = \{[g, A']' \mapsto \sigma^\sharp\, g \mid g \in \mathcal{G}\} \textbf{ in} \\ \quad (\rho_0^\sharp \cup \rho_1^\sharp, (P^{\sharp\prime\prime}, \sigma^\sharp)) \\ [\![[u, S, A_0], \mathsf{unlock}(a), A']\!]^\sharp_{\mathsf{wc}}\eta = \\ \quad \textbf{let } (W, P, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\ \quad \textbf{let } P'' = \{h \mapsto P\, g \sqcup \{S \setminus \{a\}\} \mid g \in \mathcal{G}\} \textbf{ in} \\ \quad \textbf{let } \rho = \{[g, a, S \setminus \{a\}, w, A'] \mapsto \sigma\, g \mid g \in \mathcal{G}, g \in W\, g\} \textbf{ in} \\ \quad (\rho, (W, P'', \sigma)) \end{array}$$

By construction of η, we have

$$\eta\,[u, S, A_0] = (W, P, \sigma) = \gamma(P^\sharp, \sigma^\sharp) = \gamma(\eta^\sharp\,[u, S, A_0])$$

We first show that, in this case,

$$(W, P'', \sigma) \sqsubseteq \eta\,[u', S \setminus \{a\}, A'] = (W', P', \sigma') = \gamma(\eta^\sharp\,[u', S \setminus \{a\}, A'])$$

holds. Let $\eta^\sharp\,[u', S \setminus \{a\}, A'] = (P^{\sharp\prime}, \sigma^{\sharp\prime})$ be the value provided by $\eta^\sharp$ for the endpoint of the edge for the respective digest and lockset. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, we have $P^{\sharp\prime\prime} \sqsubseteq P^{\sharp\prime}$, and $\sigma^\sharp \sqsubseteq \sigma^{\sharp\prime}$. Consequently, we have $\sigma \sqsubseteq \sigma'$ and $W \sqsubseteq W'$. Now consider

$$\begin{array}{ll}
& P'' = \{g \mapsto P\,g \sqcup \{S \setminus \{a\}\} \mid g \in \mathcal{G}\} \\
= & \{g \mapsto (\textbf{if}\, g \in P^\sharp\, \textbf{then}\, \{\{a'\} \mid a' \in \bar{\mathcal{M}}[g]\}\, \textbf{else}\, \{\emptyset\}) \sqcup \{S \setminus \{a\}\} \mid g \in \mathcal{G}\} \\
= & \{g \mapsto (\textbf{if}\, g \in P^\sharp\, \textbf{then}\, (\{\{a'\} \mid a' \in \bar{\mathcal{M}}[g]\} \sqcup \{S \setminus \{a\}\})\, \textbf{else}\, (\{\emptyset\})) \mid g \in \mathcal{G}\} \\
\sqsubseteq & \{h \mapsto (\textbf{if}\, g \in P^{\sharp\prime\prime}\, \textbf{then}\, (\{\{a'\} \mid a' \in \bar{\mathcal{M}}[g]\} \sqcup \{S \setminus \{a\}\})\, \textbf{else}\, (\{\emptyset\})) \mid g \in \mathcal{G}\} \\
= & \{g \mapsto (\textbf{if}\, g \in P^{\sharp\prime\prime}\, \textbf{then}\, (\{\{a'\} \mid a' \in \bar{\mathcal{M}}[g]\})\, \textbf{else}\, (\{\emptyset\})) \mid g \in \mathcal{G}\} \\
\sqsubseteq & \{g \mapsto (\textbf{if}\, g \in P^{\sharp\prime}\, \textbf{then}\, (\{\{a'\} \mid a' \in \bar{\mathcal{M}}[g]\})\, \textbf{else}\, (\{\emptyset\})) \mid g \in \mathcal{G}\} \\
= & P'
\end{array}$$

where we exploit that for $g \in P^{\sharp\prime\prime}$, by construction $(S \setminus \{a\}) \cap \bar{\mathcal{M}}[g] \neq \emptyset$ and thus $\{\{a'\} \mid a' \in \bar{\mathcal{M}}[g]\} \sqcup \{S \setminus \{a\}\} = \{\{a'\} \mid a' \in \bar{\mathcal{M}}[h]\}$.

We conclude that the return value of $[\![[u, S, A_0], \mathsf{unlock}(m_g), A']\!]^\sharp_{\mathsf{wc}}\eta$ is subsumed by the value $\eta\,[u', S \setminus \{a\}, A']$. It remains to consider the side-effects. The side-effects caused by $\mathcal{C}^\sharp$ and by $\mathcal{C}$, are given by

$$\begin{array}{lcl}
\rho_0^\sharp & = & \{[g, A'] \mapsto \sigma^\sharp\, g \mid g \in \mathcal{G}, (\bar{\mathcal{M}}[g] \setminus \{m_g\}) \not\subseteq S \setminus \{a\}\} \\
\rho_1^\sharp & = & \{[g, A']' \mapsto \sigma^\sharp\, g \mid a \notin \bar{\mathcal{M}}[g]\}
\end{array}$$

and

$$\rho \;=\; \{[g, a, S \setminus \{a\}, w, A'] \mapsto \sigma\, g \mid g \in \mathcal{G}, w \in W\, g\}$$

respectively. We remark that, by construction of η, we have $W\, g = \{\bar{\mathcal{M}}[g]\}$. The proof obligation then is to show that any side-effected caused by $\mathcal{C}$ is already subsumed by η by construction.

Consider the side-effect in ρ for a global variable $g \in \mathcal{G}$. We distinguish three cases:

- $a \in \bar{\mathcal{M}}[g]$: In this case, $(\bar{\mathcal{M}}[g] \setminus \{m_g\}) \not\subseteq S \setminus \{a\}$ holds as well as the relationship $[g, a, S \setminus \{a\}, \bar{\mathcal{M}}[g], A']\Delta[g, A']$. The side-effect is thus accounted for by the corresponding side-effect from $\rho_0^\sharp$ to $[g, A']$ in $\mathcal{C}^\sharp$.

- $a \in \mathcal{M} \setminus \bar{\mathcal{M}}[g]$ and $(\bar{\mathcal{M}}[g] \setminus \{m_g\}) \not\subseteq S \setminus \{a\}$. Then, $[g, a, S \setminus \{a\}, \bar{\mathcal{M}}[g], A']\Delta[g, A']$ holds, and the side-effect is thus accounted for by the corresponding side-effect from $\rho_0^\sharp$ to $[g, A']$ in $\mathcal{C}^\sharp$.

- $a \in \mathcal{M} \setminus \bar{\mathcal{M}}[g]$ and $(\bar{\mathcal{M}}[g] \setminus \{m_g\}) \subseteq S \setminus \{a\}$. Then, $[g, a, S \setminus \{a\}, \bar{\mathcal{M}}[g], A']\Delta[g, A']'$ holds, and the side-effect is thus accounted for by the corresponding side-effect from $\rho_1^\sharp$ to $[g, A']'$ in $\mathcal{C}^\sharp$.

As we have $\sigma^\sharp = \sigma$, all side-effects from ρ are accounted for, and the claim holds.

Next, consider the constraints corresponding to **reading from a global** g. Consider an edge $(u, x = g, u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, x = g]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![[u, S, A_0], x = g]\!]^\sharp_{\mathsf{wc}}\, \eta \quad \sqsubseteq \quad (\eta, \eta\,[u', S, A'])$$

holds. We have

$$\begin{aligned}
&[\![[u, S, A_0], x = g]\!]^\sharp_{\mathsf{pb}} \eta^\sharp = \\
&\quad \textbf{let } (P^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\
&\quad \textbf{let } d^\sharp = \textbf{if } g \in P^\sharp \textbf{ then } \sigma^\sharp\, g \\
&\qquad\quad \textbf{else if } S \cap \bar{\mathcal{M}}[g] = \{m_g\} \textbf{ then} \\
&\qquad\qquad \sigma^\sharp\, g \sqcup \bigsqcup_{A' \in \mathcal{A},\, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'} \eta^\sharp\,[g, A']' \\
&\qquad\quad \textbf{else } \sigma^\sharp\, g \sqcup \bigsqcup_{A' \in \mathcal{A},\, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'} \eta^\sharp\,[g, A'] \\
&\quad \textbf{in} \\
&\quad \textbf{let } \sigma^{\sharp\prime\prime} = \sigma^\sharp \oplus \{x \mapsto d^\sharp\} \textbf{ in} \\
&\quad (\emptyset, (P^\sharp, \sigma^{\sharp\prime\prime}))
\end{aligned}$$

$$\begin{aligned}
&[\![[u, S, A_0], x = g]\!]^\sharp_{\mathsf{wc}} \eta = \\
&\quad \textbf{let } (W, P, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
&\quad \textbf{let } d = \sigma\, g \sqcup \bigsqcup \{\eta[g, a, S', w, A'] \mid a \in S, S \cap S' = \emptyset, \\
&\qquad\quad \exists S'' \in P\, g : S'' \cap w = \emptyset, \\
&\qquad\quad \exists S''' \in P\, g : a \notin S''', \\
&\qquad\quad A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \textbf{ in} \\
&\quad \textbf{let } \sigma'' = \sigma \oplus \{x \mapsto d\} \textbf{ in} \\
&\quad (\emptyset, (W, P, \sigma''))
\end{aligned}$$

By construction of η, we have

$$\eta\,[u, S, A_0] = (W, P, \sigma) = \gamma(P^\sharp, \sigma^\sharp) = \gamma(\eta^\sharp\,[u, S, A_0])$$

We show that, in this case,

$$(W, P, \sigma'') \sqsubseteq \eta\,[u', S, A'] = (W', P', \sigma') = \gamma(\eta^\sharp\,[u', S, A'])$$

holds. Let $\eta^\sharp\,[u', S, A'] = (P^{\sharp\prime}, \sigma^{\sharp\prime})$ be the value provided by $\eta^\sharp$ for the endpoint of the edge for the respective digest and lockset. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, we have $P^\sharp \sqsubseteq P^{\sharp\prime}$, and $\sigma^{\sharp\prime\prime} \sqsubseteq \sigma^{\sharp\prime}$. Consequently, we have $P \sqsubseteq P'$ and (by construction) $W \sqsubseteq W'$. It thus remains to show that $\sigma'' \sqsubseteq \sigma'$ holds. To this end, let us distinguish three cases:

Case 1. $g \in P^\sharp$. Then $\sigma^\sharp \oplus \{g \mapsto \sigma^\sharp\, g\} = \sigma^{\sharp''} \sqsubseteq \sigma^{\sharp'} = \sigma'$, and $P\, g = \{\{a'\} \mid a' \in \bar{\mathcal{M}}[g]\}$.

$$\begin{aligned} d &= \sigma\, g \sqcup \bigsqcup\{\eta[g, a, S', w, A'] \mid a \in S, S \cap S' = \emptyset, \\ &\qquad \exists S'' \in P\, g : S'' \cap w = \emptyset, \\ &\qquad \exists S''' \in P\, g : a \notin S''', \\ &\qquad A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \\ &= \sigma\, g \sqcup \bigsqcup\{\eta[g, a, S', w, A'] \mid a \in S, S \cap S' = \emptyset, \\ &\qquad \exists S'' \in \{\{a'\} \mid a' \in \bar{\mathcal{M}}[g]\} : S'' \cap w = \emptyset, \\ &\qquad \exists S''' \in \{\{a'\} \mid a' \in \bar{\mathcal{M}}[g]\} : a \notin S''', \\ &\qquad A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \\ &= \sigma\, g \sqcup \bot \qquad \text{(by construction of } \eta) \end{aligned}$$

and thus $\sigma'' = \sigma \oplus \{x \mapsto \sigma\, g\} = \sigma^{\sharp''} \sqsubseteq \sigma^{\sharp'} = \sigma'$.

Case 2. $g \notin P^\sharp$, $S \cap \bar{\mathcal{M}}[g] = \{m_g\}$. Then $P\, g = \{\emptyset\}$, and

$$\sigma^\sharp \oplus \left\{ x \mapsto \left(\sigma^\sharp\, g \sqcup \bigsqcup_{A' \in \mathcal{A},\, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'} \eta^\sharp\, [g, A']' \right) \right\} = \sigma^{\sharp''} \sqsubseteq \sigma^{\sharp'} = \sigma'$$

and

$$\begin{aligned} d &= \sigma\, g \sqcup \bigsqcup\{\eta[g, a, S', w, A'] \mid a \in S, S \cap S' = \emptyset, \\ &\qquad \exists S'' \in P\, g : S'' \cap w = \emptyset, \\ &\qquad \exists S''' \in P\, g : a \notin S''', \\ &\qquad A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \\ &= \sigma\, g \sqcup \bigsqcup\{\eta[g, a, S', w, A'] \mid a \in S, S \cap S' = \emptyset, \\ &\qquad A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \\ &\sqsubseteq \sigma\, g \sqcup \bigsqcup\{\eta^\sharp\, [g, A'] \sqcup \eta^\sharp\, [g, A']' \mid \\ &\qquad A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \qquad \text{(by construction of } \eta) \\ &= \sigma\, g \sqcup \bigsqcup\{\eta^\sharp\, [g, A']' \mid A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \qquad (\text{by } \eta^\sharp\, [g, A'] \sqsubseteq \eta^\sharp\, [g, A']') \end{aligned}$$

and thus $\sigma'' = \sigma \oplus \{x \mapsto d\} \sqsubseteq \sigma^{\sharp''} \sqsubseteq \sigma^{\sharp'} = \sigma'$.

Case 3. $g \notin P^\sharp$, $S \cap \bar{\mathcal{M}}[g] \neq \{m_g\}$. Then $P\, g = \{\emptyset\}$, and

$$\sigma^\sharp \oplus \left\{ x \mapsto \left(\sigma^\sharp\, g \sqcup \bigsqcup_{A' \in \mathcal{A},\, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'} \eta^\sharp\, [g, A'] \right) \right\} = \sigma^{\sharp''} \sqsubseteq \sigma^{\sharp'} = \sigma'$$

and

$$\begin{aligned} d &= \sigma\, g \sqcup \bigsqcup\{\eta[g, a, S', w, A'] \mid a \in S, S \cap S' = \emptyset, \\ &\qquad \exists S'' \in P\, g : S'' \cap w = \emptyset, \\ &\qquad \exists S''' \in P\, g : a \notin S''', \\ &\qquad A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \\ &= \sigma\, g \sqcup \bigsqcup\{\eta[g, a, S', w, A'] \mid a \in S, S \cap S' = \emptyset, \\ &\qquad A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \\ &= \sigma\, g \sqcup \bigsqcup\{\eta^\sharp\, [g, A'] \mid A' \in \mathcal{A}, \mathsf{compat}^\sharp_{\mathcal{A}}\, A_0\, A'\} \qquad \text{(by construction of } \eta) \end{aligned}$$

where the last step uses the fact that, for an unknown that received the primed value to be consulted, $\bar{\mathcal{M}}[g] \setminus \{m_g\} \subseteq S'$ would need to hold. However, as by construction $m_g \in S$, here there is some $a \in S \cap (\bar{\mathcal{M}}[g] \setminus \{m_g\})$ and thus for each such S', $S \cap S' \neq \emptyset$. Thus, $\sigma'' = \sigma \oplus \{x \mapsto d\} = \sigma^{\sharp''} \sqsubseteq \sigma^{\sharp'} = \sigma'$.

We conclude that the return value of $[\![[u, S, A_0], x = g]\!]^\sharp_{\mathsf{wc}}\eta$ is subsumed by the value $\eta\,[u', S, A']$. As no side-effects are caused, the claim follows.

Next, consider the constraints corresponding to **writing to a global g**. Consider an edge $(u, g = x, u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, g = x]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![[u, S, A_0], g = x]\!]^\sharp_{\mathsf{wc}}\,\eta \quad \sqsubseteq \quad (\eta, \eta\,[u', S, A'])$$

holds. We have

$$\begin{aligned}
&[\![[u, S, A_0], g = x]\!]^\sharp_{\mathsf{pb}}\eta^\sharp = \\
&\quad \textbf{let } (P^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\
&\quad \textbf{let } P^{\sharp''} = P^\sharp \cup \{g\} \textbf{ in} \\
&\quad \textbf{let } \sigma^{\sharp''} = \sigma^\sharp \oplus \{g \mapsto \sigma^\sharp\, g\} \textbf{ in} \\
&\quad (\emptyset, (P^{\sharp''}, \sigma^{\sharp''}))
\end{aligned}$$

$$\begin{aligned}
&[\![[u, S, A_0], g = x]\!]^\sharp_{\mathsf{wc}}\eta = \\
&\quad \textbf{let } (W, P, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
&\quad \textbf{let } W'' = W \oplus \{g \mapsto \{S\}\} \textbf{ in} \\
&\quad \textbf{let } P'' = P \oplus \{g \mapsto \{S\}\} \textbf{ in} \\
&\quad \textbf{let } \sigma'' = \sigma \oplus \{g \mapsto \sigma\, x\} \textbf{ in} \\
&\quad (\emptyset, (W'', P'', \sigma''))
\end{aligned}$$

By construction of η, we have

$$\eta\,[u, S, A_0] = (W, P, \sigma) = \gamma(P^\sharp, \sigma^\sharp) = \gamma(\eta^\sharp\,[u, S, A_0])$$

We show that, in this case,

$$(W'', P'', \sigma'') \sqsubseteq \eta\,[u', S, A'] = (W', P', \sigma') = \gamma(\eta^\sharp\,[u', S, A'])$$

holds. Let $\eta^\sharp\,[u', S, A'] = (P^{\sharp\prime}, \sigma^{\sharp\prime})$ be the value provided by $\eta^\sharp$ for the endpoint of the edge for the respective digest and lockset. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, we have $P^{\sharp''} \sqsubseteq P^{\sharp\prime}$, and $\sigma^{\sharp''} \sqsubseteq \sigma^{\sharp\prime}$.

By construction $W = W' = \{g \mapsto \{\bar{\mathcal{M}}[g]\} \mid g \in \mathcal{G}\}$. As g is written here, by construction of $\bar{\mathcal{M}}$ all write-protecting mutexes of g are held, i.e., $\bar{\mathcal{M}}[g] \subseteq S$. Then

$$W'' = W \oplus \{g \mapsto \{S\}\} \sqsubseteq W \oplus \{g \mapsto \{\bar{\mathcal{M}}[g]\}\} = W'$$

and furthermore

$$\begin{array}{ll}
& P'' \\
= & P \oplus \{g \mapsto \{S\}\} \\
= & \{h \mapsto (\textbf{if } h \in P^\sharp \textbf{ then } \{\{a\} \mid a \in \bar{\mathcal{M}}[h]\} \textbf{ else } \{\emptyset\}) \mid h \in \mathcal{G}\} \oplus \{g \mapsto \{S\}\} \\
\sqsubseteq & \{h \mapsto (\textbf{if } h \in P^\sharp \textbf{ then } \{\{a\} \mid a \in \bar{\mathcal{M}}[h]\} \textbf{ else } \{\emptyset\}) \mid h \in \mathcal{G}\} \oplus \{g \mapsto \{\bar{\mathcal{M}}[g]\}\} \\
\sqsubseteq & \{h \mapsto (\textbf{if } h \in P^\sharp \textbf{ then } \{\{a\} \mid a \in \bar{\mathcal{M}}[h]\} \textbf{ else } \{\emptyset\}) \mid h \in \mathcal{G}\} \oplus \\
& \quad \{g \mapsto \{\{a\} \mid a \in \bar{\mathcal{M}}[g]\}\} \\
= & \{h \mapsto (\textbf{if } h \in (P^\sharp \cup g) \textbf{ then } \{\{a\} \mid a \in \bar{\mathcal{M}}[h]\} \textbf{ else } \{\emptyset\}) \mid h \in \mathcal{G}\} \\
= & \{h \mapsto (\textbf{if } h \in P^{\sharp''} \textbf{ then } \{\{a\} \mid a \in \bar{\mathcal{M}}[h]\} \textbf{ else } \{\emptyset\}) \mid h \in \mathcal{G}\} \\
\sqsubseteq & \{h \mapsto (\textbf{if } h \in P^{\sharp'} \textbf{ then } \{\{a\} \mid a \in \bar{\mathcal{M}}[h]\} \textbf{ else } \{\emptyset\}) \mid h \in \mathcal{G}\} \\
= & P'
\end{array}$$

Lastly, $\sigma'' = \sigma \oplus \{g \mapsto \sigma\, x\} = \sigma^\sharp \oplus \{g \mapsto \sigma^\sharp\, x\} = \sigma^{\sharp''} \sqsubseteq \sigma^{\sharp'} = \sigma'$, and conclude that the return value of $[\![[u, S, A_0], g = x]\!]^\sharp_{\mathsf{wc}}\eta$ is subsumed by the value $\eta\,[u', S, A']$. As no side-effects are caused, the claim follows. The proof for constraints corresponding to **assignments to local variables as well as for guards** is analogous (with the difference that P is not modified), and thus is omitted here.

Next, consider the constraints corresponding to **locking a mutex** a. Consider an edge $(u, \mathsf{lock}(a), u') \in \mathcal{E}$ and digests A', A_0, A_1 such that $A' \in [\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}(A_0, A_1)$. We verify that

$$[\![[u, S, A_0], \mathsf{lock}(a), A']\!]^\sharp_{\mathsf{wc}}\,\eta \quad \sqsubseteq \quad (\eta, \eta\,[u', S \cup \{a\}, A'])$$

holds. We have

$$[\![[u, S, A_0], \mathsf{lock}(a)]\!]^\sharp_{\mathsf{pb}}\eta^\sharp = (\emptyset, \eta^\sharp\,[u, S, A_0])$$
$$[\![[u, S, A_0], \mathsf{lock}(a)]\!]^\sharp_{\mathsf{wc}}\eta = (\emptyset, \eta\,[u, S, A_0])$$

By construction of η, we have

$$\eta\,[u, S, A_0] = \gamma(\eta^\sharp\,[u, S, A_0])$$

Let $\eta^\sharp\,[u', S \cup \{a\}, A'] = (P^{\sharp'}, \sigma^{\sharp'})$ be the value provided by $\eta^\sharp$ for the endpoint of the edge for the respective digest and lockset. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, we have $\eta^\sharp\,[u, S, A_0] \sqsubseteq \eta^\sharp\,[u', S \cup \{a\}, A']$, and thus

$$\eta\,[u, S, A_0] = \gamma(\eta^\sharp\,[u, S, A_0]) \sqsubseteq \gamma(\eta^\sharp\,[u', S \cup \{a\}, A']) = \eta\,[u', S \cup \{a\}, A']$$

We conclude that the return value of $[\![[u, S, A_0], \mathsf{lock}(a)]\!]^\sharp_{\mathsf{wc}}\eta$ is subsumed by the value $\eta\,[u', S \cup \{a\}, A']$. As no side-effects are caused, the claim follows.

Next, consider the constraints corresponding to **thread creation**. Consider an edge $(u, x = \mathsf{create}(u_1), u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, x = \mathsf{create}(u_1)]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![[u, S, A_0], x = \mathsf{create}(u_1)]\!]^\sharp_{\mathsf{wc}}\,\eta \quad \sqsubseteq \quad (\eta, \eta\,[u', S, A'])$$

holds. We have

$$
\begin{aligned}
&[\![[u, S, A_0], x = \mathsf{create}(u_1)]\!]^\sharp_{\mathsf{pb}} \eta^\sharp = \\
&\quad \textbf{let } (P^\sharp, \sigma^\sharp) = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\
&\quad \textbf{let } i^\sharp = \nu^\sharp\, u\, \sigma^\sharp\, u_1 \textbf{ in} \\
&\quad \textbf{let } \sigma^\sharp_\rho = \sigma^\sharp \oplus (\{\mathsf{self} \mapsto i^\sharp\} \cup \{g \mapsto [\![0]\!]^\sharp_{\mathcal{E}xp}\, \top \mid g \in \mathcal{G}\}) \textbf{ in} \\
&\quad \textbf{let } \sigma^{\sharp\prime\prime} = \sigma^\sharp \oplus \{x \mapsto i^\sharp\} \textbf{ in} \\
&\quad \textbf{let } \rho^\sharp = \{[u_1, \emptyset, A'] \mapsto (\emptyset, \sigma^\sharp_\rho) \mid A' \in \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0\} \textbf{ in} \\
&\quad (\rho^\sharp, (P^\sharp, \sigma^{\sharp\prime\prime})) \\
&[\![[u, S, A_0], x = \mathsf{create}(u_1)]\!]^\sharp_{\mathsf{wc}} \eta = \\
&\quad \textbf{let } (W, P, \sigma) = \eta\,[u, S, A_0] \textbf{ in} \\
&\quad \textbf{let } W_\rho = \{g \mapsto \emptyset \mid g \in \mathcal{G}\} \textbf{ in} \\
&\quad \textbf{let } P_\rho = \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}\} \textbf{ in} \\
&\quad \textbf{let } i = \nu^\sharp\, u\, \sigma\, u_1 \textbf{ in} \\
&\quad \textbf{let } \sigma_\rho = \sigma \oplus \left(\{\mathsf{self} \mapsto i\} \cup \left\{g \mapsto \left(\sigma\, g \sqcap [\![0]\!]^\sharp_{\mathcal{E}xp}\, \top\right) \mid g \in \mathcal{G}\right\}\right) \textbf{ in} \\
&\quad \textbf{let } \sigma'' = \sigma \oplus \{x \mapsto i\} \textbf{ in} \\
&\quad \textbf{let } \rho = \{[u_1, \emptyset, A'] \mapsto (W_\rho, P_\rho, \sigma_\rho) \mid A' \in \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0\} \textbf{ in} \\
&\quad (\rho, (W, P, \sigma''))
\end{aligned}
$$

By construction of η, we have

$$\eta\,[u, S, A_0] = (W, P, \sigma) = \gamma(P^\sharp, \sigma^\sharp) = \gamma(\eta^\sharp\,[u, S, A_0])$$

We first show that, in this case,

$$(W, P, \sigma'') \sqsubseteq \eta\,[u', S, A'] = (W', P', \sigma') = \gamma(\eta^\sharp\,[u', S, A'])$$

holds. Let $\eta^\sharp\,[u', S, A'] = (P^{\sharp\prime}, \sigma^{\sharp\prime})$ be the value provided by $\eta^\sharp$ for the endpoint of the edge for the respective digest and lockset. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, we have $P^\sharp \sqsubseteq P^{\sharp\prime}$, and $\sigma^{\sharp\prime\prime} \sqsubseteq \sigma^{\sharp\prime}$. Consequently, we have $P \sqsubseteq P'$ and (by construction) $W \sqsubseteq W'$. It thus remains to show that $\sigma'' \sqsubseteq \sigma'$ holds. Here, we have

$$\sigma'' = \sigma \oplus \{x \mapsto \nu^\sharp\, u\, \sigma\, u_1\} = \sigma^\sharp \oplus \{x \mapsto \nu^\sharp\, u\, \sigma^\sharp\, u_1\} = \sigma^{\sharp\prime\prime} \sqsubseteq \sigma^{\sharp\prime} = \sigma'$$

We conclude that the return value of $[\![[u, S, A_0], x = \mathsf{create}(u_1)]\!]^\sharp_{\mathsf{wc}} \eta$ is subsumed by the value $\eta\,[u', S, A']$.

It remains to consider the side-effects. As the side-effect occurs to the same unknown in both cases, it is only the value being side-effected that needs to be considered. Consider the value $\eta^\sharp\,[u_1, \emptyset, A'] = (P^{\sharp\prime}_\rho, \sigma^{\sharp\prime}_\rho)$ provided by $\eta^\sharp$ for the target unknown of the side-effect. As $\eta^\sharp$ is a solution, we have $\emptyset \sqsubseteq P^{\sharp\prime}_\rho$ and $\sigma^\sharp_\rho \sqsubseteq \sigma^{\sharp\prime}_\rho$. We now show that

$$(W_\rho, P_\rho, \sigma_\rho) \sqsubseteq \eta\,[u_1, \emptyset, A'] = (W'_\rho, P'_\rho, \sigma'_\rho) = \gamma(\eta^\sharp\,[u_1, \emptyset, A'])$$

We have

$$\begin{aligned}
\sigma_\rho &= \sigma \oplus \left(\{\mathsf{self} \mapsto \nu^\sharp\, u\, \sigma\, u_1\} \cup \left\{g \mapsto \left(\sigma\, g \sqcap [\![0]\!]^\sharp_{\mathcal{E}xp}\, \top\right) \mid g \in \mathcal{G}\right\}\right) \\
&= \sigma^\sharp \oplus \left(\{\mathsf{self} \mapsto \nu^\sharp\, u\, \sigma^\sharp\, u_1\} \cup \left\{g \mapsto \left(\sigma^\sharp\, g \sqcap [\![0]\!]^\sharp_{\mathcal{E}xp}\, \top\right) \mid g \in \mathcal{G}\right\}\right) \\
&\sqsubseteq \sigma^\sharp \oplus (\{\mathsf{self} \mapsto i^\sharp\} \cup \{g \mapsto [\![0]\!]^\sharp_{\mathcal{E}xp}\, \top \mid g \in \mathcal{G}\}) \\
&= \sigma^\sharp_\rho \\
&\sqsubseteq \sigma^{\sharp\prime}_\rho = \sigma'_\rho
\end{aligned}$$

and $W_\rho = \{g \mapsto \emptyset \mid g \in \mathcal{G}\} \sqsubseteq \{g \mapsto \{\bar{\mathcal{M}}[g]\} \mid g \in \mathcal{G}\} = W'_\rho$ as well as

$$\begin{aligned}
P_\rho &= \{g \mapsto \{\emptyset\} \mid g \in \mathcal{G}\} \\
&= \{g \mapsto (\textbf{if}\, g \in \emptyset\, \textbf{then}\, \{\{a\} \mid a \in \bar{\mathcal{M}}[g]\}\, \textbf{else}\, \{\emptyset\}) \mid g \in \mathcal{G}\} \\
&\sqsubseteq \{g \mapsto (\textbf{if}\, g \in P^{\sharp\prime}\, \textbf{then}\, \{\{a\} \mid a \in \bar{\mathcal{M}}[g]\}\, \textbf{else}\, \{\emptyset\}) \mid g \in \mathcal{G}\} \\
&= P'_\rho
\end{aligned}$$

We conclude that all side-effects from ρ are thus accounted for in $\eta\,[u_1, \emptyset, A']$, and the claim holds.

For the remaining right-hand sides (corresponding to **return**, **initMT**, **initialization**, **wait**, **signal**, and **join**), the proof continues similarly. As the constraints in $\mathcal{C}^\sharp$ and $\mathcal{C}$ take an essentially identical form for these right-hand sides, the details for these cases are omitted here. □

6.2 Soundness Proofs for Analyses Considering Clusters of Globals

This section provides soundness proofs for the analyses presented in Section 4.2. While the constraint systems of these analyses are considerably closer to the constraint system for the concrete semantics than the constraint systems considered in Section 6.1 were, the unknowns still do not quite coincide. Thus, the overall strategy is the same as in the previous section: First a version of the concrete constraint system with unknowns very close to the abstract system is constructed, and then a relationship between solutions of the abstract system and solutions of the concrete system is established.

6.2.1 Mutex-Meet with Digests

Let the constraint system from Section 4.2.2 instantiated with some digest $\mathcal{A}$ be called $\mathcal{C}^\sharp$. We consider the refined constraint system for the concrete semantics (Eq. (2.12)) instantiated with the considered actions from Section 2.4 and the product digest of $\mathcal{A}$ and the lockset digest from Fig. 2.11 and refer to this constraint system by $\mathcal{C}$. Recall that the resulting constraint system has the following set of unknowns:

- $[u, S, A]$ for $u \in \mathcal{N}$, $S \subseteq \mathcal{M}$ and $A \in \mathcal{A}$,

- $[a, S, A]$ for $g \in \mathcal{G}, a \in \mathcal{M}, S \subseteq \mathcal{M}, w \subseteq \mathcal{M}$ and $A \in \mathcal{A}$,
- $[\mathsf{initMT}, \emptyset, A]$ for $A \in \mathcal{A}$,
- $[\mathsf{return}, S, A]$ for $S \subseteq \mathcal{M}$ and $A \in \mathcal{A}$, and
- $[s, S, A]$ for $s \in \mathcal{S}$, $S \subseteq \mathcal{M}$ and $A \in \mathcal{A}$.

where we once again abbreviate $\mathsf{unlock}(a)$ by a and $\mathsf{signal}(s)$ by s. While there is quite a close correspondence between the unknowns used by the analysis and the unknowns of the concrete constraint system, they do not match up exactly. We thus, as a first step, construct a constraint system $\mathcal{C}_{\mathsf{mm}}$ over the set of local traces for which the set of unknowns matches those of $\mathcal{C}^\sharp$ - up to the unknowns used for thread returns. $\mathcal{C}_{\mathsf{mm}}$ thus uses the following set of unknowns:

- $[u, S, A]$ for $u \in \mathcal{N}$, $S \subseteq \mathcal{M}$ and $A \in \mathcal{A}$,
- $[a, Q, A]$ for $a \in \mathcal{M}, Q \in \mathcal{Q}_a$ and $A \in \mathcal{A}$,
- $[i, A]$ for $i \in \mathcal{V}_{\mathbf{tid}}$ and $A \in \mathcal{A}$, and
- $[s, A]$ for $s \in \mathcal{S}$, and $A \in \mathcal{A}$.

We remark that for each $a \in \mathcal{M}$ there is at least one corresponding unknown as we demand that $\mathcal{Q}_a \neq \emptyset$. The constraints for $\mathcal{C}_{\mathsf{mm}}$ are then given by:

$$[u_0, \emptyset, A] \quad \supseteq \quad \mathbf{fun}\ \_ \to (\emptyset, \{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\}) \quad \text{for } A \in \mathsf{init}^\sharp_{\mathcal{A}}$$

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], x = \mathsf{create}(u_1), u')]\!]_{\mathsf{mm}} \quad \text{for } (u, x = \mathsf{create}(u_1), u') \in \mathcal{E}, A' \in [\![u, x = \mathsf{create}(u_1)]\!]^\sharp_{\mathcal{A}}(A_0)$$

$$[u', S \cup \{a\}, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{lock}(a), u'), A_1]\!]_{\mathsf{mm}} \quad \text{for } (u, \mathsf{lock}(a), u') \in \mathcal{E}, A' \in [\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}(A_0, A_1)$$

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{act}, u'), A_1]\!]_{\mathsf{mm}} \quad \text{for } (u, \mathsf{act}, u') \in \mathcal{E}, \mathsf{act} \in \mathcal{A}ct_{\mathsf{observing}}, \text{not lock}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0, A_1)$$

$$[u', S \setminus \{a\}, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{unlock}(a), u'), A']\!]_{\mathsf{mm}} \quad \text{for } (u, \mathsf{unlock}(a), u') \in \mathcal{E}, A' \in [\![u, \mathsf{unlock}(a)]\!]^\sharp_{\mathcal{A}}(A_0)$$

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{act}, u'), A']\!]_{\mathsf{mm}} \quad \text{for } (u, \mathsf{act}, u') \in \mathcal{E}, \mathsf{act} \in \mathcal{A}ct_{\mathsf{observable}}, \text{not unlock or initMT}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0)$$

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{act}, u')]\!]_{\mathsf{mm}} \quad \text{for } (u, \mathsf{act}, u') \in \mathcal{E}, \mathsf{act} \in \mathcal{A}ct_{\mathsf{local}}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0)$$

$$[u', \emptyset, A'] \quad \supseteq \quad [\![([u, \emptyset, A_0], \mathsf{initMT}, u'), A']\!]_{\mathsf{mm}} \quad \text{for } (u, \mathsf{initMT}, u') \in \mathcal{E}, A' \in [\![u, \mathsf{initMT}]\!]^\sharp_{\mathcal{A}}(A_0)$$

The corresponding right-hand sides $[\![\cdot]\!]_{\mathsf{mm}}$ are then given by:

$$
\begin{aligned}
&[\![([u,S,A_0],x=\mathsf{create}(u_1),u')]\!]_{\mathsf{mm}}\,\eta_{\mathsf{mm}} = \\
&\quad \textbf{let } T = [\![(u,x=\mathsf{create}(u_1),u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u,S,A_0]) \textbf{ in} \\
&\quad (\{[u_1,\emptyset,\mathsf{new}^\sharp_{\mathcal{A}}\,u\,u_1\,A_0] \mapsto \mathsf{new}\,u_1\,(\eta_{\mathsf{mm}}\,[u,S,A_0])\},T)
\end{aligned}
$$

$$
\begin{aligned}
&[\![([u,S,A_0],\mathsf{lock}(a),u'),A_1]\!]_{\mathsf{mm}}\,\eta_{\mathsf{mm}} = \\
&\quad \textbf{let } T_1 = \bigcap\{\eta_{\mathsf{mm}}\,[a,Q,A_1] \mid Q \in \mathcal{Q}_a\} \textbf{ in} \\
&\quad \textbf{let } T = [\![(u,\mathsf{lock}(a),u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u,S,A_0],T_1) \textbf{ in} \\
&\quad (\emptyset,T)
\end{aligned}
$$

$$
\begin{aligned}
&[\![([u,S,A_0],\mathsf{unlock}(a),u'),A']\!]_{\mathsf{mm}}\,\eta_{\mathsf{mm}} = \\
&\quad \textbf{let } T = [\![(u,\mathsf{unlock}(a),u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u,S,A_0]) \textbf{ in} \\
&\quad \textbf{let } \rho = \{[a,Q,A'] \mapsto T \mid Q \in \mathcal{Q}_a\} \textbf{ in} \\
&\quad (\rho,T)
\end{aligned}
$$

$$
\begin{aligned}
&[\![([u,S,A_0],\mathsf{return},u'),A']\!]_{\mathsf{mm}}\,\eta_{\mathsf{mm}} = \\
&\quad \textbf{let } T = [\![(u,\mathsf{return},u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u,S,A_0]) \textbf{ in} \\
&\quad \textbf{let } \rho = \{[i,A'] \mapsto \{t \mid t \in T, \mathsf{id}\,t = i\} \mid i \in \mathcal{V}_{\mathbf{tid}}\} \textbf{ in} \\
&\quad (\rho,T)
\end{aligned}
$$

$$
\begin{aligned}
&[\![([u,S,A_0],x=\mathsf{join}(x'),u'),A_1]\!]_{\mathsf{mm}}\,\eta_{\mathsf{mm}} = \\
&\quad \textbf{let } T_1 = \bigcup\{\eta_{\mathsf{mm}}\,[t(x'),A_1] \mid t \in \eta_{\mathsf{mm}}\,[u,S,A_0]\} \textbf{ in} \\
&\quad \textbf{let } T = [\![(u,x=\mathsf{join}(x'),u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u,S,A_0],T_1) \textbf{ in} \\
&\quad (\emptyset,T)
\end{aligned}
$$

$$
\begin{aligned}
&[\![([u,S,A_0],\mathsf{signal}(s),u'),A']\!]_{\mathsf{mm}}\,\eta_{\mathsf{mm}} = \\
&\quad \textbf{let } T = [\![(u,\mathsf{signal}(s),u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u,S,A_0]) \textbf{ in} \\
&\quad \textbf{let } \rho = \{[s,A'] \mapsto T\} \textbf{ in} \\
&\quad (\rho,T)
\end{aligned}
$$

$$
\begin{aligned}
&[\![([u,S,A_0],\mathsf{wait}(s),u'),A_1]\!]_{\mathsf{mm}}\,\eta_{\mathsf{mm}} = \\
&\quad \textbf{let } T = [\![(u,\mathsf{wait}(s),u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u,S,A_0],\eta_{\mathsf{mm}}\,[s,A_1]) \textbf{ in} \\
&\quad (\emptyset,T)
\end{aligned}
$$

$$
\begin{aligned}
&[\![([u,S,A_0],\mathsf{initMT},u'),A']\!]_{\mathsf{mm}}\,\eta_{\mathsf{mm}} = \\
&\quad \textbf{let } T = [\![(u,\mathsf{act},u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u,S,A_0]) \textbf{ in} \\
&\quad \textbf{let } \rho = \{[a,Q,A'] \mapsto T \mid a \in \mathcal{M}, Q \in \mathcal{Q}_a\} \textbf{ in} \\
&\quad (\rho,T)
\end{aligned}
$$

$$
\begin{aligned}
&[\![([u,S,A_0],\mathsf{act},u')]\!]_{\mathsf{mm}}\,\eta_{\mathsf{mm}} = \\
&\quad \textbf{let } T = [\![(u,\mathsf{act},u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u,S,A_0]) \textbf{ in} \\
&\quad (\emptyset,T)
\end{aligned}
$$

where act refers to a *local* action.

Proposition 26. *The right-hand side function of constraint system $\mathcal{C}_{mm}$ over the lattice mapping (extended) unknowns to sets of local traces with the order as discussed in Section 2.2.2 is Scott-continuous.*

Proof. The proof here proceeds in the same manner as the proof of Proposition 10 in Section 2.3. After collecting all right-hand sides into one constraint using the Scott-continuous helper function $\mathsf{flat}_{[x]}$, it remains to show that the individual right-hand sides are composition of Scott-continuous functions. This follows from the Scott-continuity of $[\![\cdot]\!]_{\mathcal{T}}$, new, and $\pi_{[x]}$ for $[x] \in \mathsf{X}$. Furthermore, all arguments to $[\![\cdot]\!]_{\mathcal{T}}$ are constructed in a way where either the union of sets is taken, or the resulting set is constructed as the union of some function applied to each element of the set in isolation, also rendering these functions Scott-continuous. Then, the combined right-hand side is given as the least upper bound of (compositions of) functions that are Scott-continuous, and is thus also Scott-continuous. □

Proposition 27. *The constraint system $\mathcal{C}_{mm}$ has a least solution which is obtained as the least-upper bound of all Kleene iterates.*

To be able to relate solutions of $\mathcal{C}$ and solutions of $\mathcal{C}_{\mathsf{mm}}$ we define for a mapping η from unknowns of $\mathcal{C}$ to sets of local traces a mapping $\mathsf{split}[\eta]$ from unknowns of $\mathcal{C}_{\mathsf{mm}}$ to sets of local traces as follows:

$$\begin{aligned}
\mathsf{split}[\eta][u, S, A] &= \eta[u, S, A] && (\text{for } u \in \mathcal{N}, S \subseteq \mathcal{M}, A \in \mathcal{A}) \\
\mathsf{split}[\eta][a, Q, A] &= \eta[\mathsf{initMT}, \emptyset, A] \cup \textstyle\bigcup_{S \subseteq \mathcal{M}} \eta[a, S, A] && (\text{for } a \in \mathcal{M}, Q \in \mathcal{Q}_a, A \in \mathcal{A}) \\
\mathsf{split}[\eta][i, A] &= \{t \in \textstyle\bigcup_{S \subseteq \mathcal{M}} \eta[\mathsf{return}, S, A] \mid \mathsf{id}\, t = i\} && (\text{for } i \in \mathcal{V}_{\mathbf{tid}}, A \in \mathcal{A}) \\
\mathsf{split}[\eta][s, A] &= \textstyle\bigcup_{S \subseteq \mathcal{M}} \eta[s, S, A] && (\text{for } s \in \mathcal{S}, S \subseteq \mathcal{M}, A \in \mathcal{A})
\end{aligned}$$

We then obtain:

Proposition 28. *The following two statements are equivalent:*

(A) η is the least solution of $\mathcal{C}$.

(B) $\mathsf{split}[\eta]$ is the least solution of $\mathcal{C}_{mm}$.

Proof. We prove both statements simultaneously by fixpoint induction: Consider the $i-$th approximation η^i to the least solution of $\mathcal{C}$, and the $i-$th approximation η^i_{mm} to the least solution of $\mathcal{C}_{\mathsf{mm}}$. Let us call property (1) that $\eta^i_{\mathsf{mm}} = \mathsf{split}[\eta^i]$. For $i = 0$, the value of all unknowns in both constraint systems is $\emptyset$, and property (1) holds trivially. Next, we show that for constraints corresponding to a control-flow edge as well as the constraint for initialization executed in lock-step, provided that property (1) holds before the update, it still holds after the update. Considering the constraint's contribution to the unknown on the left and its side-effects (if any are triggered) suffices for this.

First, consider the constraints for **initialization**. They take identical form in both constraint systems:

$$[u_0, \emptyset, A] \supseteq \mathbf{fun}\ \_ \to (\emptyset, \{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\}) \qquad \text{for } A \in \mathsf{init}^\sharp_{\mathcal{A}}$$

In both systems, unknowns $[u_0, \emptyset, A]$ receive identical contributions. Thus, if property (1) holds for the i-th approximations, and constraints of this form are considered, it also holds for the $(i+1)$-th approximations.

Next, consider the constraints for **initMT**. Consider an edge $(u, \mathsf{initMT}, u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, \mathsf{initMT}]\!]^\sharp_{\mathcal{A}}(A_0)$. We remark that, by construction, the lockset is empty when executing initMT. For $\mathcal{C}_{\mathsf{mm}}$, the constraints take the following form:

$$[u', \emptyset, A'] \quad \supseteq \quad [\![([u, \emptyset, A_0], \mathsf{initMT}, u'), A']\!]_{\mathsf{mm}}$$

with right-hand side

$$\begin{array}{l}
[\![([u, \emptyset, A_0], \mathsf{initMT}, u'), A']\!]_{\mathsf{mm}}\, \eta_{\mathsf{mm}} = \\
\quad \textbf{let } T = [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\, [u, \emptyset, A_0]) \textbf{ in} \\
\quad \textbf{let } \rho = \{[a, Q, A'] \mapsto T \mid a \in \mathcal{M}, Q \in \mathcal{Q}_a\} \textbf{ in} \\
\quad (\rho, T)
\end{array}$$

For $\mathcal{C}$, the constraints take the following form:

$$[u', \emptyset, A'] \quad \supseteq \quad [\![([u, \emptyset, A_0], \mathsf{initMT}, [u', \emptyset, A'])]\!]$$

with right-hand side

$$\begin{array}{rcl}
[\![([u, \emptyset, A_0], \mathsf{initMT}, [u', \emptyset, A'])]\!]\, \eta & = & \textbf{let } T = [\![(u, \mathsf{initMT}, u')]\!]_{\mathcal{T}}(\eta\, [u, \emptyset, A_0]) \textbf{ in} \\
& & (\{[\mathsf{initMT}, \emptyset, A'] \mapsto T\}, T)
\end{array}$$

Provided property (1) holds for the i-th approximations, the unknowns $[u', \emptyset, A']$ of both constraint systems receive the same new contribution. For the side-effects, $\eta^{i+1}\,[\mathsf{initMT}, \emptyset, A']$ receives the additional contribution T, which is also the new contribution to all $\eta^{i+1}_{\mathsf{mm}}\,[a, Q, A']$. Consider now some $a \in \mathcal{M}$ and $Q \in \mathcal{Q}_a$:

$$\begin{array}{rcl}
\eta^{i+1}_{\mathsf{mm}}\,[a, Q, A'] & = & \eta^{i}_{\mathsf{mm}}\,[a, Q, A'] \cup T \\
& = & \mathsf{split}[\eta^i]\,[a, Q, A'] \cup T \\
& = & \eta^i[\mathsf{initMT}, \emptyset, A] \cup (\bigcup_{S \subseteq \mathcal{M}} \eta^i[a, S, A]) \cup T \\
& = & (\eta^i[\mathsf{initMT}, \emptyset, A] \cup T) \cup (\bigcup_{S \subseteq \mathcal{M}} \eta^{i+1}[a, S, A]) \\
& = & \eta^{i+1}[\mathsf{initMT}, \emptyset, A] \cup (\bigcup_{S \subseteq \mathcal{M}} \eta^{i+1}[a, S, A]) \\
& = & \mathsf{split}[\eta^{i+1}]\,[a, Q, A']
\end{array}$$

Thus, if property (1) holds for the i-th approximations, and constraints of this form are considered, it also holds for the $(i+1)$-th approximations.

Next, consider the constraints for **local** actions. Here, the same reasoning applies as for the constraints for initMT as discussed above and neither constraint causes any side-effect. Thus, if property (1) holds for the i-th approximations, and constraints corresponding to local actions are considered, it also holds for the $(i+1)$-th approximations. For constraints corresponding to **thread creation** the constraints and right-hand

sides of both constraint systems coincide. Provided property (1) holds for the i-th approximations, the unknowns on the left-hand sides of both constraint systems receive the same new contribution, as do the unknowns that receive side-effects. All of these unknowns are associated with program points. Thus, if property (1) holds for the i-th approximations, and constraints of this form are considered, it also holds for the $(i+1)$-th approximations.

Next, consider the constraints corresponding to **lock**. Consider an edge $(u, \mathsf{lock}(a), u') \in \mathcal{E}$ and digests A', A_0, where $A' \in [\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}(A_0, A_1)$. For $\mathcal{C}_{\mathsf{mm}}$, the constraints take the following form:

$$[u', S \cup \{a\}, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{lock}(a), u'), A_1]\!]_{\mathsf{mm}}$$

with right-hand side

$$\begin{array}{l}
[\![([u, S, A_0], \mathsf{lock}(a), u'), A_1]\!]_{\mathsf{mm}}\, \eta_{\mathsf{mm}} = \\
\quad \textbf{let } T_0 = \bigcap\{\eta_{\mathsf{mm}}\,[a, Q, A_1] \mid Q \in \mathcal{Q}_a\} \textbf{ in} \\
\quad \textbf{let } T = [\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u, S, A_0], T_0) \textbf{ in} \\
\quad (\emptyset, T)
\end{array}$$

For $\mathcal{C}$, the constraint takes the following form

$$[u', S \cup \{a\}, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{lock}(a), u'), A_1]\!]$$

with right-hand side

$$\begin{array}{l}
[\![([u, S, A_0], \mathsf{lock}(a), u'), A_1]\!]\, \eta = \\
\quad \textbf{let } T_1 = [\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}\,(\eta\,[u, S, A_0], \eta\,[\mathsf{initMT}, \emptyset, A_1]) \textbf{ in} \\
\quad \textbf{let } T_2 = \bigcup_{S' \subseteq \mathcal{M}} [\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}\,(\eta\,[u, S, A_0], \eta\,[a, S', A_1]) \textbf{ in} \\
\quad (\emptyset, T_1 \cup T_2)
\end{array}$$

As a first step, we relate T and $T_1 \cup T_2$ to each other.

$$\begin{array}{rcl}
T_1 \cup T_2 & = & [\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}\,(\eta^i\,[u, S, A_0], \eta^i\,[\mathsf{initMT}, \emptyset, A_1]) \\
& & \bigcup_{S' \subseteq \mathcal{M}} [\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}\,(\eta^i\,[u, S, A_0], \eta^i\,[a, S', A_1]) \\
& = & [\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}(\eta^i\,[u, S, A_0], \eta^i\,[\mathsf{initMT}, \emptyset, A_1] \cup \bigcup_{S' \subseteq \mathcal{M}} \eta^i\,[a, S', A_1])
\end{array}$$

By induction hypothesis, we have that $\eta^i_{\mathsf{mm}}\,[u, S, A_0] = \mathsf{split}[\eta^i]\,[u, S, A_0] = \eta^i\,[u, S, A_0]$ and it thus remains to relate the second arguments of $[\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}$ to each other.

$$\begin{array}{rcl}
\bigcap\{\eta^i_{\mathsf{mm}}\,[a, Q, A_1] \mid Q \in \mathcal{Q}_a\} & = & \bigcap\{\mathsf{split}[\eta^i]\,[a, Q, A_1] \mid Q \in \mathcal{Q}_a\} \\
& = & \bigcap\{\eta^i[\mathsf{initMT}, \emptyset, A_1] \cup \bigcup_{S' \subseteq \mathcal{M}} \eta^i[a, S', A_1] \mid Q \in \mathcal{Q}_a\} \\
& = & \eta^i[\mathsf{initMT}, \emptyset, A_1] \cup \bigcup_{S' \subseteq \mathcal{M}} \eta^i[a, S', A_1]
\end{array}$$

As a consequence, we obtain $T = T_1 \cup T_2$, and

$$\begin{array}{rcl}
\eta^{i+1}_{\mathsf{mm}}\,[u', S \cup \{a\}, A'] & = & \eta^{i}_{\mathsf{mm}}\,[u', S \cup \{a\}, A'] \cup T \\
& = & \mathsf{split}[\eta^i]\,[u', S \cup \{a\}, A'] \cup T \\
& = & \eta^i[u', S \cup \{a\}, A'] \cup (T_1 \cup T_2) \\
& = & \eta^{i+1}[u', S \cup \{a\}, A'] \\
& = & \mathsf{split}[\eta^{i+1}]\,[u', S \cup \{a\}, A']
\end{array}$$

As neither constraint causes any side-effect, we obtain: If property (1) holds for the i-th approximations, and constraints corresponding to locking a mutex are considered, it also holds for the $(i+1)$-th approximations.

Next, for constraints corresponding to **unlock**. Consider an edge $(u, \mathsf{unlock}(a), u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, \mathsf{unlock}(a)]\!]^\sharp_{\mathcal{A}}(A_0)$. For $\mathcal{C}_{\mathsf{mm}}$, the constraints take the following form:

$$[u', S \setminus \{a\}, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{unlock}(a), u'), A']\!]_{\mathsf{mm}}$$

with right-hand side

$$\begin{array}{l}
[\![([u, S, A_0], \mathsf{unlock}(a), u'), A']\!]_{\mathsf{mm}}\,\eta_{\mathsf{mm}} = \\
\quad \textbf{let } T = [\![(u, \mathsf{unlock}(a), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u, S, A_0]) \textbf{ in} \\
\quad \textbf{let } \rho = \{[a, Q, A'] \mapsto T \mid Q \in \mathcal{Q}_a\} \textbf{ in} \\
\quad (\rho, T)
\end{array}$$

For $\mathcal{C}$, the constraints take the following form:

$$[u', S \setminus \{a\}, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{unlock}(a), [u', S \setminus \{a\}, A'])]\!]$$

with right-hand side

$$\begin{array}{l}
[\![([u, S, A_0], \mathsf{unlock}(a), [u', S \setminus \{a\}, A'])]\!]\,\eta = \\
\quad \textbf{let } T = [\![(u, \mathsf{unlock}(a), u')]\!]_{\mathcal{T}}(\eta\,[u, S, A_0]) \textbf{ in} \\
\quad (\{[a, S \setminus \{a\}, A'] \mapsto T\}, T)
\end{array}$$

Provided property (1) holds for the i-th approximations, the unknowns $[u', S \setminus \{a\}, A']$ of both constraint systems receive the same new contribution. It thus remains to consider the side-effects. Consider some cluster $Q \in \mathcal{Q}_a$:

$$\begin{array}{rl}
& \eta^{i+1}_{\mathsf{mm}}\,[a, Q, A'] = \eta^{i}_{\mathsf{mm}}\,[a, Q, A'] \cup T = \mathsf{split}[\eta^i]\,[a, Q, A'] \cup T \\
= & \eta^i[\mathsf{initMT}, \emptyset, A'] \cup (\bigcup_{S' \subseteq \mathcal{M}} \eta^i[a, S', A']) \cup T \\
= & \eta^{i+1}[\mathsf{initMT}, \emptyset, A'] \cup (\bigcup_{S' \subseteq \mathcal{M}} \eta^i[a, S', A']) \cup T \\
= & \eta^{i+1}[\mathsf{initMT}, \emptyset, A'] \cup (\bigcup_{S' \subseteq \mathcal{M}, S' \neq S \setminus \{a\}} \eta^i[a, S', A']) \cup (\eta^i[a, S \setminus \{a\}, A'] \cup T) \\
= & \eta^{i+1}[\mathsf{initMT}, \emptyset, A'] \cup (\bigcup_{S' \subseteq \mathcal{M}, S' \neq S \setminus \{a\}} \eta^{i+1}[a, S', A']) \cup \eta^{i+1}[a, S \setminus \{a\}, A'] \\
= & \eta^{i+1}[\mathsf{initMT}, \emptyset, A'] \cup (\bigcup_{S' \subseteq \mathcal{M}} \eta^{i+1}[a, S', A']) \\
= & \mathsf{split}[\eta^{i+1}]\,[a, Q, A']
\end{array}$$

Thus, if property (1) holds for the i-th approximations, and constraints of this form are considered, it also holds for the $(i+1)$-th approximations.

Next, consider the constraints corresponding to **return**. Consider an edge $(u, \mathsf{return}, u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, \mathsf{return}]\!]^\sharp_{\mathcal{A}}(A_0)$. For $\mathcal{C}_{\mathsf{mm}}$, the constraints take the following form:

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{return}, u'), A']\!]_{\mathsf{mm}}$$

with right-hand side

$$\begin{array}{l}
[\![([u, S, A_0], \mathsf{return}, u'), A']\!]_{\mathsf{mm}}\, \eta_{\mathsf{mm}} = \\
\quad \textbf{let } T = [\![(u, \mathsf{return}, u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\, [u, S, A_0]) \textbf{ in} \\
\quad \textbf{let } \rho = \{[i, A'] \mapsto \{t \mid t \in T, \mathsf{id}\, t = i\} \mid i \in \mathcal{V}_{\mathbf{tid}}\} \textbf{ in} \\
\quad (\rho, T)
\end{array}$$

For $\mathcal{C}$, the constraints take the following form:

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{return}, [u', S, A'])]\!]$$

with right-hand side

$$\begin{array}{lcl}
[\![([u, S, A_0], \mathsf{return}, [u', S, A'])]\!]\, \eta & = & \textbf{let } T = [\![(u, \mathsf{return}, u')]\!]_{\mathcal{T}}(\eta\, [u, S, A_0]) \textbf{ in} \\
& & (\{[\mathsf{return}, S, A'] \mapsto T\}, T)
\end{array}$$

Provided property (1) holds for the i-th approximations, the unknowns $[u', S, A']$ of both constraint systems receive the same new contribution. For the side-effects, consider some $j \in \mathcal{V}_{\mathbf{tid}}$. Then

$$\begin{array}{ll}
& \eta^{i+1}_{\mathsf{mm}}\, [j, A'] = \eta^{i}_{\mathsf{mm}}\, [j, A'] \cup \{t \mid t \in T, \mathsf{id}\, t = j\} \\
= & \mathsf{split}[\eta^i]\, [j, A'] \cup \{t \mid t \in T, \mathsf{id}\, t = j\} \\
= & \{t \in \bigcup_{S' \subseteq \mathcal{M}} \eta^i[\mathsf{return}, S', A'] \mid \mathsf{id}\, t = j\} \cup \{t \mid t \in T, \mathsf{id}\, t = j\} \\
= & \{t \in \bigcup_{S' \subseteq \mathcal{M}, S' \neq S} \eta^i[\mathsf{return}, S', A'] \mid \mathsf{id}\, t = j\} \\
& \cup \{t \in \eta^i[\mathsf{return}, S, A'] \mid \mathsf{id}\, t = j\} \cup \{t \mid t \in T, \mathsf{id}\, t = j\} \\
= & \{t \in \bigcup_{S' \subseteq \mathcal{M}, S' \neq S} \eta^{i+1}[\mathsf{return}, S', A'] \mid \mathsf{id}\, t = j\} \\
& \cup \{t \in (\eta^i[\mathsf{return}, S, A'] \cup T) \mid \mathsf{id}\, t = j\} \\
= & \{t \in \bigcup_{S' \subseteq \mathcal{M}, S' \neq S} \eta^{i+1}[\mathsf{return}, S', A'] \mid \mathsf{id}\, t = j\} \cup \{t \in \eta^{i+1}[\mathsf{return}, S, A'] \mid \mathsf{id}\, t = j\} \\
= & \{t \in \bigcup_{S' \subseteq \mathcal{M}} \eta^{i+1}[\mathsf{return}, S', A'] \mid \mathsf{id}\, t = j\} \\
= & \mathsf{split}[\eta^{i+1}]\, [j, A']
\end{array}$$

Thus, if property (1) holds for the i-th approximations, and constraints of this form are considered, it also holds for the $(i+1)$-th approximations.

Next, consider the constraints corresponding to **join**. Consider an edge $(u, x = \mathsf{join}(x'), u') \in \mathcal{E}$ and digests $A', A_0, A' \in [\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}(A_0, A_1)$. For $\mathcal{C}_{\mathsf{mm}}$, the constraints take the following form:

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], x = \mathsf{join}(x'), u'), A_1]\!]_{\mathsf{mm}}$$

with right-hand side

$$\begin{array}{l}[\![([u,S,A_0],x=\mathsf{join}(x'),u'),A_1]\!]_{\mathsf{mm}}\,\eta_{\mathsf{mm}} = \\ \quad \textbf{let } T_1 = \bigcup\{\eta_{\mathsf{mm}}\,[t(x'),A_1] \mid t \in \eta_{\mathsf{mm}}\,[u,S,A_0]\} \textbf{ in} \\ \quad \textbf{let } T = [\![(u,x=\mathsf{join}(x'),u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u,S,A_0],T_1) \textbf{ in} \\ \quad (\emptyset,T)\end{array}$$

For $\mathcal{C}$, the constraints take the following form

$$[u',S,A'] \quad \supseteq \quad [\![([u,S,A_0],x=\mathsf{join}(x'),u'),A_1]\!]$$

with right-hand side

$$\begin{array}{l}[\![([u,S,A_0],x=\mathsf{join}(x'),u'),A_1]\!]\,\eta = \\ \quad \textbf{let } T = \bigcup_{S'\subseteq\mathcal{M}}[\![(u,x=\mathsf{join}(x'),u')]\!]_{\mathcal{T}}\,(\eta\,[u,S,A_0],\eta\,[\mathsf{return},S',A_1]) \textbf{ in} \\ \quad (\emptyset,T)\end{array}$$

By the argument outlined for the join case in the proof of Proposition 16, we have

$$\begin{array}{rl} & [\![(u,x=\mathsf{join}(x'),u')]\!]_{\mathcal{T}}(\eta^i_{\mathsf{mm}}\,[u,S,A_0],\bigcup\{\eta^i_{\mathsf{mm}}\,[t(x'),A_1] \mid t \in \eta^i_{\mathsf{mm}}\,[u,S,A_0]\}) \\ = & [\![(u,x=\mathsf{join}(x'),u')]\!]_{\mathcal{T}}(\eta^i_{\mathsf{mm}}\,[u,S,A_0],\bigcup_{j\in\mathcal{V}_{\mathbf{tid}}}\eta^i_{\mathsf{mm}}\,[j,A_1])\end{array}$$

We also have:

$$\begin{array}{rl} & \bigcup_{S'\subseteq\mathcal{M}}[\![(u,x=\mathsf{join}(x'),u')]\!]_{\mathcal{T}}\,\left(\eta^i\,[u,S,A_0],\eta^i\,[\mathsf{return},S',A_1]\right) \\ = & [\![(u,x=\mathsf{join}(x'),u')]\!]_{\mathcal{T}}\,\left(\eta^i\,[u,S,A_0],\bigcup_{S'\subseteq\mathcal{M}}\eta^i\,[\mathsf{return},S',A_1]\right)\end{array}$$

exploiting that $[\![\cdot]\!]_{\mathcal{T}}$ is defined point-wise. By induction hypothesis, we have that $\eta^i_{\mathsf{mm}}\,[u,S,A_0] = \mathsf{split}[\eta^i]\,[u,S,A_0] = \eta^i\,[u,S,A_0]$. It thus remains to relate the second arguments of $[\![(u,x=\mathsf{join}(x'),u')]\!]_{\mathcal{T}}$ as given above to each other.

$$\begin{array}{rl} & \bigcup_{j\in\mathcal{V}_{\mathbf{tid}}}\eta^i_{\mathsf{mm}}\,[j,A_1] = \bigcup_{j\in\mathcal{V}_{\mathbf{tid}}}\mathsf{split}[\eta^i]\,[j,A_1] \\ = & \bigcup_{j\in\mathcal{V}_{\mathbf{tid}}}\left(\{t\in\bigcup_{S'\subseteq\mathcal{M}}\eta^i[\mathsf{return},S',A_1] \mid \mathsf{id}\,t = j\}\right) \\ = & \bigcup_{S'\subseteq\mathcal{M}}\eta^i[\mathsf{return},S',A_1]\end{array}$$

Thus, $[\![\cdot]\!]_{\mathcal{T}}$ returns the same set of local traces in both cases. As neither constraint causes any side-effect, we obtain: If property (1) holds for the i-th approximations, and constraints corresponding to joining a thread are considered, it also holds for the $(i+1)$-th approximations.

Next, for constraints corresponding to **signal**. Consider an edge $(u,\mathsf{signal}(s),u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u,\mathsf{signal}(s)]\!]^\sharp_{\mathcal{A}}(A_0)$. For $\mathcal{C}_{\mathsf{mm}}$, the constraints take the following form:

$$[u',S,A'] \quad \supseteq \quad [\![([u,S,A_0],\mathsf{signal}(s),u'),A']\!]_{\mathsf{mm}}$$

with right-hand side

$$\begin{array}{l}[\![([u, S, A_0], \mathsf{signal}(s), u'), A']\!]_{\mathsf{mm}}\, \eta_{\mathsf{mm}} = \\ \quad \textbf{let } T = [\![(u, \mathsf{signal}(s), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\, [u, S, A_0]) \textbf{ in} \\ \quad \textbf{let } \rho = \{[s, A'] \mapsto T\} \textbf{ in} \\ \quad (\emptyset, T)\end{array}$$

For $\mathcal{C}$, the constraints take the following form:

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{signal}(s), [u', S, A'])]\!]$$

with right-hand side

$$\begin{array}{lll}[\![([u, S, A_0], \mathsf{signal}(s), [u', S, A'])]\!]\, \eta & = & \textbf{let } T = [\![(u, \mathsf{signal}(s), u')]\!]_{\mathcal{T}}(\eta\, [u, S, A_0]) \textbf{ in} \\ & & (\{[s, S, A'] \mapsto T\}, T)\end{array}$$

Provided property (1) holds for the i-th approximations, the unknowns $[u', S, A']$ of both constraint systems receive the same new contribution. It thus remains to consider the side-effects.

$$\begin{array}{ll} & \eta^{i+1}_{\mathsf{mm}}\, [s, A'] = \eta^{i}_{\mathsf{mm}}\, [s, A'] \cup T = \mathsf{split}[\eta^i]\, [s, A'] \cup T \\ = & (\bigcup_{S' \subseteq \mathcal{M}} \eta^i [s, S', A']) \cup T \\ = & (\bigcup_{S' \subseteq \mathcal{M}, S' \neq S} \eta^i [s, S', A']) \cup (\eta^i [s, S, A'] \cup T) \\ = & (\bigcup_{S' \subseteq \mathcal{M}, S' \neq S} \eta^{i+1} [s, S', A']) \cup \eta^{i+1} [s, S, A'] \\ = & (\bigcup_{S' \subseteq \mathcal{M}} \eta^{i+1} [s, S', A']) \\ = & \mathsf{split}[\eta^{i+1}]\, [s, A']\end{array}$$

Thus, if property (1) holds for the i-th approximations, and constraints of this form are considered, it also holds for the $(i+1)$-th approximations.

Lastly, consider the constraints corresponding to **wait**. Consider an edge $(u, \mathsf{wait}(s), u') \in \mathcal{E}$ and digests A', A_0, where $A' \in [\![u, \mathsf{wait}(s)]\!]^\sharp_{\mathcal{A}}(A_0, A_1)$. For $\mathcal{C}_{\mathsf{mm}}$, the constraints take the following form:

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{wait}(s), u'), A_1]\!]_{\mathsf{mm}}$$

with right-hand side

$$\begin{array}{l}[\![([u, S, A_0], \mathsf{wait}(s), u'), A_1]\!]_{\mathsf{mm}}\, \eta_{\mathsf{mm}} = \\ \quad \textbf{let } T = [\![(u, \mathsf{wait}(s), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\, [u, S, A_0], \eta_{\mathsf{mm}}\, [s, A_1]) \textbf{ in} \\ \quad (\emptyset, T)\end{array}$$

For $\mathcal{C}$, the constraints take the following form

$$[u', S, A'] \quad \supseteq \quad [\![([u, S, A_0], \mathsf{wait}(s), u'), A_1]\!]$$

with right-hand side

$$\begin{array}{l}[\![([u,S,A_0],\mathsf{wait}(s),u'),A_1]\!]\,\eta = \\ \quad \textbf{let } T_0 = \bigcup_{S'\subseteq\mathcal{M}} [\![(u,\mathsf{wait}(s),u')]\!]_{\mathcal{T}}\,(\eta\,[u,S,A_0],\eta\,[s,S',A_1]) \textbf{ in} \\ \quad (\emptyset,T_0)\end{array}$$

As $[\![\cdot]\!]_{\mathcal{T}}$ is defined point-wise, we have

$$\begin{array}{rl} & \bigcup_{S'\subseteq\mathcal{M}} [\![(u,\mathsf{wait}(s),u')]\!]_{\mathcal{T}}\,(\eta^i\,[u,S,A_0],\eta^i\,[s,S',A_1]) \\ = & [\![(u,\mathsf{wait}(s),u')]\!]_{\mathcal{T}}\,(\eta^i\,[u,S,A_0],(\bigcup_{S'\subseteq\mathcal{M}}\eta^i\,[s,S',A_1]))\end{array}$$

By induction hypothesis, we have that $\eta^i_{\mathsf{mm}}\,[u,S,A_0] = \mathsf{split}[\eta^i]\,[u,S,A_0] = \eta^i\,[u,S,A_0]$ and it thus remains to relate the second arguments of $[\![(u,\mathsf{wait}(s),u')]\!]_{\mathcal{T}}$ to each other.

$$\eta^i_{\mathsf{mm}}\,[s,A_1] = \mathsf{split}[\eta^i]\,[s,A_1] = \bigcup_{S'\subseteq\mathcal{M}}\eta^i[s,S',A]$$

Thus, $[\![\cdot]\!]_{\mathcal{T}}$ returns the same set of local traces in both cases. As neither constraint causes any side-effect, we obtain: If property (1) holds for the i-th approximations, and constraints corresponding to wait are considered, it also holds for the $(i+1)$-th approximations.

This concludes the case distinction and the proof of Proposition 28. □

It thus remains to relate solutions of $\mathcal{C}^\sharp$ and solutions of $\mathcal{C}_{\mathsf{mm}}$ to each other. The approach is similar to the proofs in the preceding subsection: We first define a function β that extracts some information from a local trace. In this instance, it has the type $\mathcal{T}\to(\mathcal{V}ars\to\mathcal{V})$ and extracts a map that contains the values of the locals at the sink of t as well as the last written values of globals.

$$\begin{array}{rcl}\beta\,t & = & \{x\mapsto t(x)\mid x\in\mathcal{X}\}\cup\{g\mapsto 0\mid g\in\mathcal{G},\bot=\mathsf{last\_write}_g\,t\} \\ & & \cup\,\{g\mapsto\sigma_{j-1}\,x\mid g\in\mathcal{G},((j-1,u_{j-1},\sigma_{j-1}),g=x,\_)=\mathsf{last\_write}_g\,t\}\end{array}$$

Remark 27. *Note that the type of β differs from the one of the β used for the proofs of the other analyses. This is for convenience reasons, as this proof works with a relational abstract domain, whereas the other proofs assume a non-relational abstract domain.*

The abstraction function β is used to specify concretization functions for the values of unknowns $[u,S,A]$ for program points, currently held locksets, and digests as well as for the other unknowns

$$\begin{array}{rcl}\gamma_{u,S,A}(r) & = & \{t\in\mathcal{T}\mid \mathsf{loc}\,t=u,L_t=S,\alpha_{\mathcal{A}}\,t=A,\beta\,t\in\gamma_{\mathcal{R}}\,r\} \\ \gamma_{a,A}(r) & = & \{t\in\mathcal{T}\mid \mathsf{last}\,t=\mathsf{unlock}(a)\vee\mathsf{last}\,t=\mathsf{initMT},\alpha_{\mathcal{A}}\,t=A,\beta\,t\in\gamma_{\mathcal{R}}\,r\} \\ \gamma_{i,A}(r) & = & \{t\in\mathcal{T}\mid \mathsf{last}\,t=\mathsf{return},\alpha_{\mathcal{A}}\,t=A,\mathsf{id}\,t=i,\beta\,t\in\gamma_{\mathcal{R}}\,r\} \\ \gamma_{s,A}(r) & = & \{t\in\mathcal{T}\mid \mathsf{last}\,t=\mathsf{signal}(s),\alpha_{\mathcal{A}}\,t=A,\beta\,t\in\gamma_{\mathcal{R}}\,r\}\end{array}$$

We remark that these concretization functions are monotonic. For a solution $\eta^\sharp$ of $\mathcal{C}^\sharp$, we then construct a mapping η'_{mm} by:

$$\begin{array}{lcll}
\eta'_{\mathsf{mm}}[u,S,A] & = & \gamma_{u,S,A}(\eta^\sharp\,[u,S,A]) & u \in \mathcal{N}, S \subseteq \mathcal{M}, A \in \mathcal{A} \\
\eta'_{\mathsf{mm}}[a,Q,A] & = & \gamma_{a,A}(\eta^\sharp\,[a,Q,A]) & a \in \mathcal{M}, Q \in \mathcal{Q}_a, A \in \mathcal{A} \\
\eta'_{\mathsf{mm}}[i,A] & = & \bigcup\{\gamma_{i,A}(\eta^\sharp\,[i^\sharp,A]) \mid i^\sharp \in S_{\mathcal{V}^\sharp_{\mathbf{tid}}}, i \in \gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}\{i^\sharp\}\} & i \in \mathcal{V}_{\mathbf{tid}}, A \in \mathcal{A} \\
\eta'_{\mathsf{mm}}[s,A] & = & \gamma_{s,A}(\eta^\sharp\,[s,A]) & s \in \mathcal{S}, A \in \mathcal{A}
\end{array}$$

Altogether, correctness of $\mathcal{C}^\sharp$ follows from the following theorem:

Theorem 20. *Every solution of $\mathcal{C}^\sharp$ is sound w.r.t. the local trace semantics.*

Proof. Recall from Proposition 28, that the least solution of $\mathcal{C}_{\mathsf{mm}}$ is sound w.r.t. the local trace semantics as specified by the constraint system $\mathcal{C}$. It thus suffices to prove that the mapping η'_{mm} as constructed above, is a solution of the constraint system $\mathcal{C}_{\mathsf{mm}}$. For that, we verify by fixpoint induction that for the i-th approximation η^i to the least solution η of $\mathcal{C}_{\mathsf{mm}}$, $\eta^i \subseteq \eta'_{\mathsf{mm}}$ holds.

To establish soundness for the constraints for locking a mutex where a meet is performed (an in fact for all observing actions), it is necessary to show that the abstract value associated with a program point does not contain information about unprotected globals and that each unknown for a mutex and cluster only stores information about globals that are part of this cluster. Intuitively, one would like to state that $\eta^\sharp\,[a,Q,A] = (\eta^\sharp\,[a,Q,A])\big|_Q$. However, this does not hold for arbitrary solutions, and we have not shown the existence of a least solution for $\mathcal{C}^\sharp$ and do not want to demand its existence to make the soundness argument potentially also applicable in cases where such a least solution does not exist. Thus, we consider an alternative property, namely that not only $\beta\,t$ is in the concretization of the relational abstract value, but so are all other variable assignments where all variables not protected receive a non-deterministic value.

We define an operation similar to restrict from mappings from variables to concrete values that maps one mapping to the set of all mappings obtained when replacing all the values of variables which are forgotten with some non-deterministic value (as always assumed to be type-preserving):

$$\sigma\big|_X = \{\sigma \oplus \{x_i \mapsto v_i, \ldots x_m \mapsto v_m\} \mid x_i \in (\mathcal{V}ars \setminus X), v_i \in \mathcal{V}\}$$

We call auxiliary property (a) that

$$\begin{array}{ll}
\forall t \in \eta^i\,[u,S,A] : (\beta\,t)\big|_{\mathcal{X}\cup\{g\in\mathcal{G},\bar{\mathcal{M}}[g]\cap S\neq\emptyset\}} \subseteq \gamma_{\mathcal{R}}\,(\eta^\sharp\,[u,S,A]) & u \in \mathcal{N}, S \subseteq \mathcal{M}, A \in \mathcal{A} \\
\forall t \in \eta^i\,[a,Q,A] : (\beta\,t)\big|_Q \subseteq \gamma_{\mathcal{R}}\,(\eta^\sharp\,[a,Q,A]) & a \in \mathcal{M}, Q \in \mathcal{Q}_a, A \in \mathcal{A}
\end{array}$$

holds. The reason for this definition via β instead of directly embedding this in the concretization is that, for some of these mappings, there is no corresponding local trace attaining this configuration, which would complicate the further arguments. For the zero-th iteration η^i is $\emptyset$ everywhere, and thus auxiliary property (a) hold.

We first consider the constraints for **initialization**, the start point u_0 and the empty lockset. We verify that for all $A \in \mathsf{init}^\sharp_{\mathcal{A}}$:

$$(\emptyset, \{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\}) \quad \subseteq \quad (\eta'_{\mathsf{mm}}, \eta'_{\mathsf{mm}}\,[u_0, \emptyset, A])$$

As no side-effects are triggered, it suffices to check that $\{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\} \subseteq \eta'_{\mathsf{mm}}\,[u_0, \emptyset, A]$ holds.

$$\mathsf{init}(A)^\sharp\ \_ \quad = \quad (\emptyset, [\![\mathsf{self} \leftarrow^\sharp [\![i_0]\!]^\sharp_{\mathcal{E}xp}\ \top]\!]^\sharp_{\mathcal{R}}\top)$$

Let $\eta^\sharp\,[u_0, \emptyset, A] = r^\sharp$ the value provided by $\eta^\sharp$ for the start point and the empty lockset. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $[\![\mathsf{self} \leftarrow^\sharp [\![i_0]\!]^\sharp_{\mathcal{E}xp}\ \top]\!]^\sharp_{\mathcal{R}}\top \sqsubseteq r^\sharp$ holds. By definition,

$$\eta'_{\mathsf{mm}}\,[u_0, \emptyset, A] = \gamma_{u_0,\emptyset,A}(r^\sharp) \quad = \quad \{t \in \mathcal{T} \mid \mathsf{loc}\,t = u_0, L_t = \emptyset, \alpha_{\mathcal{A}}\,t = A, \beta\,t \in \gamma_{\mathcal{R}}\,r^\sharp\}$$

Consider a trace t from $\{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\}$. Then, $\mathsf{loc}\,t = u_0$, $L_t = \emptyset$, and $\alpha_{\mathcal{A}}\,t = A$ all hold vacuously. Let $\sigma = \beta\,t$. We remark that, by construction, $\sigma\,\mathsf{self} = i_0$. Then, by soundness of the operations on the base domain and the relational domain, we obtain

$$\sigma \in \sigma|_{\mathcal{X}} \subseteq \gamma_{\mathcal{R}}\left([\![\mathsf{self} \leftarrow^\sharp [\![i_0]\!]^\sharp_{\mathcal{E}xp}\ \top]\!]^\sharp_{\mathcal{R}}\top\right) \subseteq \gamma_{\mathcal{R}}(r^\sharp)$$

Altogether $t \in \eta'_{\mathsf{mm}}\,[u_0, \emptyset, A]$ holds for all $t \in \{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\}$ and the auxiliary property (a) holds for the next approximation as well.

Next, consider the constraints for **initMT**. Consider an edge $(u, \mathsf{initMT}, u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, \mathsf{initMT}]\!]^\sharp_{\mathcal{A}}(A_0)$. We remark that, by construction, the lockset is empty when executing initMT. We verify that

$$[\![([u, \emptyset, A_0], \mathsf{initMT}, u')]\!]_{\mathsf{mm}}\,\eta^i \quad \subseteq \quad (\eta'_{\mathsf{mm}}, \eta'_{\mathsf{mm}}\,[u', \emptyset, A'])$$

We have

$$\begin{array}{rcl} [\![([u, \emptyset, A_0], \mathsf{initMT}, u'), A']\!]_{\mathsf{mm}}\,\eta & = & \textbf{let}\ T = [\![(u, \mathsf{act}, u')]\!]_{\mathcal{T}}(\eta\,[u, \emptyset, A_0])\ \textbf{in} \\ & & \textbf{let}\ \rho = \{[a, Q, A'] \mapsto T \mid a \in \mathcal{M}, Q \in \mathcal{Q}_a\}\ \textbf{in} \\ & & (\rho, T) \end{array}$$

$$\begin{array}{rcl} [\![[u, \emptyset, A_0], \mathsf{initMT}, A']\!]^\sharp \eta^\sharp & = & \textbf{let}\ r(Q) = [\![\{g \leftarrow 0 \mid g \in Q\}]\!]^\sharp_{\mathcal{R}}\top\ \textbf{in} \\ & & \textbf{let}\ \rho^\sharp = \{[a, Q, A'] \mapsto r(Q) \mid a \in \mathcal{M}, Q \in \mathcal{Q}_a\}\ \textbf{in} \\ & & (\rho, \eta^\sharp\,[u, \emptyset, A_0]) \end{array}$$

Let $\eta^\sharp\,[u, \emptyset, A_0] = r^\sharp$ and $\eta^\sharp\,[u', \emptyset, A'] = r^{\sharp'}$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge, the empty lockset, and the resulting digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $r^\sharp \sqsubseteq r^{\sharp'}$ holds. Then, by definition:

$$\eta'_{\mathsf{mm}}[u', \emptyset, A'] = \gamma_{u',\emptyset,A'}(r^{\sharp'}) = \{t \in \mathcal{T} \mid \mathsf{loc}\,t = u', L_t = \emptyset, \alpha_{\mathcal{A}}\,t = A', \beta\,t \in \gamma_{\mathcal{R}}\,(r^{\sharp'})\}$$

For every trace $t \in \eta^i\,[u, \emptyset, A_0]$, let $\beta\, t = \sigma$. By induction hypothesis, $\sigma \in \gamma_{\mathcal{R}}\,(r^\sharp)$ and also $\sigma|_{\mathcal{X}} \in \gamma_{\mathcal{R}}\,(r^\sharp)$. Let $t' \in [\![(u, \mathsf{initMT}, u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = \emptyset$, $\alpha_{\mathcal{A}}\, t' = A'$, and $\beta\, t' = \sigma$. Thus, $\beta\, t' \in \gamma_{\mathcal{R}}\,(r^{\sharp'})$ and $(\beta\, t')|_{\mathcal{X}} \subseteq \gamma_{\mathcal{R}}\,(r^{\sharp'})$. Altogether, $t' \in \eta'_{\mathsf{mm}}\,[u', \emptyset, A']$ holds for all $t \in \eta^i\,[u, \emptyset, A_0]$. We conclude that the return value of $[\![([u, \emptyset, A_0], \mathsf{initMT}, u'), A']\!]_{\mathsf{mm}}\,\eta^i$ is subsumed by the value $\eta'_{\mathsf{mm}}\,[u', \emptyset, A']$ and that auxiliary property (a) holds for the next approximation as well. Next, we consider the side-effects. Consider some mutex a and associated cluster $Q \in \mathcal{Q}_a$. The side-effects to the unknown $[a, Q, A']$ are the given by

$$\begin{array}{lcl} \rho_1 & = & \{[a, Q, A'] \mapsto T\} \\ \rho_1^\sharp & = & \{[a, Q, A'] \mapsto ([\![\{g \leftarrow 0 \mid g \in Q\}]\!]^\sharp_{\mathcal{R}} \top)\} \end{array}$$

Let $\eta^\sharp\,[a, Q, A'] = r^\sharp_\rho$ be the value given by $\eta^\sharp$ for the unknown receiving the side-effect. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $([\![\{g \leftarrow 0 \mid g \in Q\}]\!]^\sharp_{\mathcal{R}} \top) \sqsubseteq r^\sharp_\rho$ holds. Then, by definition:

$$\begin{array}{ll} & \eta'_{\mathsf{mm}}[a, Q, A'] = \gamma_{a,A'}(r^\sharp_\rho) \\ = & \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{unlock}(a) \vee \mathsf{last}\, t = \mathsf{initMT}, \alpha_{\mathcal{A}}\, t = A', \beta\, t \in \gamma_{\mathcal{R}}\, r^\sharp_\rho\} \end{array}$$

Consider a trace $t \in \eta^i\,[u, \emptyset, A_0]$ and let $t' = [\![(u, \mathsf{initMT}, u')]\!]_{\mathcal{T}}\{t\}$. Then, $\mathsf{last}\, t' = \mathsf{initMT}$, $\alpha_{\mathcal{A}}\, t' = A'$. Let $\beta\, t' = \sigma$. Then, as $\mathsf{last\_write}_g\, t' = \bot$ for $g \in \mathcal{G}$, we have $\sigma\, g = 0$. Thus (as $Q \subseteq \mathcal{G}$ by construction), $\beta\, t' \in \gamma_{\mathcal{R}}([\![\{g \leftarrow 0 \mid g \in Q\}]\!]^\sharp_{\mathcal{R}}) \subseteq \gamma_{\mathcal{R}}\, r^\sharp_\rho$. Therefore, $t' \in \eta'_{\mathsf{mm}}\,[a, Q, A']$ holds for all $t \in \eta^i\,[u, \emptyset, A_0]$. Hence, all side-effects for initMT of $\mathcal{C}_{\mathsf{mm}}$ are accounted for in η'_{mm}, and the claim holds.

For this analysis, the right-hand sides for **reading from a global**, **writing to a global**, and **computations on locals** (other than assignments of ?) take the same form. We exemplify the proof for a read from a global $x = g$; the other cases are analogous. Consider an edge $(u, x = g, u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, x = g]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![([u, S, A_0], x = g, u')]\!]_{\mathsf{mm}}\,\eta^i \quad \subseteq \quad (\eta'_{\mathsf{mm}}, \eta'_{\mathsf{mm}}\,[u', S, A'])$$

We have

$$\begin{array}{lcl} [\![([u, S, A_0], x = g, u')]\!]_{\mathsf{mm}}\,\eta_{\mathsf{mm}} & = & (\emptyset, [\![(u, x = g, u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u, S, A_0])) \\ [\![[u, S, A_0], x = g]\!]^\sharp \eta^\sharp & = & (\emptyset, [\![x \leftarrow g]\!]^\sharp_{\mathcal{R}}\,(\eta^\sharp\,[u, S, A_0])) \end{array}$$

Let $\eta^\sharp\,[u, S, A_0] = r^\sharp$ and $\eta^\sharp\,[u', S, A'] = r^{\sharp'}$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge, the current lockset, and the resulting digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $[\![x \leftarrow g]\!]^\sharp_{\mathcal{R}} r^\sharp \sqsubseteq r^{\sharp'}$ holds. Then, by definition:

$$\eta'_{\mathsf{mm}}[u', S, A'] = \gamma_{u',S,A'}(r^{\sharp'}) = \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = S, \alpha_{\mathcal{A}}\, t = A', \beta\, t \in \gamma_{\mathcal{R}}\,(r^{\sharp'})\}$$

For every trace $t \in \eta^i\,[u, S, A_0]$, let $\beta\, t = \sigma$. By induction hypothesis, $\sigma \in \gamma_{\mathcal{R}}\,(r^\sharp)$ as well as $\sigma|_{\mathcal{X} \cup \{g \in \mathcal{G}, \bar{\mathcal{M}}[g] \cap S \neq \emptyset\}} \subseteq \gamma_{\mathcal{R}}\,(r^\sharp)$. Let $t' = [\![(u, x = g, u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S$,

$\alpha_{\mathcal{A}}\, t' = A'$, $\sigma' = \beta\, t' = \sigma \oplus \{x \mapsto [\![g]\!]_{\mathcal{E}xp}\, \sigma\}$ Here, we use the function $[\![\cdot]\!]_{\mathcal{E}xp}$ to evaluate the global variable, which is not problematic as, by construction, $\sigma\, g$ contains the current value of g. Recall from Fig. 3.2 that $\gamma_{\mathcal{R}}\, ([\![x \leftarrow e]\!]^\sharp_{\mathcal{R}}\, r) \supseteq \{\sigma \oplus \{x \mapsto [\![e]\!]_{\mathcal{E}xp}\, \sigma\} \mid \sigma \in \gamma_{\mathcal{R}} r\}$ holds. We obtain

$$\begin{array}{rcl}
\sigma' & \in & \sigma'|_{\mathcal{X} \cup \{g \in \mathcal{G}, \bar{\mathcal{M}}[g] \cap S \neq \emptyset\}} \\
& = & (\sigma \oplus \{x \mapsto [\![g]\!]_{\mathcal{E}xp}\, \sigma\})\big|_{\mathcal{X} \cup \{g \in \mathcal{G}, \bar{\mathcal{M}}[g] \cap S \neq \emptyset\}} \\
& = & \{\hat{\sigma} \oplus \{x \mapsto [\![g]\!]_{\mathcal{E}xp}\, \sigma\} \mid \hat{\sigma} \in \sigma|_{\mathcal{X} \cup \{g \in \mathcal{G}, \bar{\mathcal{M}}[g] \cap S \neq \emptyset\}}\} \\
& \subseteq & \{\hat{\sigma} \oplus \{x \mapsto [\![g]\!]_{\mathcal{E}xp}\, \hat{\sigma}\} \mid \hat{\sigma} \in \gamma_{\mathcal{R}}\, r^\sharp\} \\
& \subseteq & \gamma_{\mathcal{R}}\, ([\![x \leftarrow g]\!]^\sharp_{\mathcal{R}} r^\sharp) \\
& \subseteq & \gamma_{\mathcal{R}}(r^{\sharp'})
\end{array}$$

where we use that x and g are in $\mathcal{X} \cup \{g \in \mathcal{G}, \bar{\mathcal{M}}[g] \cap S \neq \emptyset\}$. Altogether, $t' \in \eta'_{\mathsf{mm}}\, [u', S, A']$ holds for all $t \in \eta^i\, [u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], x = g, u')]\!]_{\mathsf{mm}}\, \eta^i$ is subsumed by the value $\eta'_{\mathsf{mm}}\, [u', S, A']$ and that auxiliary property (a) holds for the next approximation as well for the unknown $[u', S, A']$. Since the constraint causes no side-effects, the claim holds.

For **guards** as well as non-deterministic assignments $x = ?$, the proof proceeds along the same lines as the proof given in Section 6.1.1, but argues via the soundness of $[\![?e]\!]^\sharp_{\mathcal{R}}$, respectively the soundness of the relational version of $\cdot|_{\mathcal{V}ars \setminus \{x\}}$. We do not detail this here.

Next, consider the constraints corresponding to **locking** a mutex a. Consider an edge $(u, \mathsf{lock}(a), u') \in \mathcal{E}$ and digests A', A_0, and A_1 such that $A' \in [\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}(A_0, A_1)$. We verify that

$$[\![([u, S, A_0], \mathsf{lock}(a), u'), A_1]\!]_{\mathsf{mm}}\, \eta^i \quad \subseteq \quad (\eta'_{\mathsf{mm}}, \eta'_{\mathsf{mm}}\, [u', S \cup \{a\}, A'])$$

We have

$$\begin{array}{l}
[\![([u, S, A_0], \mathsf{lock}(a), u'), A_1]\!]_{\mathsf{mm}}\, \eta_{\mathsf{mm}} = \\
\quad \textbf{let } T_1 = \bigcap \{\eta_{\mathsf{mm}}\, [a, Q, A_1] \mid Q \in \mathcal{Q}_a\} \textbf{ in} \\
\quad \textbf{let } T = [\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\, [u, S, A_0], T_1) \textbf{ in} \\
\quad (\emptyset, T)
\end{array}$$

$$\begin{array}{l}
[\![[u, S, A_0], \mathsf{lock}(a), A_1]\!]^\sharp \eta^\sharp = \\
\quad \textbf{let } r^\sharp = \eta^\sharp\, [u, S, A_0] \textbf{ in} \\
\quad \textbf{let } r^{\sharp''} = r^\sharp \sqcap \left(\prod_{Q \in \mathcal{Q}_a} \eta^\sharp\, [a, Q, A_1]\right) \textbf{ in} \\
\quad (\emptyset, r^{\sharp''})
\end{array}$$

Let $\eta^\sharp\, [u, S, A_0] = r^\sharp$ and $\eta^\sharp\, [u', S \cup \{a\}, A'] = r^{\sharp'}$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge and the resulting lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $r^{\sharp''} \sqsubseteq r^{\sharp'}$ holds. Then, by definition:

$$\begin{array}{rl}
& \eta'_{\mathsf{mm}}[u', S \cup \{a\}, A'] = \gamma_{u', S \cup \{a\}, A'}(r^{\sharp'}) \\
= & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = S \cup \{a\}, \alpha_{\mathcal{A}}\, t = A', \beta\, t \in \gamma_{\mathcal{R}}\, (r^{\sharp'})\}
\end{array}$$

For every trace $t \in \eta^i\,[u, S, A_0]$, let $\beta\, t = \sigma$. By induction hypothesis, $\sigma \in \gamma_{\mathcal{R}}\,(r^\sharp)$, as well as $\sigma|_{\mathcal{X} \cup \{g \in \mathcal{G}, \bar{\mathcal{M}}[g] \cap S \neq \emptyset\}} \subseteq \gamma_{\mathcal{R}}\,(r^\sharp)$. For any trace $t_1 \in \bigcap\{\eta^i\,[a, Q, A_1] \mid Q \in \mathcal{Q}_a\}$, let $\beta\, t_1 = \sigma_1$. Then, by induction hypothesis, $\sigma_1 \in \bigcap\{\gamma_{\mathcal{R}}(\eta^\sharp\,[a, Q, A_1]) \mid Q \in \mathcal{Q}_a\}$. Also, we have $\sigma_1|_Q \subseteq \gamma_{\mathcal{R}}(\eta^\sharp\,[a, Q, A_1])$ for any $Q \in \mathcal{Q}_a$. As a consequence, we obtain $\sigma_1|_{\bar{\mathcal{G}}[a]} \subseteq \bigcap\{\gamma_{\mathcal{R}}(\eta^\sharp\,[a, Q, A_1]) \mid Q \in \mathcal{Q}_a\}$.

Let $t' \in [\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}(\{t\}, \{t_1\})$. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S \cup \{a\}$, and $\alpha_{\mathcal{A}}\, t' = A'$. Let $\sigma' = \beta\, t'$. Then, $\forall x \in \mathcal{X}, \sigma'\, x = \sigma\, x$. For all globals protected by a, i.e., $g \in \bar{\mathcal{G}}[a]$, on the other hand, we have either

- $S \cap \bar{\mathcal{M}}[g] \neq \emptyset$, in which case $\sigma'\, g = \sigma\, g = \sigma_1\, g$, or
- $S \cap \bar{\mathcal{M}}[g] = \emptyset$, in which case $\sigma' g = \sigma_1 g$.

Thus,

$$\begin{array}{rcl} \sigma' & \in & \sigma|_{\mathcal{X} \cup \{g \in \mathcal{G}, \bar{\mathcal{M}}[g] \cap S \neq \emptyset\}} \cap \sigma_1|_{\bar{\mathcal{G}}[a]} \\ & \subseteq & \gamma_{\mathcal{R}}\,(r^\sharp) \cap \bigcap\{\gamma_{\mathcal{R}}(\eta^\sharp\,[a, Q, A_1]) \mid Q \in \mathcal{Q}_a\} \\ & \subseteq & \gamma_{\mathcal{R}}\,\left(r^\sharp \sqcap \prod_{Q \in \mathcal{Q}_a}(\eta^\sharp\,[a, Q, A_1])\right) \\ & = & \gamma_{\mathcal{R}}\, r^{\sharp\prime\prime} \subseteq \gamma_{\mathcal{R}}\, r^{\sharp\prime} \end{array}$$

where we rely on the $\sqcap$ operation in $\mathcal{R}$ being sound w.r.t. the intersection of concretizations. Also, we have $\sigma'|_{\mathcal{X} \cup \{g \in \mathcal{G}, \bar{\mathcal{M}}[g] \cap (S \cup \{a\}) \neq \emptyset\}} \subseteq \sigma|_{\mathcal{X} \cup \{g \in \mathcal{G}, \bar{\mathcal{M}}[g] \cap S \neq \emptyset\}} \cap \sigma_1|_{\bar{\mathcal{G}}[a]}$. Altogether, $t' \in \eta'_{\mathsf{mm}}\,[u', S \cup \{a\}, A']$ holds for all $t \in \eta^i\,[u, S, A_0]$ and $t_1 \in \bigcap\{\eta^i\,[a, Q, A_1] \mid Q \in \mathcal{Q}_a\}$. We conclude that the return value of $[\![([u, S, A_0], \mathsf{lock}(a), u'), A_1]\!]_{\mathsf{mm}}\, \eta^i$ is subsumed by the value $\eta'_{\mathsf{mm}}\,[u', S \cup \{a\}, A']$ and that auxiliary property (a) holds for the next approximation as well for the unknown $[u', S \cup \{a\}, A']$. Since the constraint causes no side-effects, the claim holds.

Next, consider the constraints corresponding to **unlocking** a mutex a. Consider an edge $(u, \mathsf{unlock}(a), u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, \mathsf{unlock}(a)]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![([u, S, A_0], \mathsf{unlock}(a), u'), A']\!]_{\mathsf{mm}}\, \eta^i \quad \subseteq \quad (\eta'_{\mathsf{mm}}, \eta'_{\mathsf{mm}}\,[u', S \setminus \{a\}, A'])$$

We have

$$\begin{array}{l} [\![([u, S, A_0], \mathsf{unlock}(a), u'), A']\!]_{\mathsf{mm}}\, \eta_{\mathsf{mm}} = \\ \quad \textbf{let } T = [\![(u, \mathsf{unlock}(a), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u, S, A_0]) \textbf{ in} \\ \quad \textbf{let } \rho = \{[a, Q, A'] \mapsto T \mid Q \in \mathcal{Q}_a\} \textbf{ in} \\ \quad (\rho, T) \end{array}$$

$$\begin{array}{l} [\![[u, S, A_0], \mathsf{unlock}(a), A']\!]^\sharp \eta^\sharp = \\ \quad \textbf{let } r^\sharp = \eta^\sharp\,[u, S, A_0] \textbf{ in} \\ \quad \textbf{let } \rho^\sharp = \{[a, Q, A'] \mapsto r^\sharp|_Q \mid Q \in \mathcal{Q}_a\} \textbf{ in} \\ \quad \textbf{let } r^{\sharp\prime\prime} = r^\sharp|_{\mathcal{X} \cup \bigcup\{\bar{\mathcal{G}}[a'] \mid a' \in (S \setminus a)\}} \textbf{ in} \\ \quad (\rho^\sharp, r^{\sharp\prime\prime}) \end{array}$$

Let $\eta^\sharp\,[u, S, A_0] = r^\sharp$ and $\eta^\sharp\,[u', S \setminus \{a\}, A'] = r^{\sharp'}$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge and the resulting lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $r^{\sharp''} \sqsubseteq r^{\sharp'}$ holds. Then, by definition:

$$\begin{array}{rl} & \eta'_{\mathsf{mm}}[u', S \setminus \{a\}, A'] = \gamma_{u', S\setminus\{a\}, A'}(r^{\sharp'}) \\ = & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = S \setminus \{a\}, \alpha_{\mathcal{A}}\, t = A', \beta\, t \in \gamma_{\mathcal{R}}\,(r^{\sharp'})\} \end{array}$$

For every trace $t \in \eta^i\,[u, S, A_0]$, let $\beta\, t = \sigma$. By induction hypothesis, $\sigma \in \gamma_{\mathcal{R}}\,(r^\sharp)$ and $\sigma|_{\mathcal{X} \cup \bigcup\{\bar{\mathcal{G}}[a'] \mid a' \in S\}} \subseteq \gamma_{\mathcal{R}}\,(r^\sharp)$. Let $t' \in [\![(u, \mathsf{unlock}(a), u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S \setminus \{a\}$, $\alpha_{\mathcal{A}}\, t' = A'$, and $\beta\, t' = \sigma$. Thus,

$$\beta\, t' \in \gamma_{\mathcal{R}}\,(r^\sharp) \subseteq \gamma_{\mathcal{R}}\,\left(r^\sharp|_{\mathcal{X} \cup \bigcup\{\bar{\mathcal{G}}[a'] \mid a' \in (S\setminus a)\}}\right) = \gamma_{\mathcal{R}}\, r^{\sharp''} \subseteq \gamma_{\mathcal{R}}\,(r^{\sharp'})$$

and also $(\beta\, t')|_{\mathcal{X} \cup \bigcup\{\bar{\mathcal{G}}[a'] \mid a' \in (S\setminus a)\}} \subseteq \gamma_{\mathcal{R}}\,(r^{\sharp'})$. Altogether, $t' \in \eta'_{\mathsf{mm}}\,[u', S \setminus \{a\}, A']$ holds for all $t \in \eta^i\,[u, S, A_0]$. We conclude that the value $\eta'_{\mathsf{mm}}\,[u', S \setminus \{a\}, A']$ subsumes the return value of $[\![([u, S, A_0], \mathsf{unlock}(a), u'), A']\!]_{\mathsf{mm}}\,\eta^i$ and that auxiliary property (a) holds for the unknown $[u', S \setminus \{a\}, A']$ in the next approximation. Next, we consider the side-effects. Consider some mutex a and associated cluster $Q \in \mathcal{Q}_a$. The side-effects to the unknown $[a, Q, A']$ are the given by

$$\begin{array}{rcl} \rho_1 & = & \{[a, Q, A'] \mapsto T\} \\ \rho_1^\sharp & = & \{[a, Q, A'] \mapsto r^\sharp|_Q\} \end{array}$$

As $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $r^\sharp|_Q \sqsubseteq \eta^\sharp\,[a, Q, A']$ holds. Now consider a trace t' as constructed before. Then, $\mathsf{last}\, t' = \mathsf{unlock}(a)$, $\alpha_{\mathcal{A}}\, t' = A'$. As $\beta' t = \beta\, t$, we have $\beta t' \in \gamma_{\mathcal{R}}\, r^\sharp$ by induction hypothesis. Thus, $\beta\, t' \in \gamma_{\mathcal{R}}(r^\sharp) \subseteq \gamma_{\mathcal{R}}\,(r^\sharp|_Q) \subseteq \gamma_{\mathcal{R}}\,(\eta^\sharp\,[a, Q, A'])$. Therefore, $t' \in \gamma_{a,A'}(\eta^\sharp\,[a, Q, A']) = \eta'_{\mathsf{mm}}\,[a, Q, A']$ holds for all $t \in \eta^i\,[u, S, A_0]$. Also, we get $(\beta\, t')|_Q \subseteq \gamma_{\mathcal{R}}\,(r^\sharp|_Q)$. Hence, all side-effects for $\mathsf{unlock}(a)$ of $\mathcal{C}_{\mathsf{mm}}$ are accounted for in η'_{mm}, auxiliary property (a) also holds for the next approximation for unknowns receiving side-effects, and the claim holds.

Next, consider the constraints corresponding to **starting a new thread**. Consider an edge $(u, x = \mathsf{create}(u_1), u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, x = \mathsf{create}(u_1)]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![([u, S, A_0], x = \mathsf{create}(u_1), u')]\!]_{\mathsf{mm}}\,\eta^i \quad \subseteq \quad (\eta'_{\mathsf{mm}}, \eta'_{\mathsf{mm}}\,[u', S, A'])$$

We have

$$
\begin{aligned}
&[\![([u,S,A_0], x = \mathsf{create}(u_1), u')]\!]_{\mathsf{mm}}\,\eta_{\mathsf{mm}} = \\
&\quad \textbf{let } T = [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u,S,A_0]) \textbf{ in} \\
&\quad \textbf{let } \rho = \{[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\,u\,u_1\,A_0] \mapsto \mathsf{new}\,u_1\,(\eta_{\mathsf{mm}}\,[u,S,A_0])\} \\
&\quad (\rho, T)
\end{aligned}
$$

$$
\begin{aligned}
&[\![[u,S,A_0], x = \mathsf{create}(u_1)]\!]^\sharp\eta = \\
&\quad \textbf{let } r^\sharp = \eta\,[u,S,A_0] \textbf{ in} \\
&\quad \textbf{let } i = \nu^\sharp\,u\,(\mathsf{unlift}\,r^\sharp)\,u_1 \textbf{ in} \\
&\quad \textbf{let } r^{\sharp\prime\prime} = [\![x \leftarrow^\sharp i]\!]^\sharp_{\mathcal{R}} r^\sharp \\
&\quad \textbf{let } r^\sharp_\rho = \left([\![\mathsf{self} \leftarrow^\sharp i]\!]^\sharp_{\mathcal{R}}\,r^\sharp\right)\Big|_{\mathcal{X}} \textbf{ in} \\
&\quad \textbf{let } \rho^\sharp = \{[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\,u\,u_1\,A_0] \mapsto r^\sharp_\rho\} \textbf{ in} \\
&\quad (\rho^\sharp, r^{\sharp\prime\prime})
\end{aligned}
$$

where we, for notational convenience, denote by $\mathsf{new}^\sharp_{\mathcal{A}}\,u\,u_1\,A_0$ the only element of this singleton set. Let $\eta^\sharp\,[u,S,A_0] = r^\sharp$ and $\eta^\sharp\,[u',S,A'] = r^{\sharp\prime}$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge and the resulting lockset and digest. Since $\eta^\sharp$ is a solution of $C^\sharp$, $r^{\sharp\prime\prime} \sqsubseteq r^{\sharp\prime}$ holds. Then, by definition:

$$\eta'_{\mathsf{mm}}[u',S,A'] = \gamma_{u',S,A'}(r^{\sharp\prime}) = \{t \in \mathcal{T} \mid \mathsf{loc}\,t = u', L_t = S, \alpha_{\mathcal{A}}\,t = A', \beta\,t \in \gamma_{\mathcal{R}}\,(r^{\sharp\prime})\}$$

For every trace $t \in \eta^i\,[u,S,A_0]$, let $\beta\,t = \sigma$. By induction hypothesis, $\sigma \in \gamma_{\mathcal{R}}\,(r^\sharp)$, as well as $\sigma|_{\mathcal{X}\cup\{g\in\mathcal{G},\bar{\mathcal{M}}[g]\cap S\neq\emptyset\}} \subseteq \gamma_{\mathcal{R}}\,(r^\sharp)$.

Let $t' \in [\![(u, x = \mathsf{create}(u_1), u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\,t' = u'$, $L_{t'} = S$. Let $\sigma' = \beta\,t' = \sigma \oplus \{x \mapsto \nu(t)\}$. Then, as $\nu(t) \in \gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}(\nu^\sharp\,u\,(\mathsf{unlift}\,r^\sharp)\,u_1)$ by (3.2), and the properties of the building blocks of the composite operation $[\![\cdot \leftarrow^\sharp \cdot]\!]^\sharp_{\mathcal{R}}$ (including Fig. 3.2),

$$
\begin{aligned}
\sigma' = \sigma \oplus \{x \mapsto \nu(t)\} \quad &\in \quad (\sigma \oplus \{x \mapsto \nu(t)\})|_{\mathcal{X}\cup\{g\in\mathcal{G},\bar{\mathcal{M}}[g]\cap S\neq\emptyset\}} \\
&= \quad \{\hat{\sigma} \oplus \{x \mapsto \nu(t)\} \mid \hat{\sigma} \in \sigma|_{\mathcal{X}\cup\{g\in\mathcal{G},\bar{\mathcal{M}}[g]\cap S\neq\emptyset\}}\} \\
&\subseteq \quad \{\hat{\sigma} \oplus \{x \mapsto \nu(t)\} \mid \hat{\sigma} \in \gamma_{\mathcal{R}}\,r^\sharp\} \\
&\subseteq \quad \gamma_{\mathcal{R}}\,([\![x \leftarrow^\sharp \nu^\sharp\,u\,(\mathsf{unlift}\,r^\sharp)\,u_1]\!]^\sharp_{\mathcal{R}} r^\sharp) \\
&= \quad \gamma_{\mathcal{R}}(r^{\sharp\prime\prime}) \subseteq \gamma_{\mathcal{R}}(r^{\sharp\prime})
\end{aligned}
$$

where the second equality exploits that $x \in \mathcal{X}$ holds. Altogether, $t' \in \eta'_{\mathsf{mm}}\,[u',S,A']$ holds for all $t \in \eta^i\,[u,S,A_0]$. We conclude that the return value of $[\![([u,S,A_0], x = \mathsf{create}(u_1), u'), A']\!]_{\mathsf{mm}}\,\eta^i$ is subsumed by the value $\eta'_{\mathsf{mm}}\,[u',S,A']$ and that auxiliary property (a) holds for the next approximation as well for the unknown $[u',S,A']$.

Next, we consider the side-effects of the corresponding right-hand-side functions for a t as considered previously.

$$
\begin{aligned}
\rho &= \{[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\,u\,u_1\,A_0] \mapsto \mathsf{new}\,u_1\,\{t\}\} \\
\rho^\sharp &= \{[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\,u\,u_1\,A_0] \mapsto r^\sharp_\rho\}
\end{aligned}
$$

Let $\eta^\sharp\,[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\,u\,u_1\,A_0] = r^{\sharp'}_\rho$ the value provided by $\mathcal{C}^\sharp$ for the unknown receiving the side-effect. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $r^\sharp_\rho \sqsubseteq r^{\sharp'}_\rho$ holds. Then, by definition:

$$\begin{aligned} & \eta'_{\mathsf{mm}}[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\,u\,u_1\,A_0] = \gamma_{u_1,\emptyset,\mathsf{new}^\sharp_{\mathcal{A}}\,u\,u_1\,A_0}(r^{\sharp'}_\rho) \\ = \quad & \{t \in \mathcal{T} \mid \mathsf{loc}\,t = u_1, L_t = \emptyset, \alpha_{\mathcal{A}}\,t = \mathsf{new}^\sharp_{\mathcal{A}}\,u\,u_1\,A_0, \beta\,t \in \gamma_{\mathcal{R}}\,(r^{\sharp'}_\rho)\} \end{aligned}$$

Let $t'' = \mathsf{new}\,u_1\,\{t\}$. Then, $\mathsf{loc}\,t'' = u_1$, $L_{t''} = \emptyset$, $\alpha_{\mathcal{A}}\,t'' = \mathsf{new}^\sharp_{\mathcal{A}}\,u\,u_1\,A_0$, and let $\sigma'' = \beta\,t'' = \sigma \oplus \{\mathsf{self} \mapsto \nu(t)\}$. Then,

$$\begin{aligned} \sigma'' = \sigma \oplus \{\mathsf{self} \mapsto \nu(t)\} \quad & \in \quad (\sigma \oplus \{\mathsf{self} \mapsto \nu(t)\})|_{\mathcal{X}} \\ & \subseteq \quad \bigcup\{(\hat{\sigma} \oplus \{\mathsf{self} \mapsto \nu(t)\})|_{\mathcal{X}} \mid \hat{\sigma} \in \gamma_{\mathcal{R}} r^\sharp\} \\ & \subseteq \quad \bigcup\{\hat{\sigma}|_{\mathcal{X}} \mid \hat{\sigma} \in \gamma_{\mathcal{R}}\,([\![\mathsf{self} \leftarrow^\sharp \nu^\sharp\,u\,(\mathsf{unlift}\,r^\sharp)\,u_1]\!]^\sharp_{\mathcal{R}} r^\sharp)\} \\ & \subseteq \quad \gamma_{\mathcal{R}}\,\left(([\![\mathsf{self} \leftarrow^\sharp \nu^\sharp\,u\,(\mathsf{unlift}\,r^\sharp)\,u_1]\!]^\sharp_{\mathcal{R}} r^\sharp)\Big|_{\mathcal{X}}\right) \\ & = \quad \gamma_{\mathcal{R}}(r^\sharp_\rho) \subseteq \gamma_{\mathcal{R}}(r^{\sharp'}_\rho) \end{aligned}$$

Altogether, $t'' \in \eta'_{\mathsf{mm}}\,[u_1, \emptyset, \mathsf{new}^\sharp_{\mathcal{A}}\,u\,u_1\,A_0]$ holds for all $t \in \eta^i\,[u, S, A_0]$. Hence, all side-effects for $x = \mathsf{create}(u_1)$ of $\mathcal{C}_{\mathsf{mm}}$ are accounted for in η'_{mm}, auxiliary property (a) also holds for the next approximation for unknowns receiving side-effects and the claim holds.

Next, consider the constraints corresponding to **returning** from a thread. Consider an edge $(u, \mathsf{return}, u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u, \mathsf{return}]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![([u, S, A_0], \mathsf{return}, u'), A']\!]_{\mathsf{mm}}\,\eta^i \quad \subseteq \quad (\eta'_{\mathsf{mm}}, \eta'_{\mathsf{mm}}\,[u', S, A'])$$

We have

$$\begin{array}{l} [\![([u, S, A_0], \mathsf{return}, u'), A']\!]_{\mathsf{mm}}\,\eta_{\mathsf{mm}} = \\ \quad \textbf{let}\ T = [\![(u, \mathsf{return}, u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u, S, A_0])\ \textbf{in} \\ \quad \textbf{let}\ \rho = \{[i, A'] \mapsto \{t \mid t \in T, \mathsf{id}\,t = i\} \mid i \in \mathcal{V}_{\mathsf{tid}}\}\ \textbf{in} \\ \quad (\rho, T) \end{array}$$

$$\begin{array}{l} [\![[u, S, A_0], \mathsf{return}, A']\!]^\sharp \eta^\sharp = \\ \quad \textbf{let}\ r^\sharp = \eta^\sharp\,[u, S, A_0]\ \textbf{in} \\ \quad \textbf{let}\ I^\sharp = (\mathsf{unlift}\,r^\sharp)\,\mathsf{self}\ \textbf{in} \\ \quad \textbf{let}\ r^\sharp_\rho = r^\sharp\big|_{\{\mathsf{ret}\}}\ \textbf{in} \\ \quad \textbf{let}\ \rho^\sharp = \left\{[i^\sharp, A'] \mapsto r^\sharp_\rho \mid i^\sharp \in I^\sharp\right\}\ \textbf{in} \\ \quad (\rho^\sharp, r^\sharp) \end{array}$$

Let $\eta^\sharp\,[u, S, A_0] = r^\sharp$ and $\eta^\sharp\,[u', S, A'] = r^{\sharp'}$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge and the resulting lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $r^\sharp \sqsubseteq r^{\sharp'}$ holds. Then, by definition:

$$\eta'_{\mathsf{mm}}[u', S, A'] = \gamma_{u',S,A'}(r^{\sharp'}) = \{t \in \mathcal{T} \mid \mathsf{loc}\,t = u', L_t = S, \alpha_{\mathcal{A}}\,t = A', \beta\,t \in \gamma_{\mathcal{R}}\,(r^{\sharp'})\}$$

For every trace $t \in \eta^i\,[u, S, A_0]$, let $\beta\, t = \sigma$. By induction hypothesis, $\sigma \in \gamma_{\mathcal{R}}\,(r^\sharp)$ and $\sigma|_{\mathcal{X} \cup \{g \in \mathcal{G}, \bar{\mathcal{M}}[g] \cap S \neq \emptyset\}} \subseteq \gamma_{\mathcal{R}}\,(r^\sharp)$ both hold. Let $t' \in [\![(u, \mathsf{return}, u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S$. Let $\sigma' = \beta\, t' = \beta\, t = \sigma$. Thus,

$$\sigma' \in \sigma'\big|_{\mathcal{X} \cup \{g \in \mathcal{G}, \bar{\mathcal{M}}[g] \cap S \neq \emptyset\}} = \sigma|_{\mathcal{X} \cup \{g \in \mathcal{G}, \bar{\mathcal{M}}[g] \cap S \neq \emptyset\}} \subseteq \gamma_{\mathcal{R}}\,(r^\sharp) \subseteq \gamma_{\mathcal{R}}\,(r^{\sharp'})$$

Altogether, $t' \in \eta'_{\mathsf{mm}}\,[u', S, A']$ holds for all $t \in \eta^i\,[u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], \mathsf{return}, u'), A']\!]_{\mathsf{mm}}\,\eta^i$ is subsumed by the value $\eta'_{\mathsf{mm}}\,[u', S, A']$ and that auxiliary property (a) also holds for the next approximation for the unknown $[u', S, A']$.

Next, we consider the side-effects of the corresponding right-hand-side functions for local traces t and t' as considered previously.

$$\begin{array}{lcl} \rho & = & \{[\mathsf{id}\, t', A'] \mapsto \{t'\}\} \\ \rho^\sharp & = & \left\{[i^\sharp, A'] \mapsto r^\sharp_\rho \mid i^\sharp \in (\mathsf{unlift}\, r^\sharp)\, \mathsf{self}\right\} \end{array}$$

where we use that $\mathsf{id}\, t = \mathsf{id}\, t'$. As $\sigma \in \gamma_{\mathcal{R}}\,(r^\sharp)$, we have $\mathsf{id}\, t' = \sigma\, \mathsf{self} \in \gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}\,((\mathsf{unlift}\, r^\sharp)\, \mathsf{self})$. As $\mathcal{V}^\sharp_{\mathbf{tid}}$ is a powerset lattice, and the concretization is defined by the union of concretizations of the singleton sets, there is at least one $i^\sharp_\rho \in ((\mathsf{unlift}\, r^\sharp)\, \mathsf{self})$, such that $\mathsf{id}\, t' \in \gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}\,\{i^\sharp_\rho\}$. Consider one such $i^\sharp_\rho$, and let $\eta^\sharp\,[i^\sharp_\rho, A'] = r^{\sharp'}_\rho$ the value provided by $\mathcal{C}^\sharp$ for the unknown receiving the side-effect. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $r^\sharp_\rho \sqsubseteq r^{\sharp'}_\rho$ holds. By definition:

$$\eta'_{\mathsf{mm}}[\mathsf{id}\, t', A'] = \bigcup\{\gamma_{(\mathsf{id}\, t'), A'}(\eta^\sharp\,[i^\sharp, A']) \mid i^\sharp \in S_{\mathcal{V}^\sharp_{\mathbf{tid}}}, \mathsf{id}\, t' \in (\gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}\,\{i^\sharp\})\}$$

Now consider

$$\gamma_{(\mathsf{id}\, t'), A'}(\eta^\sharp\,[i^\sharp_\rho, A']) \quad \subseteq \quad \eta'_{\mathsf{mm}}[\mathsf{id}\, t', A']$$

Then, by definition:

$$\begin{array}{rl} & \gamma_{(\mathsf{id}\, t'), A'}(\eta^\sharp\,[i^\sharp_\rho, A']) = \gamma_{(\mathsf{id}\, t'), A'}\left(r^{\sharp'}_\rho\right) \\ = & \{t'' \in \mathcal{T} \mid \mathsf{last}\, t'' = \mathsf{return}, \alpha_{\mathcal{A}}\, t'' = A', \mathsf{id}\, t'' = \mathsf{id}\, t', \beta\, t'' \in \gamma_{\mathcal{R}}\, r^{\sharp'}_\rho\} \end{array}$$

Then $\mathsf{last}\, t' = \mathsf{return}$, $\alpha_{\mathcal{A}}\, t' = A'$, (vacuously $\mathsf{id}\, t' = \mathsf{id}\, t'$), and

$$\beta\, t' = \sigma \in \gamma_{\mathcal{R}}\,(r^\sharp) \subseteq \gamma_{\mathcal{R}}\,\left(r^\sharp\big|_{\{\mathsf{ret}\}}\right) = \gamma_{\mathcal{R}}\, r^\sharp_\rho \subseteq \gamma_{\mathcal{R}}\, r^{\sharp'}_\rho$$

Altogether, $t' \in \eta'_{\mathsf{mm}}\,[\mathsf{id}\, t, A']$ holds for all $t \in \eta^i\,[u, S, A_0]$. Hence, all side-effects for return of $\mathcal{C}_{\mathsf{mm}}$ are accounted for in η'_{mm}, and the claim holds.

Next, consider the constraints corresponding to calling **join**. Consider an edge $(u, x = \mathsf{join}(x'), u') \in \mathcal{E}$ and digests A', A_0 and A_1 such that $A' \in [\![u, x = \mathsf{join}(x')]\!]^\sharp_{\mathcal{A}}(A_0, A_1)$. We verify that

$$[\![([u, S, A_0], x = \mathsf{join}(x'), u'), A_1]\!]_{\mathsf{mm}}\,\eta^i \quad \subseteq \quad (\eta'_{\mathsf{mm}}, \eta'_{\mathsf{mm}}\,[u', S, A'])$$

We have

$$
\begin{array}{l}
[\![([u,S,A_0],x=\mathsf{join}(x'),u'),A_1]\!]_{\mathsf{mm}}\,\eta_{\mathsf{mm}} = \\
\quad \textbf{let } T_1 = \bigcup\{\eta_{\mathsf{mm}}\,[t(x'),A_1] \mid t \in \eta_{\mathsf{mm}}\,[u,S,A_0]\} \textbf{ in} \\
\quad \textbf{let } T = [\![(u,x=\mathsf{join}(x'),u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u,S,A_0],T_1) \textbf{ in} \\
\quad (\emptyset,T)
\end{array}
$$

$$
\begin{array}{l}
[\![[u,S,A_0],x{=}\mathsf{join}(x'),A_1]\!]^\sharp\eta^\sharp = \\
\quad \textbf{let } r^\sharp = \eta^\sharp\,[u,S,A_0] \textbf{ in} \\
\quad \textbf{let } v^\sharp = \bigsqcup_{i^\sharp \in ((\mathsf{unlift}\, r^\sharp)\, x')} \mathsf{unlift}\,(\eta^\sharp[i^\sharp,A_1])\ \mathsf{ret} \textbf{ in} \\
\quad \textbf{if } v^\sharp = \bot \textbf{ then} \\
\qquad (\emptyset,\underline{\bot}) \\
\quad \textbf{else} \\
\qquad \textbf{let } r^{\sharp\prime\prime} = [\![x \leftarrow^\sharp v^\sharp]\!]^\sharp_{\mathcal{R}} r^\sharp \textbf{ in} \\
\qquad (\emptyset, r^{\sharp\prime\prime})
\end{array}
$$

Let $\eta^\sharp\,[u,S,A_0] = r^\sharp$ and $\eta^\sharp\,[u',S,A'] = r^{\sharp'}$ the value provided by $\mathcal{C}^\sharp$ for the end point of the given control-flow edge and lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, we either have $v = \bot$ or $r^{\sharp\prime\prime} \sqsubseteq r^{\sharp'}$ holds. Then, by definition:

$$
\begin{array}{rl}
& \eta'_{\mathsf{mm}}[u',S,A'] = \gamma_{u',S,A'}(r^{\sharp'}) \\
= & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = S, \alpha_{\mathcal{A}}\, t = A', \beta\, t \in \gamma_{\mathcal{R}}\,(r^{\sharp'})\}
\end{array}
$$

For every trace $t \in \eta^i\,[u,S,A_0]$, let $\beta\, t = \sigma$ By induction hypothesis, $\sigma \in \gamma_{\mathcal{R}}\,(r^\sharp)$ as well as $\sigma|_{\mathcal{X}\cup\{g\in\mathcal{G},\bar{\mathcal{M}}[g]\cap S\neq\emptyset\}} \subseteq \gamma_{\mathcal{R}}\,(r^\sharp)$. Let

$$
\begin{array}{rcl}
T' & = & [\![(u,x=\mathsf{join}(x'),u')]\!]_{\mathcal{T}}(\{t\},\bigcup\{\eta^i\,[(t'(x')),A_1] \mid t' \in \eta^i\,[u,S,A_0]\}) \\
& = & [\![(u,x=\mathsf{join}(x'),u')]\!]_{\mathcal{T}}(\{t\},\eta^i\,[(t(x')),A_1])
\end{array}
$$

where the equality exploits that for a given local trace t, $[\![(u,x=\mathsf{join}(x'),u')]\!]_{\mathcal{T}}(\{t\},\{t'\})$ only yields a non-empty set if the thread *id* of the thread being joined is the one stored in x'. We distinguish the case where the resulting set of traces is empty and the case where it is empty. In case T' is empty, it is trivially subsumed by $\eta'_{\mathsf{mm}}\,[u',S,A']$.

Consider, thus, local traces $t' \in T'$ and $t'' \in \eta^i\,[(t(x')),A_1]$ such that $\{t'\} = [\![(u,x=\mathsf{join}(x'),u')]\!]_{\mathcal{T}}(\{t\},\{t''\})$. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S$, $\alpha_{\mathcal{A}}\, t' = A'$. Let $\sigma' = \beta\, t'$. Then, $\forall y \in \mathcal{X}\setminus\{x\}, \sigma'\, y = \sigma\, y$. For all globals protected by some currently held mutex $\{g \mid g \in \mathcal{G}, S \cap \bar{\mathcal{M}}[g] \neq \emptyset\}$, we also have $t'(g) = t(g)$ and thus $\sigma'\, g = \sigma\, g$. For x, on the other hand we have $\sigma' x = t'(x) = t''(\mathsf{ret})$. It is thus necessary to relate $t''(\mathsf{ret})$ to $v^\sharp$. We have

$$
\begin{array}{rcl}
t'' & \in & \eta^i\,[t(x'),A_1] \\
& \subseteq & \eta'_{\mathsf{mm}}\,[t(x'),A_1] \\
& = & \bigcup\{\gamma_{t(x'),A_1}(\eta^\sharp\,[i^\sharp,A_1]) \mid i^\sharp \in S_{\mathcal{V}^\sharp_{\mathbf{tid}}}, t(x') \in (\gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}\{i^\sharp\})\} \\
& \subseteq & \bigcup\{\gamma_{t(x'),A_1}(\eta^\sharp\,[i^\sharp,A_1]) \mid i^\sharp \in ((\mathsf{unlift}\, r^\sharp)\, x')\}
\end{array}
$$

and therefore

$$\begin{array}{rl} & \{t''(\mathsf{ret})\} \\ \subseteq & \{t'''(\mathsf{ret}) \mid t''' \in (\bigcup\{\gamma_{t(x'),A_1}(\eta^\sharp\,[i^\sharp,A_1]) \mid i^\sharp \in ((\mathsf{unlift}\,r^\sharp)\,x')\})\} \\ \subseteq & \{t'''(\mathsf{ret}) \mid t''' \in \gamma_{t(x'),A_1}(\bigsqcup\{(\eta^\sharp\,[i^\sharp,A_1]) \mid i^\sharp \in ((\mathsf{unlift}\,r^\sharp)\,x')\})\} \\ \subseteq & \gamma_{\mathcal{V}^\sharp}\,(\mathsf{unlift}\,(\bigsqcup\{(\eta^\sharp\,[i^\sharp,A_1]) \mid i^\sharp \in ((\mathsf{unlift}\,r^\sharp)\,x')\}\,)\,\mathsf{ret}) \\ = & \gamma_{\mathcal{V}^\sharp}\,((\bigsqcup\{\mathsf{unlift}\,(\eta^\sharp\,[i^\sharp,A_1])\,\mathsf{ret} \mid i^\sharp \in ((\mathsf{unlift}\,r^\sharp)\,x')\}\,)) = \gamma_{\mathcal{V}^\sharp}(v^\sharp) \end{array}$$

where the next-to-last step uses the distributive property of unlift (see (3.3) in Section 3.2). As a consequence, we obtain $v^\sharp \neq \bot$, and thus $r^{\sharp''} \sqsubseteq r^{\sharp'}$. Altogether, we have

$$\begin{array}{rl} \sigma' \in & \sigma'|_{\mathcal{X}\cup\{g\in\mathcal{G}|S\cap\bar{\mathcal{M}}[g]\neq\emptyset\}} \\ = & (\sigma \oplus \{x \mapsto t''(\mathsf{ret})\})|_{\mathcal{X}\cup\{g\in\mathcal{G}|S\cap\bar{\mathcal{M}}[g]\neq\emptyset\}} \\ = & \bigcup\{\hat{\sigma} \oplus \{x \mapsto t''(\mathsf{ret})\} \mid \hat{\sigma} \in \sigma|_{\mathcal{X}\cup\{g\in\mathcal{G}|S\cap\bar{\mathcal{M}}[g]\neq\emptyset\}}\} \\ \subseteq & \bigcup\{\hat{\sigma} \oplus \{x \mapsto t''(\mathsf{ret})\} \mid \hat{\sigma} \in \gamma_{\mathcal{R}}\,r^\sharp\} \\ \subseteq & \gamma_{\mathcal{R}}\,([\![x \leftarrow^\sharp v^\sharp]\!]^\sharp_{\mathcal{R}} r^\sharp) = \gamma_{\mathcal{R}}\,(r^{\sharp''}) \subseteq \gamma_{\mathcal{R}}\,(r^{\sharp'}) \end{array}$$

where the second equality follows from $x \in \mathcal{X}$. Thus, $t' \in \eta'_{\mathsf{mm}}\,[u',S,A']$ holds for all $t \in \eta^i\,[u,S,A_0]$. We conclude that the return value of $[\![([u,S,A_0],x = \mathsf{join}(x'),u'),A_1]\!]_{\mathsf{mm}}\,\eta^i$ is subsumed by the value $\eta'_{\mathsf{mm}}\,[u',S,A']$ and that auxiliary property (a) also holds for the next approximation for the unknown $[u',S,A']$. As neither constraint causes any side-effects, the statement holds.

Next, consider the constraints corresponding to calling **signal**. Consider an edge $(u,\mathsf{signal}(s),u') \in \mathcal{E}$ and digests A', A_0 such that $A' \in [\![u,\mathsf{signal}(s)]\!]^\sharp_{\mathcal{A}}(A_0)$. We verify that

$$[\![([u,S,A_0],\mathsf{signal}(s),u'),A']\!]_{\mathsf{mm}}\,\eta^i \quad\subseteq\quad (\eta'_{\mathsf{mm}},\eta_{\mathsf{mm}}\,[u',S,A'])$$

We have

$$\begin{array}{l} [\![([u,S,A_0],\mathsf{signal}(s),u'),A']\!]_{\mathsf{mm}}\,\eta_{\mathsf{mm}} = \\ \quad \textbf{let}\ T = [\![(u,\mathsf{signal}(s),u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u,S,A_0])\ \textbf{in} \\ \quad \textbf{let}\ \rho = \{[s,A'] \mapsto T\}\ \textbf{in} \\ \quad (\emptyset,T) \end{array}$$

$$\begin{array}{l} [\![[u,S,A_0],\mathsf{signal}(s)]\!]^\sharp\eta^\sharp = \\ \quad \textbf{let}\ r^\sharp = \eta^\sharp\,[u,S,A_0]\ \textbf{in} \\ \quad (\{[s,A'] \mapsto r^\sharp\},r^\sharp) \end{array}$$

Let $\eta^\sharp\,[u,S,A_0] = r^\sharp$ and $\eta^\sharp\,[u',S,A'] = r^{\sharp'}$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge and the resulting lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $r^\sharp \sqsubseteq r^{\sharp'}$ holds. Then, by definition:

$$\eta'_{\mathsf{mm}}[u',S,A'] = \gamma_{u',S,A'}(r^{\sharp'}) = \{t \in \mathcal{T} \mid \mathsf{loc}\,t = u', L_t = S, \alpha_{\mathcal{A}}\,t = A', \beta\,t \in \gamma_{\mathcal{R}}\,(r^{\sharp'})\}$$

For every trace $t \in \eta^i\,[u,S,A_0]$, let $\beta\,t = \sigma$. By induction hypothesis, $\sigma \in \gamma_{\mathcal{R}}\,(r^\sharp)$ and $\sigma|_{\mathcal{X}\cup\{g\in\mathcal{G},\bar{\mathcal{M}}[g]\cap S\neq\emptyset\}} \subseteq \gamma_{\mathcal{R}}\,(r^\sharp)$ both hold. Let $t' \in [\![(u,\mathsf{signal}(s),u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\,t' = u'$, $L_{t'} = S$. Let $\sigma' = \beta\,t' = \beta\,t = \sigma$. Thus,

$$\sigma' \in \sigma'|_{\mathcal{X}\cup\{g\in\mathcal{G},\bar{\mathcal{M}}[g]\cap S\neq\emptyset\}} = \sigma|_{\mathcal{X}\cup\{g\in\mathcal{G},\bar{\mathcal{M}}[g]\cap S\neq\emptyset\}} \subseteq \gamma_{\mathcal{R}}\,(r^\sharp) \subseteq \gamma_{\mathcal{R}}\,(r^{\sharp'})$$

Altogether, $t' \in \eta'_{\mathsf{mm}}\,[u', S, A']$ holds for all $t \in \eta^i\,[u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], \mathsf{signal}(s), u'), A']\!]_{\mathsf{mm}}\,\eta^i$ is subsumed by the value $\eta'_{\mathsf{mm}}\,[u', S, A']$ and that auxiliary property (a) also holds for the next approximation for the unknown $[u', S, A']$.

Next, we consider the side-effects of the corresponding right-hand-side functions for local traces t and t' as considered previously.

$$\begin{array}{rcl} \rho & = & \{[s, A'] \mapsto T\} \\ \rho^\sharp & = & \{[s, A'] \mapsto r^\sharp\} \end{array}$$

Let $\eta^\sharp\,[s, A'] = r^\sharp_\rho$ the value provided by $\mathcal{C}^\sharp$ for the unknown receiving the side-effect. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $r^\sharp \sqsubseteq r^\sharp_\rho$ hold. By definition:

$$\begin{array}{rl} & \eta'_{\mathsf{mm}}[s, A'] = \gamma_{s,A'}(\eta^\sharp\,[s, A']) \\ = & \{t \in \mathcal{T} \mid \mathsf{last}\,t = \mathsf{signal}(s), \alpha_{\mathcal{A}}\,t = A', \beta\,t \in \gamma_{\mathcal{R}}\,(r^\sharp_\rho)\} \end{array}$$

Consider a trace t' as above: Then $\mathsf{last}\,t' = \mathsf{signal}(s)$, $\alpha_{\mathcal{A}}\,t' = A'$, and and $\beta\,t' = \beta\,t \in \gamma_{\mathcal{R}}\,(r^\sharp) \subseteq \gamma_{\mathcal{R}}\,(r^\sharp_\rho)$ Altogether, $t' \in \eta'_{\mathsf{mm}}\,[s, A']$ holds for all $t \in \eta^i\,[u, S, A_0]$. Hence, all side-effects for $\mathsf{signal}(s)$ of $\mathcal{C}_{\mathsf{mm}}$ are accounted for in η'_{mm}, and the claim holds.

Lastly, consider the constraints corresponding to a call of **wait**. Consider an edge $(u, \mathsf{wait}(s), u') \in \mathcal{E}$ and digests A', A_0 and A_1 such that the resulting digest $A' \in [\![u, \mathsf{wait}(s)]\!]^\sharp_{\mathcal{A}}(A_0, A_1)$. We verify that

$$[\![([u, S, A_0], \mathsf{wait}(s), u'), A_1]\!]_{\mathsf{mm}}\,\eta^i \quad \subseteq \quad (\eta'_{\mathsf{mm}}, \eta'_{\mathsf{mm}}\,[u', S, A'])$$

We have

$$\begin{array}{l} [\![([u, S, A_0], \mathsf{wait}(s), u'), A_1]\!]_{\mathsf{mm}}\,\eta_{\mathsf{mm}} = \\ \quad \textbf{let } T = [\![(u, \mathsf{wait}(s), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u, S, A_0], \eta_{\mathsf{mm}}\,[s, A_1])\ \textbf{in} \\ \quad (\emptyset, T) \end{array}$$

$$\begin{array}{l} [\![[u, S, A_0], \mathsf{wait}(s), A_1]\!]^\sharp\eta^\sharp = \\ \quad \textbf{let } r^\sharp = \eta^\sharp\,[u, S, A_0]\ \textbf{in} \\ \quad \textbf{if } \eta^\sharp\,[s, A_1] = \bot\ \textbf{then} \\ \qquad (\emptyset, \bot) \\ \quad \textbf{else} \\ \qquad (\emptyset, r^\sharp) \end{array}$$

Let $\eta^\sharp\,[u, S, A_0] = r^\sharp$ and $\eta^\sharp\,[u', S, A'] = r^{\sharp'}$ the value provided by $\mathcal{C}^\sharp$ for the end point of the given control-flow edge and lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, we either have $\eta^\sharp\,[s, A_1] = \bot$ or $r^\sharp \sqsubseteq r^{\sharp'}$ holds. Then, by definition:

$$\begin{array}{rl} & \eta'_{\mathsf{mm}}[u', S, A'] = \gamma_{u',S,A'}(r^{\sharp'}) \\ = & \{t \in \mathcal{T} \mid \mathsf{loc}\,t = u', L_t = S, \alpha_{\mathcal{A}}\,t = A', \beta\,t \in \gamma_{\mathcal{R}}\,(r^{\sharp'})\} \end{array}$$

For every trace $t \in \eta^i\,[u, S, A_0]$, let $\beta\,t = \sigma$. By induction hypothesis, $\sigma \in \gamma_{\mathcal{R}}\,(r^\sharp)$ as well as $\sigma|_{\mathcal{X} \cup \{g \in \mathcal{G}, \mathcal{M}[g] \cap S \neq \emptyset\}} \subseteq \gamma_{\mathcal{R}}\,(r^\sharp)$. We distinguish between the case where the

resulting set of traces is T is empty, and the case where it is not. If T is empty, it is trivially subsumed by $\eta'_{\mathsf{mm}}\,[u', S, A']$. Let us thus consider the case where it is not empty: Consider a $t' \in T$ and a $t'' \in \eta^i\,[s, A_1]$ such that $\{t'\} = [\![(u, \mathsf{wait}(s), u')]\!]_{\mathcal{T}}(\{t\}, \{t''\})$. By induction hypothesis, we have:

$$t'' \in \eta^i\,[s, A_1] \quad \subseteq \quad \eta'_{\mathsf{mm}}\,[s, A_1] = \gamma_{s,A_1}(\eta^\sharp\,[s, A_1])$$

By the properties of γ_{s,A_1}, we have $\eta^\sharp\,[s, A_1] \neq \bot$. Thus, $r^\sharp \sqsubseteq r^{\sharp'}$ holds.

Consider again the local trace t': Then $\mathsf{loc}\,t' = u'$, $L_{t'} = S$, and $\alpha_{\mathcal{A}}\,t' = A'$. Let $\sigma' = \beta\,t'$. Then, $\forall y \in \mathcal{X} : \sigma'\,y = \sigma\,y$. For all globals protected by some currently held mutex $\{g \mid g \in \mathcal{G}, S \cap \bar{\mathcal{M}}[g] \neq \emptyset\}$, we also have $t'(g) = t(g)$ and thus $\sigma'\,g = \sigma\,g$. Altogether, we have

$$\sigma' \in \sigma'|_{\mathcal{X} \cup \{g \in \mathcal{G} \mid S \cap \bar{\mathcal{M}}[g] \neq \emptyset\}} = \sigma|_{\mathcal{X} \cup \{g \in \mathcal{G} \mid S \cap \bar{\mathcal{M}}[g] \neq \emptyset\}} \subseteq \gamma_{\mathcal{R}}\,(r^\sharp) \subseteq \gamma_{\mathcal{R}}\,(r^{\sharp'})$$

Altogether, $t' \in \eta'_{\mathsf{mm}}\,[u', S, A']$ holds for all $t \in \eta^i\,[u, S, A_0]$ and auxiliary property (a) also holds for the next approximation for the unknown $[u', S, A']$. As neither constraint causes any side-effects, the statement holds for edges corresponding to calls of $\mathsf{wait}(s)$.

This concludes the case distinction for the inductive step and, thus, the soundness proof for Mutex-Meet with Digests. □

6.2.2 Mutex-Meet with Joins

Intuitively, the key insight needed to establish soundness of this analysis is that side-effects in the abstract constraint system can be abandoned whenever the ego thread definitely did not write to any global from $\bar{\mathcal{G}}[a]$ since acquiring a. The latter holds whenever $W^\sharp \cap \bar{\mathcal{G}}[a] = \emptyset$ at the given $\mathsf{unlock}(a)$. By the same argument, whenever the ego thread has actually written a global from $\bar{\mathcal{G}}[a]$ since acquiring a, its thread *id* coincides with the thread *id* of the thread executing the last $\mathsf{unlock}(a)$ after a write to any global from $\bar{\mathcal{G}}[a]$.

More technically, the proof for this analysis proceeds in the following manner: After establishing a helpful auxiliary property about the k-th approximations to the least solution of the concrete constraint system, an instance of $\mathcal{A}$ tracking a rich abstraction of the history of the local trace is introduced. While for the other analyses, the analyses themselves made use of the same $\mathcal{A}$, in this case the analysis does not and instead splits only according to thread *id*s as before. The rich $\mathcal{A}$ is only used to argue that side-effect to unknowns associated with unaffected clusters can be abandoned in the abstract constraint system.

We refer to the constraint system of the analysis from Section 4.2.4 by $\mathcal{C}^\sharp$, and to the concrete constraint system introduced in the previous section by $\mathcal{C}_{\mathsf{mm}}$ once again. We further refer to the $\mathcal{A}$ to compute thread *id*s (see Fig. 2.14) by $\mathcal{A}_1$.

The following insight into the k-th approximation to the solution of the constraint system $\mathcal{C}_{\mathsf{mm}}$ computed during fixpoint iteration is required: For any trace t associated with an

unknown for a given program point, lockset, and digest in the k-th approximation η^k to the least solution η of $\mathcal{C}_{\mathsf{mm}}$, the subtrace of t ending in the last $\mathsf{unlock}(a)$ to immediately succeed a write to a global g protected by a, i.e., $g \in \bar{\mathcal{G}}[a]$, has been side-effected to all unknowns for clusters associated with a with an appropriate digest value. More formally, let $\mathsf{unlock\_after\_write}_a : \mathcal{T} \to \mathbf{E} \cup \{\bot\}$ be a function to extract, for a mutex a, the first $\mathsf{unlock}(a)$ action that immediately succeeds the last write to a global in $\bar{\mathcal{G}}[a]$. Then, we have:

Proposition 29. *Consider the k-th approximation η^k to the least solution η of $\mathcal{C}_{mm}$. Then, for all $u \in \mathcal{N}$, $S \subseteq \mathcal{M}$, and $A \in \mathcal{A}$ we have*

$$\begin{aligned}&\forall a \in \mathcal{M}, Q \in \mathcal{Q}_a : t \in \eta^k\,[u, S, A] \implies \\ &\quad \{t' \mid \mathit{unlock\_after\_write}_a\, t = (\bar{u}_i, \_, \_), t' = \downarrow_{\bar{u}_i}(t), \alpha_{\mathcal{A}}\, t' = A'\} \subseteq \eta^k\,[a, Q, A']\end{aligned}$$

Proof. The proof itself is obtained by fixpoint induction and relies on the fact that as soon as a trace ending in $\mathsf{unlock}(a)$ is created, it is immediately side-effected and that this either happened in the current iteration if t actually ends in such an unlock, or must otherwise have happened in an earlier iteration. We omit the details here. □

As hinted at before, the proof of the analysis from Section 4.2.4 abandons the generality of $\mathcal{A}$ and instead fixes a specific instance of $\mathcal{A}$ for the analyses. With the help of this digest, many portions of the proof from Section 6.2.1 carry over. It remains to show that side-effects can be abandoned in certain situations and that no global unknowns need to be consulted for *join-local* contributions. Use of such a digest in the analysis would make it overly precise and thus potentially also very slow. The full digest thus is used only as an intermediate step of the proof.

The modified instance of $\mathcal{A}$ then tracks, in addition to the thread *id* digests from Section 2.8 (and to the lockset digests as in the previous section), for each mutex the information of which abstract thread id (computed in the same manner as in Section 2.8) did the last unlock immediately succeeding a thread-local write to a global protected by that mutex, how many such unlocks have happened since program start, and — for currently held mutexes — whether any protected global has been written since the mutex was acquired. To this end, we choose

$$\mathcal{A} = (\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \times 2^{\mathcal{P}}) \times (\mathcal{M} \to (\mathbb{N}_0 \times (\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \times 2^{\mathcal{P}}) \times 2^{\mathcal{G}}))$$

and use the right-hand sides given in Fig. 6.3. The function $[\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}$ for a mutex a is then given by

$$[\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}((i, C), H)\,((i', C'), H') = \begin{cases} \emptyset & \neg \mathsf{may\_run}\,(i, C)\,(H'\,a)_2 \\ \left\{\left(\begin{array}{l}(i, C), \\ \{b \mapsto H\,b \mid b \in \mathcal{M}, (H\,b)_1 \geq (H'\,b)_1\} \\ \cup \{b \mapsto \bar{H}'\,b \mid b \in \mathcal{M}, (H\,b)_1 < (H'\,b)_1\}\end{array}\right)\right\} & \text{else} \end{cases}$$

$$\mathsf{init}^\sharp_{\mathcal{A}} = \{(((\mathsf{main}, \emptyset), \emptyset), \{a \mapsto (0, ((\mathsf{main}, \emptyset), \emptyset)) \mid a \in \mathcal{M}\})\}$$

$$[\![u, x{=}\mathsf{create}(u_1)]\!]^\sharp_{\mathcal{A}}\,((i, C), H) = \{((i, C \cup \{\langle u, u_1\rangle\}), H)\}$$

$$[\![u, g = x]\!]^\sharp_{\mathcal{A}}\,((i, C), H) = \{((i, C), \{a \mapsto ((H\,a)_1, (H\,a)_2, (H\,a)_3 \cup (\{g\} \cap \bar{\mathcal{G}}[a])) \mid a \in \mathcal{M}\})\}$$

$$[\![u, \mathsf{unlock}(a)]\!]^\sharp_{\mathcal{A}}((i, C), H) = \begin{cases} \{((i, C), H)\} & \text{if } (H\,a)_3 = \emptyset \\ \left\{ \begin{array}{l} ((i, C), (H \oplus \{a \mapsto \\ \quad ((H\,a)_1 + 1, (i, C), \emptyset)\})) \end{array} \right\} & \text{else} \end{cases}$$

$$[\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}\,((i, C), H) = \{((i, C), H)\} \quad \text{(other non-observing)}$$

$$\mathsf{new}^\sharp_{\mathcal{A}}\,u\,u_1\,(((d, s), C), H) = \textbf{let } (d', s') = (d, s) \circ \langle u, u_1\rangle \textbf{ in } \textbf{if } s' = \emptyset \wedge \langle u, u_1\rangle \in C \textbf{ then } (((d, \{\langle u, u_1\rangle\}), \emptyset), H) \textbf{ else } (((d', s'), \emptyset), H)$$

Figure 6.3: Modified instance of $\mathcal{A}$ tracking additional information on top of thread *id*s. The definition of $\circ$ is the one given in Fig. 2.14.

where $(\cdot)_k$ is shorthand for accessing the k-th component of a tuple and $\bar{H}$ denotes setting the 3-rd component of all $H\,b$ to $\emptyset$. This definition is also used for $[\![u, x = \mathsf{join}(x')]\!]^\sharp_{\mathcal{A}}$ and $[\![u, \mathsf{wait}(s)]\!]^\sharp_{\mathcal{A}}$. We remark that the may_run predicate here does not check for the current (i', C') of the thread performing the respective unlock, but instead refers to the (i, C) at the last thread-local write to a global protected by a.

The digest from Fig. 6.3 with the right-hand sides for observing actions as given above is an admissible digest. The key step in the proof is showing that the may_run check can soundly refer to the thread *id* at this earlier write. Intuitively, this holds as the predicate may_run is monotonic in some sense. If the thread unlocking the mutex can be started in the concrete, this also holds for any threads that thread has interacted with. We do not provide a formal argument of admissibility here.

The concretization for abstract values at unknowns of the form $[a, Q, ((i, C), H)]$ is defined such that it contains any local trace ending in an $\mathsf{unlock}(a)$ in which the last thread-local write to a global in $\bar{\mathcal{G}}[a]$ was by the thread with the thread *id* $(H\,a)_2$.

To define appropriate concretizations, we once again introduce a function β that maps a local trace to some information extracted from it that can then be related to the information computed by the analysis. To this end, we first define some additional helper functions: The function jl_joins : $\mathcal{T} \to 2^{\mathbf{E}}$ extracts from a local trace all calls to join that are *join-local*. Let $\mathsf{jl\_unlock\_after\_write}_a : \mathcal{T} \to \mathbf{E} \cup \{\bot\}$ be a function to extract, for a mutex a, the first *join-local* $\mathsf{unlock}(a)$ action that immediately succeeds the last *join-local* write to a global in $\bar{\mathcal{G}}[a]$. If mutex a is currently held by the ego thread in the argument trace, $\mathsf{jl\_unlock\_after\_write}_a$ only considers the subtrace that ends with

the ego-thread acquiring a.

The function β then has the following type:

$$\beta : \mathcal{T} \to (2^{\mathcal{V}_{\mathsf{tid}}} \times (\mathcal{M} \to (\mathcal{V}ars \to \mathcal{V})) \times 2^{\mathcal{G}} \times (\mathcal{V}ars \to \mathcal{V}))$$

We set:

$$\begin{array}{lcl}
\beta\, t & = & (J, L, W, r) \text{ where} \\
J & = & \{\sigma_i\, x' \mid ((i, u_i, \sigma_i), x{=}\mathsf{join}(x'), _) \in \mathsf{jl_joins}\ t\} \\
L & = & \{a \mapsto \mathsf{ex}\, t' \mid a \in \mathcal{M}, \mathsf{jl_unlock_after_write}_a\, t = \bot, t' = {\downarrow_{(u_0,0,\sigma_0)}}\ (t)\} \\
 & & \cup \{a \mapsto \mathsf{ex}\, t' \mid a \in \mathcal{M}, \mathsf{jl_unlock_after_write}_a\, t = (\bar{u}_i, _, _), t' = {\downarrow_{\bar{u}_i}}\ (t)\} \\
W & = & \bigcup_{a \in \mathcal{M}} \{g \mid g \in \bar{\mathcal{G}}[a], \mathsf{last_tl_write}_g\, t = (\bar{u}_i, _, _), \\
 & & \qquad ((\mathsf{last_tl_unlock}_a\, t = (\bar{u}_j, _, _) \wedge \bar{u}_j \leq \bar{u}_i) \vee \mathsf{last_tl_unlock}_a\, t = \bot)\} \\
r & = & \mathsf{ex}\, t
\end{array}$$

where the helper function $\mathsf{ex} : \mathcal{T} \to (\mathcal{V}ars \to \mathcal{V})$ is given by

$$\begin{array}{lcl}
\mathsf{ex}\, t & = & \{x \mapsto t(x) \mid x \in \mathcal{X}\} \cup \{g \mapsto 0 \mid g \in \mathcal{G}, \bot = \mathsf{last_write}_g\, t\} \\
 & & \cup\, \{g \mapsto \sigma_{j-1}\, x \mid g \in \mathcal{G}, ((j-1, u_{j-1}, \sigma_{j-1}), g = x, _) = \mathsf{last_write}_g\, t\}
\end{array}$$

The abstraction function β is used to specify concretization functions for the values of unknowns $[u, S, ((i, C), H)]$ for program points, currently held locksets, and digests as well as for the other unknowns, where $\mathcal{A}_1$ refers to the digests as considered in Section 2.8.

$$\begin{array}{lcl}
\gamma_{u,S,((i,C),H)}(J^\sharp, L^\sharp, W^\sharp, r^\sharp) & = & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u, L_t = S, \alpha_{\mathcal{A}_1}\, t = (i, C), \\
 & & \qquad (J, L, W, r) = \beta\, t, \\
 & & \qquad J^\sharp \subseteq_j J, L \sqsubseteq_t L^\sharp, W \subseteq W^\sharp, r \in \gamma_{\mathcal{R}}\, r^\sharp\} \\
\gamma_{i',((i,C),H)}(J^\sharp, L^\sharp, r^\sharp) & = & \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{return}, \alpha_{\mathcal{A}_1}\, t = (i, C), \mathsf{id}\, t = i', \\
 & & \qquad (J, L, W, r) = \beta\, t, \\
 & & \qquad J^\sharp \subseteq_j J, L \sqsubseteq_t L^\sharp, r \in \gamma_{\mathcal{R}}\, r^\sharp\} \\
\gamma_{a,((i,C),H)}(r^\sharp) & = & \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{unlock}(a), \\
 & & \qquad \mathsf{unlock_after_write}_a\, t = (\bar{u}_i, _, _), \\
 & & \qquad t' = {\downarrow_{\bar{u}_i}}\ (t), \alpha_{\mathcal{A}_1}\, t' = (H\, a)_2 \\
 & & \qquad (J, L, W, r) = \beta\, t', r \in \gamma_{\mathcal{R}}\, r^\sharp\} \\
 & & \cup \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{unlock}(a) \vee \mathsf{last}\, t = \mathsf{initMT}, \\
 & & \qquad \mathsf{unlock_after_write}_a\, t = \bot\} \\
\gamma_{s,((i,C),H)}(J^\sharp, L^\sharp, W^\sharp, r^\sharp) & = & \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{signal}(s), \alpha_{\mathcal{A}_1}\, t = (i, C), \\
 & & \qquad (J, L, W, r) = \beta\, t, \\
 & & \qquad J^\sharp \subseteq_j J, L \sqsubseteq_t L^\sharp, W \subseteq W^\sharp, r \in \gamma_{\mathcal{R}}\, r^\sharp\}
\end{array}$$

where $J^\sharp \subseteq_j J$ is shorthand for $\forall i^\sharp \in J^\sharp : \mathsf{unique}\, i^\sharp \implies \gamma_{\mathcal{V}^\sharp_{\mathbf{tid}},\mathcal{A}}\, i^\sharp \subseteq J$, i.e., checking that all abstract thread *id*s that are known to be must-joined and are unique are indeed joined in the local trace. By abuse of notation, let $L \sqsubseteq_t L^\sharp$ denote checking that

$$\forall a \in \mathcal{M} : (\mathsf{jl\_unlock\_after\_write}_a\, t = \mathsf{unlock\_after\_write}_a\, t \implies \forall Q \in \mathcal{Q}_a : L\, a \in \gamma_{\mathcal{R}}\, (L^\sharp(a, Q)))$$

This predicate thus checks that, if the last unlock of a mutex a immediately succeeding a write to a global protected by a is join-local to t, $L^\sharp\,(a, Q)$ soundly abstracts the value of locals and globals at the program point corresponding to that unlock for all $Q \in \mathcal{Q}_a$.

We remark that these concretization functions are monotonic. For a solution $\eta^\sharp$ of $\mathcal{C}^\sharp$ as considered in Section 4.2.4, we then construct a mapping η'_{mm} by:

$$\begin{array}{lcll}
\eta'_{\mathsf{mm}}[u, S, ((i,C),H)] & = & \gamma_{u,S,((i,C),H)}(\eta^\sharp\,[u, S, (i,C)]) & u \in \mathcal{N}, S \subseteq \mathcal{M}, \\
\eta'_{\mathsf{mm}}[a, Q, ((i,C),H)] & = & \gamma_{a,((i,C),H)}(\eta^\sharp\,[a, Q, (H\,a)_2]) & a \in \mathcal{M}, Q \in \mathcal{Q}_a \\
\eta'_{\mathsf{mm}}[i', ((i,C),H)] & = & \gamma_{i',((i,C),H)}(\eta^\sharp\,[(i,C)]) & i' \in \mathcal{V}_{\mathbf{tid}} \\
\eta'_{\mathsf{mm}}[s, ((i,C),H)] & = & \gamma_{s,((i,C),H)}(\eta^\sharp\,[s, (i,C)]) & s \in \mathcal{S}
\end{array}$$

for $((i,C),H) \in \mathcal{A}$. Altogether, correctness of $\mathcal{C}^\sharp$ follows from the following theorem:

Theorem 21. *Every solution of $\mathcal{C}^\sharp$ is sound w.r.t. the local trace semantics.*

Proof. Recall from Proposition 28, that the least solution of $\mathcal{C}_{\mathsf{mm}}$ (refined with an admissible digest) is sound w.r.t. the local trace semantics as specified by the constraint system $\mathcal{C}$. As the digest considered above is admissible, it suffices to prove that the mapping η'_{mm} as constructed above, is a solution of the constraint system $\mathcal{C}_{\mathsf{mm}}$. For that, we verify by fixpoint induction that for the k-th approximation η^k to the least solution η of $\mathcal{C}_{\mathsf{mm}}$, $\eta^k \subseteq \eta'_{\mathsf{mm}}$ holds.

As in the proof for Theorem 20, we once again require an auxiliary property stating that abstract values only speak about locals and protected globals (for unknowns associated with program points), respectively those globals that form a part of a given cluster (for unknowns associated with a mutex and a cluster). We call auxiliary property

(a) that

$$\begin{aligned}
\forall t \in \eta^k\,[u, S, ((i,C),H)] \quad & : \beta\, t = (J, L, W, r), (J^\sharp, L^\sharp, W^\sharp, r^\sharp) = \eta^\sharp\,[u, S, (i,C)] \implies \\
& \quad r|_{\mathcal{X} \cup \{g \in \mathcal{G}, \bar{\mathcal{M}}[g] \cap S \neq \emptyset\}} \subseteq \gamma_{\mathcal{R}}\, r^\sharp \wedge \forall a \in \mathcal{M} : \\
& \quad \mathsf{jl_unlock_after_write}_a\, t = \mathsf{unlock_after_write}_a\, t \implies \\
& \qquad \forall Q \in \mathcal{Q}_a : (L\,a)|_Q \in \gamma_{\mathcal{R}}\,(L^\sharp\,(a,Q)) \\
& \qquad\qquad (\text{for } u \in \mathcal{N}, S \subseteq \mathcal{M}, ((i,C),H) \in \mathcal{A})
\end{aligned}$$

$$\begin{aligned}
\forall t \in \eta^k\,[a, Q, ((i,C),H)] \quad & : \beta\, t = (J, L, W, r) \wedge \mathsf{unlock_after_write}_a\, t \neq \bot \implies \\
& \quad r|_Q \in \gamma_{\mathcal{R}}\,(\eta^\sharp\,[a, Q, (H\,a)_2]) \\
& \qquad\qquad (\text{for } a \in \mathcal{M}, Q \in \mathcal{Q}_a, ((i,C),H) \in \mathcal{A})
\end{aligned}$$

$$\begin{aligned}
\forall t \in \eta^k\,[i', ((i,C),H)] \quad & : \beta\, t = (J, L, r), (J^\sharp, L^\sharp, W^\sharp, r^\sharp) = \eta^\sharp\,[(i,C)] \implies \forall a \in \mathcal{M} : \\
& \quad \mathsf{jl_unlock_after_write}_a\, t = \mathsf{unlock_after_write}_a\, t \implies \\
& \qquad \forall Q \in \mathcal{Q}_a : (L\,a)|_Q \in \gamma_{\mathcal{R}}\,(L^\sharp\,(a,Q)) \\
& \qquad\qquad (\text{for } u \in \mathcal{N}, S \subseteq \mathcal{M}, ((i,C),H) \in \mathcal{A})
\end{aligned}$$

For the zero-th iteration η^k is $\emptyset$ everywhere, and thus auxiliary property (a) holds.

We first consider the constraints for **initialization**, the start point u_0 and the empty lockset, and verify that for all $A \in \mathsf{init}^\sharp_{\mathcal{A}}$:

$$(\emptyset, \{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\}) \quad \subseteq \quad (\eta'_{\mathsf{mm}}, \eta'_{\mathsf{mm}}\,[u_0, \emptyset, A])$$

As no side-effects are triggered, it suffices to check that $\{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\} \subseteq \eta'_{\mathsf{mm}}\,[u_0, \emptyset, A]$ holds.

$$\begin{aligned}
\mathsf{init}((i,C))^\sharp\,_ \quad = \quad & \textbf{let } L\,(a,Q) = [\![\{g \leftarrow 0 \mid g \in Q\}]\!]^\sharp_{\mathcal{R}} \top \textbf{ in} \\
& \textbf{let } r = [\![\mathsf{self} \leftarrow^\sharp \mathsf{single}\, i]\!]^\sharp_{\mathcal{R}} \top \textbf{ in} \\
& (\emptyset, (\emptyset, \{(a,Q) \mapsto L\,(a,Q) \mid a \in \mathcal{M}, Q \in \mathcal{Q}_a\}, \emptyset, r))
\end{aligned}$$

Let $\eta^\sharp\,[u_0, \emptyset, (i,C)] = (J^\sharp, L^\sharp, W^\sharp, r^\sharp)$ the value provided by $\eta^\sharp$ for the start point and the empty lockset. Since $\eta^\sharp$ is a solution of $C^\sharp$, we have

$$\begin{aligned}
\emptyset \quad & \supseteq \quad J^\sharp \\
\{(a,Q) \mapsto [\![\{g \leftarrow 0 \mid g \in Q\}]\!]^\sharp_{\mathcal{R}} \top \mid a \in \mathcal{M}, Q \in \mathcal{Q}_a\} \quad & \sqsubseteq \quad L^\sharp \\
\emptyset \quad & \subseteq \quad W^\sharp \\
[\![\mathsf{self} \leftarrow^\sharp \mathsf{single}\, i]\!]^\sharp_{\mathcal{R}} \top \quad & \sqsubseteq \quad r^\sharp
\end{aligned}$$

By definition,

$$\begin{aligned}
& \eta'_{\mathsf{mm}}\,[u_0, \emptyset, ((i,C),H)] = \gamma_{u_0,\emptyset,((i,C),H)}(\eta^\sharp\,[u_0, \emptyset, (i,C)]) \\
= \quad & \gamma_{u_0,\emptyset,((i,C),H)}(J^\sharp, L^\sharp, W^\sharp, r^\sharp) \\
= \quad & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u_0, L_t = \emptyset, \alpha_{\mathcal{A}_1}\, t = (i,C), (J, L, W, r) = \beta\, t, \\
& \qquad J^\sharp \subseteq_j J, L \sqsubseteq_t L^\sharp, W \subseteq W^\sharp, r \in \gamma_{\mathcal{R}}\, r^\sharp\}
\end{aligned}$$

Consider a trace t from $\{t \mid t \in \mathsf{init}, ((i,C),H) = \alpha_{\mathcal{A}}(t)\}$. Then, $\mathsf{loc}\, t = u_0$, $L_t = \emptyset$, and $\alpha_{\mathcal{A}_1}\, t = (i,C)$ all hold vacuously. Let $\beta\, t = (J,L,W,r)$. By construction, we have $J = \emptyset$, and since $\emptyset \supseteq J^\sharp$ implies $J^\sharp = \emptyset$, thus $\bigcup\{\gamma_{\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}}\, i^\sharp \mid i^\sharp \in J^\sharp\} = \emptyset \subseteq \emptyset = J$ and thus $J^\sharp \subseteq_j J$ holds. For any mutex a, we have $\mathsf{jl\_unlock\_after\_write}_a\, t = \mathsf{unlock\_after\_write}_a\, t = \bot$ and for all $Q \in \mathcal{Q}_a$,

$$\begin{array}{rcl} La & = & (\{x \mapsto t(x) \mid x \in \mathcal{X}\} \cup \{g \mapsto 0 \mid g \in \mathcal{G}\}) \\ & \in & (\{x \mapsto t(x) \mid x \in \mathcal{X}\} \cup \{g \mapsto 0 \mid g \in \mathcal{G}\})|_Q \\ & \subseteq & \gamma_{\mathcal{R}}([\![\{g \leftarrow 0 \mid g \in Q\}]\!]^\sharp_{\mathcal{R}} \top) \subseteq (\gamma_{\mathcal{R}} L^\sharp(a,Q)) \end{array}$$

and thus $L \sqsubseteq_t L^\sharp$ holds as does the auxiliary property (a) w.r.t. the requirements on L. Further, we have $W = \emptyset \subseteq W^\sharp$. We remark that, by construction, $r\,\mathsf{self} = i_0$, and by Eq. (2.13) we have $i_0 \in \gamma_{\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}}\, i$. Then, by soundness of the operations on the base domain and the relational domain, and the property of single we obtain

$$r \in r|_{\mathcal{X}} \subseteq \gamma_{\mathcal{R}}\left([\![\mathsf{self} \leftarrow^\sharp \mathsf{single}\, i]\!]^\sharp_{\mathcal{R}} \top\right) \subseteq \gamma_{\mathcal{R}}(r^\sharp)$$

Altogether $t \in \eta'_{\mathsf{mm}}\,[u_0, \emptyset, A]$ holds for all $t \in \{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\}$ and the auxiliary property (a) also holds for the next iteration.

For this analysis, the right-hand sides for **reading from a global**, **computations on locals**, **guards** take the same form as for the analysis discussed in Section 6.2.1 and the soundness argument carries over mostly unchanged (also for the auxiliary property (a) for L as neither of these actions are observing). We thus do not provide the detailed proof for these right-hand sides here.

Next, consider the constraints corresponding to **locking** a mutex a. Consider an edge $(u, \mathsf{lock}(a), u') \in \mathcal{E}$ and digests $A' = ((i,C),H')$, $A_0 = ((i,C),H_0)$, and $A_1 = ((i_1,C_1),H_1)$ such that $((i,C),H') \in [\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}(((i,C),H_0),((i_1,C_1),H_1))$. We verify that

$$[\![([u,S,((i,C),H_0)], \mathsf{lock}(a), u'), ((i_1,C_1),H_1)]\!]_{\mathsf{mm}}\, \eta^k \subseteq (\eta'_{\mathsf{mm}}, \eta'_{\mathsf{mm}}\,[u', S \cup \{a\}, ((i,C),H')])$$

Let us call $((H_1)\,a)_2 = (\bar{i}_1, \bar{C}_1)$. We have

$$\begin{array}{l} [\![([u,S,A_0], \mathsf{lock}(a), u'), A_1]\!]_{\mathsf{mm}}\, \eta_{\mathsf{mm}} = \\ \quad \textbf{let } T_1 = \bigcap\{\eta_{\mathsf{mm}}\,[a,Q,A_1] \mid Q \in \mathcal{Q}_a\} \textbf{ in} \\ \quad \textbf{let } T = [\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u,S,A_0], T_1) \textbf{ in} \\ \quad (\emptyset, T) \end{array}$$

$$
\begin{array}{l}
[\![[u,S,(i,C)],\mathsf{lock}(a),(\bar{i}_1,\bar{C}_1)]\!]^\sharp\eta^\sharp = \\
\quad \textbf{let } (J^\sharp,L^\sharp,W^\sharp,r^\sharp) = \eta^\sharp\,[u,S,(i,C)] \textbf{ in} \\
\quad \textbf{let } r_m^\sharp = \textbf{if } \mathsf{unique}\,\bar{i}_1 \wedge (i=\bar{i}_1 \vee \bar{i}_1 \in J^\sharp) \textbf{ then} \\
\qquad\qquad \bigsqcap_{Q\in\mathcal{Q}_a} L^\sharp\,(a,Q) \\
\qquad\quad \textbf{else} \\
\qquad\qquad \left(\bigsqcap_{Q\in\mathcal{Q}_a} \eta^\sharp\,[a,Q,(\bar{i}_1,\bar{C}_1)]\right) \sqcup \left(\bigsqcap_{Q\in\mathcal{Q}_a} L^\sharp\,(a,Q)\right) \\
\quad \textbf{in} \\
\quad \textbf{let } r^{\sharp\prime\prime} = r^\sharp \sqcap r_m^\sharp \textbf{ in} \\
\quad \left(\emptyset,\left(J^\sharp,L^\sharp,W^\sharp,r^{\sharp\prime\prime}\right)\right)
\end{array}
$$

Let $\eta^\sharp\,[u,S,(i,C)] = (J^\sharp,L^\sharp,W^\sharp,r^\sharp)$ and $\eta^\sharp\,[u',S\cup\{a\},(i,C)] = (J^{\sharp\prime},L^{\sharp\prime},W^{\sharp\prime},r^{\sharp\prime})$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge and the resulting lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $J^\sharp \sqsubseteq J^{\sharp\prime}$, $L^\sharp \sqsubseteq L^{\sharp\prime}$, $W^\sharp \subseteq W^{\sharp\prime}$, and $r^{\sharp\prime\prime} \sqsubseteq r^{\sharp\prime}$ all hold. Then, by definition:

$$
\begin{array}{rl}
& \eta'_{\mathsf{mm}}[u',S\cup\{a\},((i,C),H')] = \gamma_{u',S\cup\{a\},((i,C),H')}((J^{\sharp\prime},L^{\sharp\prime},W^{\sharp\prime},r^{\sharp\prime})) \\
= & \{t\in\mathcal{T} \mid \mathsf{loc}\,t = u', L_t = S\cup\{a\}, \alpha_{\mathcal{A}_1}\,t = (i,C), (J,L,W,r) = \beta\,t, \\
& \qquad J^{\sharp\prime} \subseteq_j J, L \sqsubseteq_t L^{\sharp\prime}, W \subseteq W^{\sharp\prime}, r \in \gamma_{\mathcal{R}}\,r^{\sharp\prime}\}
\end{array}
$$

For every trace $t \in \eta^k\,[u,S,A_0]$, let $\beta\,t = (J,L,W,r)$. By induction hypothesis, $J^\sharp \subseteq_j J$, $L \sqsubseteq_t L^\sharp$, $W \subseteq W^\sharp$, and $r \in \gamma_{\mathcal{R}}\,(r^\sharp)$. For any trace $t_1 \in \bigcap\{\eta^k\,[a,Q,((i_1,C_1),H_1)] \mid Q\in\mathcal{Q}_a\}$ for which we have $[\![(u,\mathsf{lock}(a),u')]\!]_{\mathcal{T}}(\{t\},\{t_1\}) \neq \emptyset$ let $(\_,\_,\_,r_1) = \beta\,t_1$. We have by induction hypothesis that for every $Q\in\mathcal{Q}_a$, $t_1 \in \eta'_{\mathsf{mm}}\,[a,Q,((i_1,C_1),H_1)]$. By definition, we have

$$
\begin{array}{rl}
& \eta'_{\mathsf{mm}}[a,Q,((i_1,C_1),H_1)] \\
= & \gamma_{a,((i,C),H)}(\eta^\sharp\,[a,Q,(H_1\,a)_2]) = \gamma_{a,((i,C),H)}(\eta^\sharp\,[a,Q,(\bar{i}_1,\bar{C}_1)]) \\
= & \{t\in\mathcal{T} \mid \mathsf{last}\,t = \mathsf{unlock}(a), \mathsf{unlock\_after\_write}_a\,t = (\bar{u}_i,\_,\_), \\
& \qquad t' = \downarrow_{\bar{u}_i}(t), \alpha_{\mathcal{A}_1}\,t' = (H\,a)_2 \\
& \qquad (J,L,W,r) = \beta\,t', r \in \gamma_{\mathcal{R}}\,(\eta^\sharp\,[a,Q,(\bar{i}_1,\bar{C}_1)])\} \\
& \cup\{t\in\mathcal{T} \mid \mathsf{last}\,t = \mathsf{unlock}(a) \vee \mathsf{last}\,t = \mathsf{initMT}, \mathsf{unlock\_after\_write}_a\,t = \bot\}
\end{array}
$$

Consider a cluster $Q\in\mathcal{Q}_a$ and let $\eta^\sharp[a,Q,(\bar{i}_1,\bar{C}_1)] = r_1^\sharp$. By induction hypothesis, we thus have

(1) either

$$t_1 \in \{t''\in\mathcal{T} \mid \mathsf{last}\,t'' = \mathsf{unlock}(a) \vee \mathsf{last}\,t'' = \mathsf{initMT}, \mathsf{unlock\_after\_write}_a\,t'' = \bot\}$$

Then $\mathsf{jl\_unlock\_after\_write}_a\,t = \bot$ as well, and by induction hypothesis, there is an r_0 (namely the one extracted from the local trace ending in $(u_0,0,\sigma_0)$) such that $r_0 \in \gamma_R\,(L^\sharp\,(a,Q))$ for all $Q\in\mathcal{Q}_a$. r_1 coincides with r_0 on the values of globals, and we thus get by auxiliary property (a):

$$r_1|_{\bar{\mathcal{G}}[a]} \subseteq r_1|_Q = r_0|_Q \subseteq \gamma_R\,(L^\sharp\,(a,Q))$$

(2) Otherwise, we have

$$t_1 \in \{t'' \in \mathcal{T} \mid \mathsf{last}\, t'' = \mathsf{unlock}(a), \mathsf{unlock\_after\_write}_a\, t'' = (\bar{u}_i, \_, \_), \\ t''' =\downarrow_{\bar{u}_i} (t''), \alpha_{\mathcal{A}_1}\, t''' = (\bar{i}_1, \bar{C}_1), (J_1'''1, L_1''', W_1''', r_1''') = \beta\, t''', r_1''' \in \gamma_{\mathcal{R}}\, r_1^\sharp\}$$

As r_1''' and r_1 coincide on the values of all protected globals, we obtain (once again using the auxiliary property (a))

$$r_1|_{\bar{\mathcal{G}}[a]} \subseteq r_1|_Q = r_1'''|_Q \subseteq \gamma_R\,(r_1^\sharp)$$

Let $t' \in [\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}(\{t\}, \{t_1\})$. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S \cup \{a\}$, and $\alpha_{\mathcal{A}_1}\, t' = \alpha_{\mathcal{A}_1}\, t = (i, C)$. Let $(J', L', W', r') = \beta\, t'$.

- First, consider the r component. We first observe that once more $\forall x \in \mathcal{X}, r'\, x = r\, x$. Let us first consider the case where $\mathsf{unique}\, \bar{i}_1 \wedge (i = \bar{i}_1 \vee \bar{i}_1 \in J^\sharp)$ does not hold: For all globals protected by a, we then have either
  - $S \cap \bar{\mathcal{M}}[g] \neq \emptyset$, in which case $r'\, g = r\, g = r_1\, g$, or
  - $S \cap \bar{\mathcal{M}}[g] = \emptyset$, in which case $r'\, g = r_1\, g$.

  Thus,

$$\begin{array}{rcl} r' & \in & (r')|_{\mathcal{X} \cup \{g \in \mathcal{G}, \bar{\mathcal{M}}[g] \cap (S \cup \{a\}) \neq \emptyset\}} \\ & \subseteq & r|_{\mathcal{X} \cup \{g \in \mathcal{G}, \bar{\mathcal{M}}[g] \cap S \neq \emptyset\}} \cap r_1|_{\bar{\mathcal{G}}[a]} \\ & \subseteq & \gamma_{\mathcal{R}}\,(r^\sharp) \cap (\bigcap\{\gamma_{\mathcal{R}}(\eta^\sharp\,[a, Q, (\bar{i}_1, \bar{C}_1)]) \mid Q \in \mathcal{Q}_a\} \\ & & \qquad \cup (\bigcap\{\gamma_{\mathcal{R}}(L^\sharp\,(a, Q)) \mid Q \in \mathcal{Q}_a\}) \\ & \subseteq & \gamma_{\mathcal{R}}\left(r^\sharp \sqcap \left((\prod_{Q \in \mathcal{Q}_a} (\eta^\sharp\,[a, Q, (\bar{i}_1, \bar{C}_1)])) \sqcup \left(\prod_{Q \in \mathcal{Q}_a} L^\sharp\,(a, Q)\right)\right)\right) \\ & = & \gamma_{\mathcal{R}}\, r^{\sharp''} \subseteq \gamma_{\mathcal{R}}\, r^{\sharp'} \end{array}$$

  where we use that the $\sqcap$ operation in $\mathcal{R}$ is sound w.r.t. the intersection of concretizations. Also, auxiliary property (a) holds for the next approximation for the r component.

  Now for the case where $\mathsf{unique}\, \bar{i}_1 \wedge (i = \bar{i}_1 \vee \bar{i}_1 \in J^\sharp)$ holds:
  - $\mathsf{unique}\, \bar{i}_1 \wedge i = \bar{i}_1$ implies that it was the ego thread of t itself that performed the last write to a global protected by a and the following unlock or that there was no such write. As a consequence, we have $r'\, g = r\, g$ for $g \in \bar{\mathcal{G}}[a]$. By induction hypothesis and the auxiliary property (a), we also have for all clusters $Q \in \mathcal{Q}_a$, $L\,(a, Q)|_Q = r|_Q \in \gamma_R(L^\sharp\,(a, Q))$.
  - In the other case, we have $\gamma_{\mathcal{V}^\sharp_{\mathbf{tid}, \mathcal{A}}}\, \bar{i} \subseteq J$, and thus the last write to a global protected by a and the following unlock is join-local to t. The same argument holds here.

Thus,

$$\begin{array}{rcl} r' & \in & (r')|_{\mathcal{X}\cup\{g\in\mathcal{G},\mathcal{M}[g]\cap(S\cup\{a\})\neq\emptyset\}} \\ & \subseteq & r|_{\mathcal{X}\cup\{g\in\mathcal{G},\mathcal{M}[g]\cap S\neq\emptyset\}} \cap r|_{\mathcal{G}[a]} \\ & \subseteq & \gamma_{\mathcal{R}}\left(r^\sharp\right) \cap \left(\bigcap\{\gamma_{\mathcal{R}}(L^\sharp\,(a,Q)) \mid Q\in\mathcal{Q}_a\}\right) \\ & \subseteq & \gamma_{\mathcal{R}}\left(r^\sharp \sqcap \left(\prod_{Q\in\mathcal{Q}_a} L^\sharp\,(a,Q)\right)\right) \\ & = & \gamma_{\mathcal{R}}\, r^{\sharp''} \subseteq \gamma_{\mathcal{R}}\, r^{\sharp'} \end{array}$$

where we use that the $\sqcap$ operation in $\mathcal{R}$ is sound w.r.t. the intersection of concretizations. Also, auxiliary property (a) holds for the next approximation for the r component.

- Next for J: As t' and t coincide on their join-local parts (save for the new edge labeled $\mathsf{lock}(a)$ appearing in t' only), $\mathsf{jl\_joins}\ t = \mathsf{jl\_joins}\ t'$ holds. Thus, we have $J^\sharp \subseteq_j J = J'$, and as $J^\sharp \sqsubseteq J^{\sharp'}$, also $J^{\sharp'} \subseteq_j J'$.

- By the same argument, we have that for all $a \in \mathcal{M}$ for which the last unlock directly following a write to a global protected by a is join-local to t', i.e., $\mathsf{jl\_unlock\_after\_write}_a\, t' = \mathsf{unlock\_after\_write}_a\, t'$, that unlock was also join-local to t, and $L'\,a = L\,a$. Thus, for all such mutexes, we have $L'\,a = L\,a \in \gamma_{\mathcal{R}}\, L^\sharp\,(a,Q) \subseteq \gamma_{\mathcal{R}}\, L^{\sharp'}\,(a,Q)$ for all $Q \in \mathcal{Q}_a$. Thus, auxiliary property (a) also holds w.r.t. the requirements on L.

- Finally, using the same argument once more, we have $W' = W \subseteq W^\sharp \subseteq W^{\sharp'}$.

Thus, $t' \in \eta'_{\mathsf{mm}}\,[u', S\cup\{a\}, A']$ holds for all $t \in \eta^k\,[u,S,A_0]$ and $t_1 \in \bigcap\{\eta^k\,[a,Q,A_1] \mid Q \in \mathcal{Q}_a\}$. We conclude that the return value of $[\![([u,S,A_0],\mathsf{lock}(a),u'),A_1]\!]_{\mathsf{mm}}\,\eta^k$ is subsumed by the value $\eta'_{\mathsf{mm}}\,[u', S\cup\{a\}, A']$ and that auxiliary property (a) holds for the next approximation as well for the unknown $[u', S\cup\{a\}, A']$. Since the constraint causes no side-effects, the claim holds.

Next, for constraints corresponding to **unlock**. Consider an edge $(u,\mathsf{unlock}(a),u') \in \mathcal{E}$ and appropriate digests $A' = ((i,C),H')$ and $A_0 = ((i,C),H_0)$ such that $((i,C),H') \in [\![u,\mathsf{unlock}(a)]\!]^\sharp_{\mathcal{A}}((i,C),H_0)$. We verify that

$$[\![([u,S,((i,C),H_0)],\mathsf{unlock}(a),u'),((i,C),H')]\!]_{\mathsf{mm}}\,\eta^k \subseteq (\eta'_{\mathsf{mm}},\eta'_{\mathsf{mm}}\,[u',S\setminus\{a\},((i,C),H')])$$

We have

$$\begin{array}{l} [\![([u,S,((i,C),H_0)],\mathsf{unlock}(a),u'),((i,C),H')]\!]_{\mathsf{mm}}\,\eta_{\mathsf{mm}} = \\ \quad \textbf{let}\ T = [\![(u,\mathsf{unlock}(a),u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u,S,((i,C),H_0)])\ \textbf{in} \\ \quad \textbf{let}\ \rho = \{[a,Q,((i,C),H')] \mapsto T \mid Q\in\mathcal{Q}_a\}\ \textbf{in} \\ \quad (\rho,T) \end{array}$$

$$
\begin{array}{l}
[\![[u,S,(i,C)],\mathsf{unlock}(a),(i,C)]\!]^\sharp\eta^\sharp = \\
\quad \textbf{let } (J^\sharp,L^\sharp,W^\sharp,r^\sharp) = \eta^\sharp\,[u,S,(i,C)] \textbf{ in} \\
\quad \textbf{let } (L^{\sharp''},\rho^\sharp) = \textbf{if } \bar{\mathcal{G}}[a]\cap W^\sharp = \emptyset \textbf{ then} \\
\qquad\quad (L^\sharp,\emptyset) \\
\qquad \textbf{else} \\
\qquad\quad (L^\sharp \oplus \{(a,Q)\mapsto r^\sharp\big|_Q \mid Q\in\mathcal{Q}_a\},\{[a,Q,(i,C))]\mapsto r^\sharp\big|_Q \mid Q\in\mathcal{Q}_a\}) \\
\quad \textbf{in} \\
\quad \textbf{let } r^{\sharp''} = r^\sharp\big|_{\mathcal{X}\cup\bigcup\{\bar{\mathcal{G}}[a']\mid a'\in(S\setminus a)\}} \textbf{ in} \\
\quad \textbf{let } W^{\sharp''} = \{g \mid g\in W^\sharp, \bar{\mathcal{M}}[g]\cap S\setminus\{a\}\neq\emptyset\} \\
\quad \textbf{in } (\rho^\sharp,(J^\sharp,L^{\sharp''},W^{\sharp\prime\prime},r^{\sharp''}))
\end{array}
$$

Let $\eta^\sharp\,[u,S,(i,C)] = (J^\sharp,L^\sharp,W^\sharp,r^\sharp)$ and $\eta^\sharp\,[u',S\setminus\{a\},(i,C)] = (J^{\sharp'},L^{\sharp'},W^{\sharp'},r^{\sharp'})$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge and the resulting lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $J^\sharp \sqsubseteq J^{\sharp'}$, $L^{\sharp''} \sqsubseteq L^{\sharp'}$, $W^{\sharp''} \subseteq W^{\sharp'}$, and $r^{\sharp''} \sqsubseteq r^{\sharp'}$ all hold. Then, by definition:

$$
\begin{array}{rl}
& \eta'_{\mathsf{mm}}[u',S\setminus\{a\},((i,C),H')] = \gamma_{u',S\setminus\{a\},((i,C),H')}((J^{\sharp'},L^{\sharp'},W^{\sharp'},r^{\sharp'})) \\
= & \{t\in\mathcal{T} \mid \mathsf{loc}\,t = u', L_t = S\setminus\{a\}, \alpha_{\mathcal{A}_1}\,t = (i,C), (J,L,W,r) = \beta\,t, \\
& \qquad J^{\sharp'} \subseteq_j J, L \sqsubseteq_t L^{\sharp'}, W\subseteq W^{\sharp'}, r\in\gamma_{\mathcal{R}}\,r^{\sharp'}\}
\end{array}
$$

For every trace $t\in\eta^k\,[u,S,A_0]$, let $\beta\,t = (J,L,W,r)$. By induction hypothesis, $J^\sharp \subseteq_j J$, $L\sqsubseteq_t L^\sharp$, $W\subseteq W^\sharp$, and $r\in\gamma_{\mathcal{R}}\,(r^\sharp)$. Let $t'\in[\![(u,\mathsf{unlock}(a),u')]\!]_{\mathcal{T}}\{t\}$. Then $\mathsf{loc}\,t' = u'$, $L_{t'} = S\setminus\{a\}$, and $\alpha_{\mathcal{A}_1}\,t' = \alpha_{\mathcal{A}_1}\,t = (i,C)$. Let $(J',L',W',r') = \beta\,t'$.

- As t' and t coincide on their join-local parts (save for the new edge labeled $\mathsf{unlock}(a)$ appearing in t' only), $\mathsf{jl\_joins}\ t = \mathsf{jl\_joins}\ t'$ holds. Thus, we have $J^\sharp \subseteq_j J = J'$, and as $J^\sharp \sqsubseteq J^{\sharp'}$, also $J^{\sharp'} \subseteq_j J'$.

- For L and L', we have

$$
\begin{array}{rl}
L' = & \{b\mapsto \mathsf{ex}\,t'' \mid b\in\mathcal{M}, \mathsf{jl\_unlock\_after\_write}_b\,t' = \bot, t'' = {\downarrow_{(u_0,0,\sigma_0)}}\,(t')\} \\
& \cup\{b\mapsto \mathsf{ex}\,t'' \mid b\in\mathcal{M}, \mathsf{jl\_unlock\_after\_write}_b\,t' = (\bar{u}_i,\_,\_), t'' = {\downarrow_{\bar{u}_i}}\,(t')\} \\
= & L\oplus(\{a\mapsto \mathsf{ex}\,t'' \mid \mathsf{jl\_unlock\_after\_write}_a\,t' = \bot, t'' = {\downarrow_{(u_0,0,\sigma_0)}}\,(t')\} \\
& \cup\{a\mapsto \mathsf{ex}\,t'' \mid \mathsf{jl\_unlock\_after\_write}_a\,t' = (\bar{u}_i,\_,\_), t'' = {\downarrow_{\bar{u}_i}}\,(t')\})
\end{array}
$$

  For all mutexes $b\in(\mathcal{M}\setminus a)$, the last unlock directly following a write to a global protected by it is the same in t and t', and thus we have $\mathsf{jl\_unlock\_after\_write}_b\,t' = \mathsf{unlock\_after\_write}_b\,t'$, only whenever the relationship $\mathsf{jl\_unlock\_after\_write}_b\,t = \mathsf{unlock\_after\_write}_b\,t$ held before. In this case, we have

$$
L\,b\in(L\,b)|_Q \subseteq \gamma_{\mathcal{R}}(L^\sharp\,(b,Q)) \subseteq \gamma_{\mathcal{R}}(L^{\sharp''}\,(b,Q)) \subseteq \gamma_{\mathcal{R}}(L^{\sharp'}\,(b,Q))
$$

  for all $Q\in\mathcal{Q}_b$. Thus, auxiliary property (a) also continues to hold for any such $L\,b$. It thus remains to consider the mutex a. Let us first consider the case where

$W^\sharp \cap \bar{\mathcal{G}}[a] = \emptyset$ and thus $L^{\sharp''}(a, Q) = L^\sharp(a, Q)$ for all $Q \in \mathcal{Q}_a$. Then, as by induction hypothesis $W \subseteq W^\sharp$, we also have $W \cap \bar{\mathcal{G}}[a] = \emptyset$. Recall that

$$\begin{array}{rcl} W & = & \bigcup_{a \in \mathcal{M}} \{g \mid g \in \bar{\mathcal{G}}[a], \mathsf{last\_tl\_write}_g\, t = (\bar{u}_i, \_, \_), \\ & & ((\mathsf{last\_tl\_unlock}_a\, t = (\bar{u}_j, \_, \_) \wedge \bar{u}_j \leq \bar{u}_i) \vee \mathsf{last\_tl\_unlock}_a\, t = \bot)\} \end{array}$$

Thus, no join-local write has happened since the last join-local unlock, and the same argument as given above holds.

For the other case, if $\mathsf{jl\_unlock\_after\_write}_a\, t' \neq \mathsf{unlock\_after\_write}_a\, t'$ the property holds trivially as does auxiliary property (a).

Consider now the case where $\mathsf{jl\_unlock\_after\_write}_a\, t' = \mathsf{unlock\_after\_write}_a\, t'$. We then need to show that $L'\, a \in \gamma_{\mathcal{R}}\, L^{\sharp'}(a, Q)$ for $Q \in \mathcal{Q}_a$. As $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, we have $r^\sharp\big|_Q = L^{\sharp''}(a, Q) \sqsubseteq L^{\sharp'}(a, Q)$. We have $L'\, a = \mathsf{ex}\, t''$ where $t'' = {\downarrow_{(u_0, 0, \sigma_0)}}\, (t')$ if $\mathsf{jl\_unlock\_after\_write}_a\, t' = \bot$ and $t'' = {\downarrow_{\bar{u}_i}}\, (t')$ if $\mathsf{jl\_unlock\_after\_write}_a\, t' = (\bar{u}_i, \_, \_)$. As the last unlock of a immediately following a write to a global protected by a is join-local, we have $(\mathsf{ex}\, t'')\, g = (\mathsf{ex}\, t)\, g = (\mathsf{ex}\, t')\, g = r\, g$ for all $g \in \bar{\mathcal{G}}[a]$. By induction hypothesis, we have $r \in \gamma_{\mathcal{R}}\, r^\sharp$, and thus $(\mathsf{ex}\, t'') \in \gamma_{\mathcal{R}}(r^\sharp\big|_Q)$ for $Q \in \mathcal{Q}_a$ as $Q \subseteq \bar{\mathcal{G}}[a]$ holds for all such Q. All together, in this case, we have

$$L'\, a \in L'\, a\big|_Q \subseteq \gamma_{\mathcal{R}}\, (r^\sharp\Big|_Q) \subseteq \gamma_{\mathcal{R}}\, (L^{\sharp''}(a, Q)) = \gamma_{\mathcal{R}}\, (L^{\sharp'}(a, Q))$$

for all $Q \in \mathcal{Q}_a$. As a consequence, we obtain $L' \sqsubseteq_t L^{\sharp'}$ and auxiliary property (a) also holds for the next approximation.

- Next for W' and $W^{\sharp'}$: We have

$$\begin{array}{rcl} W' & = & \bigcup_{b \in \mathcal{M}} \{g \mid g \in \bar{\mathcal{G}}[b], \mathsf{last\_tl\_write}_g\, t' = (\bar{u}_i, \_, \_), \\ & & ((\mathsf{last\_tl\_unlock}_b\, t' = (\bar{u}_j, \_, \_) \wedge \bar{u}_j \leq \bar{u}_i) \vee \mathsf{last\_tl\_unlock}_b\, t' = \bot)\} \\ & = & (\bigcup_{b \in \mathcal{M}} \{g \mid g \in \bar{\mathcal{G}}[b], \mathsf{last\_tl\_write}_g\, t' = (\bar{u}_i, \_, \_), \\ & & ((\mathsf{last\_tl\_unlock}_b\, t' = (\bar{u}_j, \_, \_) \wedge \bar{u}_j \leq \bar{u}_i) \vee \mathsf{last\_tl\_unlock}_b\, t' = \bot)\}) \\ & & \setminus \{g \mid g \in \mathcal{G}, \bar{\mathcal{M}}[g] \cap S' = \emptyset\} \\ & \subseteq & W \setminus \{g \mid g \in \mathcal{G}, \bar{\mathcal{M}}[g] \cap S' = \emptyset\} \\ & \subseteq & W^\sharp \setminus \{g \mid g \in \mathcal{G}, \bar{\mathcal{M}}[g] \cap S' = \emptyset\} \\ & \subseteq & W^\sharp \setminus \{g \mid g \in W^\sharp, \bar{\mathcal{M}}[g] \cap S' = \emptyset\} \\ & = & \{g \mid g \in W^\sharp, \bar{\mathcal{M}}[g] \cap (S \setminus \{a\}) \neq \emptyset\} \\ & = & W^{\sharp''} \subseteq W^{\sharp'} \end{array}$$

The second equality exploits that a global is only written when its protecting mutexes are held, and if none of these mutexes are held by the ego thread currently, they all must have been unlocked at least once since the last write.

- Lastly, we have from property (a) that $r|_{\mathcal{X} \cup \bigcup \{\bar{\mathcal{G}}[a'] \mid a' \in S\}} \in \gamma_{\mathcal{R}}(r^\sharp)$ and

$$r' = r \in \gamma_{\mathcal{R}}(r^\sharp) \subseteq \gamma_{\mathcal{R}} \left(r^\sharp\Big|_{\mathcal{X} \cup \bigcup \{\bar{\mathcal{G}}[a'] \mid a' \in (S \setminus a)\}} \right) = \gamma_{\mathcal{R}}(r^{\sharp''}) \subseteq \gamma_{\mathcal{R}}(r^{\sharp'})$$

and also $r|_{\mathcal{X} \cup \bigcup \{\bar{\mathcal{G}}[a'] \mid a' \in S \setminus a\}} \subseteq \gamma_{\mathcal{R}}(r^{\sharp'})$ (auxiliary property (a)).

Altogether, $t' \in \eta'_{\mathsf{mm}}\,[u', S \setminus \{a\}, A']$ holds for all $t \in \eta^k\,[u, S, A_0]$. We conclude that the return value of $[\![([u, S, A_0], \mathsf{unlock}(a), u')]\!]_{\mathsf{mm}}\,\eta^k$ is subsumed by the value $\eta'_{\mathsf{mm}}\,[u', S \setminus \{a\}, A']$ and that auxiliary property (a) holds for the unknown on the left-hand side. It remains to consider the side-effects. Consider some mutex a and associated cluster $Q \in \mathcal{Q}_a$. First, consider the case where $W^\sharp \cap \bar{\mathcal{G}}[a] \neq \emptyset$, and $(H'\,a)_2 = (i, C)$. The side-effects to the unknowns associated with a and Q are then given by

$$\begin{array}{rcl} \rho_1 & = & \{[a, Q, ((i, C), H')] \mapsto T\} \\ \rho_1^\sharp & = & \{[a, Q, (i, C)] \mapsto r^\sharp\big|_Q\} \end{array}$$

Let $r_\rho^{\sharp'} = \eta^\sharp\,[a, Q, (i, C)]$. As $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $r^\sharp\big|_Q \sqsubseteq r_\rho^{\sharp'}$ holds. By construction, we have

$$\begin{array}{rcl} & & \eta'_{\mathsf{mm}}[a, Q, ((i, C), H')] = \gamma_{a,((i,C),H')}(\eta^\sharp\,[a, Q, (H'\,a)_2]) \\ & = & \gamma_{a,((i,C),H')}(\eta^\sharp\,[a, Q, (i, C)]) \\ & = & \gamma_{a,((i,C),H')}(r_\rho^{\sharp'}) \\ & = & \{t \in \mathcal{T} \mid \mathsf{last}\,t = \mathsf{unlock}(a), \mathsf{unlock\_after\_write}_a\,t = (\bar{u}_i, \_, \_), t' = \downarrow_{\bar{u}_i}(t), \\ & & \quad \alpha_{\mathcal{A}_1}\,t' = (i, C), (J, L, W, r) = \beta\,t', r \in \gamma_{\mathcal{R}}\,r_\rho^{\sharp'}\} \\ & & \cup \{t \in \mathcal{T} \mid \mathsf{last}\,t = \mathsf{unlock}(a) \vee \mathsf{last}\,t = \mathsf{initMT}, \mathsf{unlock\_after\_write}_a\,t = \bot\} \end{array}$$

Consider a $t' \in T$ as constructed before. Then, we have $\mathsf{last}\,t' = \mathsf{unlock}(a)$, and either

- for $\mathsf{unlock\_after\_write}_a\,t' = (\bar{u}_i, \_, \_)$, $t'' = \downarrow_{\bar{u}_i}(t')$ with $\alpha_{\mathcal{A}_1}\,t'' = (i, C)$ as $(H'\,a)_2 = (i, C)$. Now consider $(\_, \_, \_, r_\rho) = \beta\,t''$. For $g \in Q$, since $(H'\,a)_2 = (i, C)$ and that last write thus was thread-local, we have $r_\rho\,g = (\mathsf{ex}\,t')\,g = (\mathsf{ex}\,t)\,g$. By induction hypothesis, $(\mathsf{ex}\,t) \in \gamma_{\mathcal{R}}\,r^\sharp$, and thus $r_\rho\big|_Q \in \gamma_{\mathcal{R}}\left(r^\sharp\big|_Q\right) \subseteq \gamma_{\mathcal{R}}\left(r_\rho^{\sharp'}\right)$ and auxiliary property (a) continues to hold; or
- $\mathsf{unlock\_after\_write}_a\,t' = \bot$.

Therefore, $t' \in \gamma_{a,((i,C),H')}(\eta^\sharp\,[a, Q, (H'\,a)_2]) = \eta'_{\mathsf{mm}}[a, Q, ((i, C), H')]$ holds for all $t \in \eta^i\,[u, S, ((i, C), H_0)]$ when $W^\sharp \cap \bar{\mathcal{M}}[a] \neq \emptyset$ and $(H'\,a)_2 = (i, C)$.

It thus remains to consider the case where $W^\sharp \cap \bar{\mathcal{G}}[a] = \emptyset$ or $(H'\,a)_2 \neq (i, C)$. Both conditions imply that the last unlock of the mutex a immediately following a write to a global protected by a is not the unlock considered here. Let $(i', C') = (H'\,a)_2$ and let $r_\rho^{\sharp'} = \eta^\sharp\,[a, Q, (i', C')]$. The side-effects in the concrete for some cluster $Q \in \mathcal{Q}_a$ are then given by

$$\rho_1 \quad = \quad \{[a, Q, ((i, C), H')] \mapsto T\}$$

whereas no (relevant) side-effects are caused in the abstract. By construction, we have

$$\begin{array}{rcl} & & \eta'_{\mathsf{mm}}[a, Q, ((i, C), H')] = \gamma_{a,((i,C),H')}(\eta^\sharp\,[a, Q, (H'\,a)_2]) \\ & = & \gamma_{a,((i,C),H')}(\eta^\sharp\,[a, Q, (i', C')]) \\ & = & \gamma_{a,((i,C),H')}(r_\rho^{\sharp'}) \\ & = & \{t \in \mathcal{T} \mid \mathsf{last}\,t = \mathsf{unlock}(a), \mathsf{unlock\_after\_write}_a\,t = (\bar{u}_i, \_, \_), t' = \downarrow_{\bar{u}_i}(t), \\ & & \quad \alpha_{\mathcal{A}_1}\,t' = (i', C'), (J, L, W, r) = \beta\,t', r \in \gamma_{\mathcal{R}}\,r_\rho^{\sharp'}\} \\ & & \cup \{t \in \mathcal{T} \mid \mathsf{last}\,t = \mathsf{unlock}(a) \vee \mathsf{last}\,t = \mathsf{initMT}, \mathsf{unlock\_after\_write}_a\,t = \bot\} \end{array}$$

Consider a $t' \in T$ as constructed before. We have $\mathsf{last}\, t' = \mathsf{unlock}(a)$. Consider first the case where $\mathsf{unlock\_after\_write}_a\, t' = \bot$: In this case we have $t' \in \eta'_{\mathsf{mm}}\,[a, Q, ((i, C), H')]$. Alternatively, consider $\mathsf{unlock\_after\_write}_a\, t = \mathsf{unlock\_after\_write}_a\, t' = (\bar{u}_i, \_, \_)$, $t'' = \downarrow_{\bar{u}_i} (t')$. We remark that this local trace was considered in some earlier iteration of the fixpoint computation. By Proposition 29 and the induction hypothesis, we have $t'' \in \eta^k\,[a, Q, ((i', C'), H'')] \subseteq \eta'_{\mathsf{mm}}\,[a, Q, ((i', C'), H'')]$ for $\alpha_{\mathcal{A}}(t'') = ((i', C'), H'')$ where $(H''\, a)_2 = (i', C')$. As

$$\eta'_{\mathsf{mm}}\,[a, Q, (i', C'), H''] = \gamma_{a,((i',C'),H'')}(\eta^\sharp\,[a, Q, (i', C')]) = \gamma_{a,((i',C'),H'')}(r_\rho^{\sharp'})$$

we have for $\beta\, t'' = (\_, \_, \_, r)$ that $r \in \gamma_{\mathcal{R}}\, r_\rho^{\sharp'}$ and thus also $t' \in \eta'_{\mathsf{mm}}\,[a, Q, ((i, C), H')]$. Hence, all side-effects for $\mathsf{unlock}(a)$ of $\mathcal{C}_{\mathsf{mm}}$ are accounted for in η'_{mm}, and the claim holds.

Next, consider the constraints corresponding to calling **join**. Consider an edge $(u, x = \mathsf{join}(x'), u') \in \mathcal{E}$ and digests $A' = ((i, C), H')$, $A_0 = ((i, C), H_0)$, and $A_1 = ((i_1, C_1), H_1)$ such that $((i, C), H') \in [\![u, x = \mathsf{join}(x')]\!]^\sharp_{\mathcal{A}}(((i, C), H_0), ((i_1, C_1), H_1))$. We verify that

$$[\![([u, S, ((i, C), H_0)], x = \mathsf{join}(x'), u'), ((i_1, C_1), H_1)]\!]_{\mathsf{mm}}\, \eta^k \subseteq (\eta'_{\mathsf{mm}}, \eta'_{\mathsf{mm}}\,[u', S, ((i, C), H')])$$

We have

$$\begin{array}{l}
[\![([u, S, A_0], x = \mathsf{join}(x'), u'), A_1]\!]_{\mathsf{mm}}\, \eta_{\mathsf{mm}} = \\
\quad \textbf{let } T_1 = \bigcup\{\eta_{\mathsf{mm}}\,[t(x'), A_1] \mid t \in \eta_{\mathsf{mm}}\,[u, S, A_0]\} \textbf{ in} \\
\quad \textbf{let } T = [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\,[u, S, A_0], T_1) \textbf{ in} \\
\quad (\emptyset, T)
\end{array}$$

$$\begin{array}{l}
[\![[u, S, (i, C)], x = \mathsf{join}(x'), (i_1, C_1)]\!]^\sharp \eta^\sharp = \\
\quad \textbf{let } (J^\sharp, L^\sharp, W^\sharp, r^\sharp) = \eta^\sharp\,[u, S, (i, C)] \textbf{ in} \\
\quad \textbf{if } i_1 \notin ((\mathsf{unlift}\, r^\sharp)\, x') \textbf{ then} \\
\qquad (\emptyset, \underline{\bot}) \\
\quad \textbf{elseif } \mathsf{unique}\, i_1 \wedge (i = i_1 \vee i_1 \in J^\sharp) \textbf{ then} \\
\qquad (\emptyset, \underline{\bot}) \\
\quad \textbf{else} \\
\qquad \textbf{let } (J_j^\sharp, L_j^\sharp, v_j^\sharp) = \eta^\sharp[(i_1, C_1)] \textbf{ in} \\
\qquad \textbf{let } J^{\sharp''} = J^\sharp \cup J_j^\sharp \cup \{i_1\} \textbf{ in} \\
\qquad \textbf{let } L^{\sharp''} = L^\sharp \sqcup L_j^\sharp \textbf{ in} \\
\qquad \textbf{let } r^{\sharp''} = [\![x \leftarrow^\sharp (\mathsf{unlift}\, v_j^\sharp)\, \mathsf{ret}]\!]^\sharp_{\mathcal{R}} r^\sharp \textbf{ in} \\
\qquad (\emptyset, (J^{\sharp''}, L^{\sharp''}, W^\sharp, r^{\sharp''}))
\end{array}$$

Let $\eta^\sharp\,[u, S, (i, C)] = (J^\sharp, L^\sharp, W^\sharp, r^\sharp)$ and $\eta^\sharp\,[u', S, (i, C)] = (J^{\sharp'}, L^{\sharp'}, W^{\sharp'}, r^{\sharp'})$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge and the resulting lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, one of the following holds:

(1) $i_1 \notin ((\mathsf{unlift}\, r^\sharp)\, x')$; or

(2) $\mathsf{unique}\, i_1 \wedge (i = i_1 \vee i_1 \in J^\sharp)$; or

(3) $J^{\sharp''} \sqsubseteq J^{\sharp'}$, $L^{\sharp''} \sqsubseteq L^{\sharp'}$, $W^\sharp \subseteq W^{\sharp'}$, and $r^{\sharp''} \sqsubseteq r^{\sharp'}$ all hold.

Then, by definition:

$$\begin{array}{rl} & \eta'_{\mathsf{mm}}[u', S, ((i,C), H')] = \gamma_{u',S,((i,C),H')}((J^{\sharp'}, L^{\sharp'}, W^{\sharp'}, r^{\sharp'})) \\ = & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = S, \alpha_{\mathcal{A}_1}\, t = (i,C), (J,L,W,r) = \beta\, t, \\ & \qquad J^{\sharp'} \subseteq_j J, L \sqsubseteq_t L^{\sharp'}, W \subseteq W^{\sharp'}, r \in \gamma_{\mathcal{R}}\, r^{\sharp'}\} \end{array}$$

Further, let $\eta^\sharp\,[(i_1, C_1)] = (J_j^\sharp, L_j^\sharp, v_j^\sharp)$. For every trace $t \in \eta^k\,[u, S, A_0]$, let $\beta\, t = (J, L, W, r)$. By induction hypothesis, $J^\sharp \subseteq_j J$, $L \sqsubseteq_t L^\sharp$, $W \subseteq W^\sharp$, and $r \in \gamma_{\mathcal{R}}\,(r^\sharp)$. Let

$$\begin{array}{rcl} T' & = & [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\{t\}, \bigcup\{\eta^k\,[(t'(x')), A_1] \mid t' \in \eta^k\,[u, S, A_0]\}) \\ & = & [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\{t\}, \eta^k\,[(t(x')), A_1]) \end{array}$$

where the equality exploits that for a given local trace t, $[\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\{t\}, \{t'\})$ only yields a non-empty set if the thread *id* of the thread being joined is the one stored in x'. We distinguish between whether (1), (2), or (3) holds.

First, assume that (3) holds. If T' is empty, it is subsumed by $\eta'_{\mathsf{mm}}\,[u', S, A']$ vacuously and auxiliary property (a) also holds for the next approximation. Consider thus a $t' \in T'$ and $t'' \in \eta^k\,[(t(x')), A_1]$ such that $\{t'\} = [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\{t\}, \{t''\})$. By induction hypothesis, we have

$$\begin{array}{rcl} t'' & \in & \gamma_{t(x'),((i_1,C_1),H_1)}(\eta^\sharp\,[(i_1, C_1)]) = \gamma_{t(x'),((i_1,C_1),H_1)}((J_j^\sharp, L_j^\sharp, v_j^\sharp)) = \\ & & \{t'' \in \mathcal{T} \mid \mathsf{last}\, t'' = \mathsf{return}, \alpha_{\mathcal{A}_1}\, t'' = (i_1, C_1), \mathsf{id}\, t'' = t(x'), (J_j, L_j, W_j, r_j) = \beta\, t'', \\ & & \qquad J_j^\sharp \subseteq_j J_j, L_j \sqsubseteq_t L_j^\sharp, r_j \in \gamma_{\mathcal{R}}\, v_j^\sharp\} \end{array}$$

Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S$, $\alpha_{\mathcal{A}_1}\, t' = (i, C)$. Let $(J', L', W', r') = \beta\, t'$.

- We have $t(x') \in \gamma_{\mathcal{V}^\sharp_{\mathbf{tid}},\mathcal{A}}\, i_1$ and thus

$$\begin{array}{rcll} J^{\sharp''} \subseteq J^{\sharp'} = J^\sharp \cup J_j^\sharp \cup \{i_1\} & \subseteq_j & J \cup J_j \cup \{t(x')\} & \\ & = & \{\sigma_i\, x' \mid ((i, u_i, \sigma_i), x{=}\mathsf{join}(x'), \_) \in \mathsf{jl\_joins}\ t'\} & \\ & = & J' & \end{array}$$

- For L', we have for each mutex a,

$$L'a \;=\; \begin{cases} \mathsf{ex}\, t''' & \text{if } \mathsf{jl\_unlock\_after\_write}_a\, t' = \bot, t''' = {\downarrow_{(u_0,0,\sigma_0)}}\,(t') \\ \mathsf{ex}\, t''' & \text{if } \mathsf{jl\_unlock\_after\_write}_a\, t' = (\bar{u}_i, \_, \_), t''' = {\downarrow_{\bar{u}_i}}\,(t') \end{cases}$$

  Furthermore, we have either $\mathsf{jl\_unlock\_after\_write}_a\, t' = \mathsf{jl\_unlock\_after\_write}_a\, t$ or $\mathsf{jl\_unlock\_after\_write}_a\, t' = \mathsf{jl\_unlock\_after\_write}_a\, t''$. Consider some mutex a. If $\mathsf{jl\_unlock\_after\_write}_a\, t' \neq \mathsf{unlock\_after\_write}_a\, t'$, the property $L' \sqsubseteq_t L^{\sharp'}$ w.r.t. a as well auxiliary property (a) w.r.t. a hold trivially. Otherwise, first consider the case

where $\mathsf{jl\_unlock\_after\_write}_a\, t' = \mathsf{jl\_unlock\_after\_write}_a\, t$. Then, by induction hypothesis we have $L'\, a = L\, a$ and by auxiliary property (a) thus

$$\begin{array}{lcl} \forall Q \in \mathcal{Q}_a : L'\, a \in L'\, a|_Q = L\, a|_Q & \subseteq & \gamma_{\mathcal{R}}\, L^\sharp\, (a, Q) \subseteq \gamma_{\mathcal{R}}\, (L^\sharp\, (a, Q) \sqcup L_j^\sharp\, (a, Q)) \\ & = & \gamma_{\mathcal{R}}\, L^{\sharp''}\, (a, Q) \subseteq \gamma_{\mathcal{R}}\, L^{\sharp'}\, (a, Q) \end{array}$$

Analogously when $\mathsf{jl\_unlock\_after\_write}_a\, t' = \mathsf{jl\_unlock\_after\_write}_a\, t''$ for $L_j\, a$ and $L_j^\sharp\, a$. Altogether, we have $L' \sqsubseteq_t L^{\sharp'}$ and auxiliary property (a) holds for the next approximation.

- Next for W' and $W^{\sharp'}$: As t and t' coincide on their thread-local parts, (save for the additional join edge), we have

$$\begin{array}{lcl} W' & = & \bigcup_{a \in \mathcal{M}} \{g \mid g \in \bar{\mathcal{G}}[a], \mathsf{last\_tl\_write}_g\, t' = (\bar{u}_i, \_, \_), \\ & & ((\mathsf{last\_tl\_unlock}_a\, t' = (\bar{u}_j, \_, \_) \wedge \bar{u}_j \leq \bar{u}_i) \vee \mathsf{last\_tl\_unlock}_a\, t' = \bot)\} \\ & = & \bigcup_{a \in \mathcal{M}} \{g \mid g \in \bar{\mathcal{G}}[a], \mathsf{last\_tl\_write}_g\, t = (\bar{u}_i, \_, \_), \\ & & ((\mathsf{last\_tl\_unlock}_a\, ' = (\bar{u}_j, \_, \_) \wedge \bar{u}_j \leq \bar{u}_i) \vee \mathsf{last\_tl\_unlock}_a\, t = \bot)\} \\ & = & W \subseteq W^\sharp \subseteq W^{\sharp'} \end{array}$$

- Lastly, for r': Then, $\forall y \in \mathcal{X} \setminus \{x\}, r'\, y = r\, y$. For all globals protected by some currently held mutex $\{g \mid g \in \mathcal{G}, S \cap \bar{\mathcal{M}}[g] \neq \emptyset\}$, we also have $t'(g) = t(g)$ and thus $r'\, g = r\, g$. For x, on the other hand, we have $r'\, x = t'(x) = t''(\mathsf{ret})$. It thus remains to relate $v_j^\sharp$ to $t''(\mathsf{ret})$. By induction hypothesis, we have $r_j \in \gamma_{\mathcal{R}}\, v_j^\sharp$ and thus $r'\, \mathsf{ret} = r_j\, \mathsf{ret} \in \gamma_{\mathcal{V}^\sharp}\left((\mathsf{unlift}\, v_j^\sharp)\, \mathsf{ret}\right)$. Altogether, we have

$$\begin{array}{lcl} r' & \in & r'|_{\mathcal{X} \cup \bigcup \{g \in \mathcal{G} \mid S \cap \bar{\mathcal{M}}[g] \neq \emptyset\}} \\ & \subseteq & \{\hat{r} \oplus \{x \mapsto r_j\, \mathsf{ret}\} \mid \hat{r} \in r|_{\mathcal{X} \cup \{g \in \mathcal{G} \mid S \cap \bar{\mathcal{M}}[g] \neq \emptyset\}}\} \\ & \subseteq & \{\hat{r} \oplus \{x \mapsto r_j\, \mathsf{ret}\} \mid \hat{r} \in \gamma_{\mathcal{R}}\, r^\sharp\} \\ & \subseteq & \{\hat{r} \oplus \{x \mapsto v\} \mid \hat{r} \in \gamma_{\mathcal{R}}\, r^\sharp, v \in \gamma_{\mathcal{V}^\sharp}\left((\mathsf{unlift}\, v_j^\sharp)\, \mathsf{ret}\right)\} \\ & \subseteq & \gamma_{\mathcal{R}}\, ([\![x \leftarrow^\sharp (\mathsf{unlift}\, v_j^\sharp)\, \mathsf{ret}]\!]_{\mathcal{R}}^\sharp r^\sharp) = \gamma_{\mathcal{R}}\, (r^{\sharp''}) \subseteq \gamma_{\mathcal{R}}\, (r^{\sharp'}) \end{array}$$

It thus remains to consider the cases where (3) does not hold, but (1) or (2) hold. We show that, in this case, $T' = \emptyset$.

First, consider the case where $i_1 \notin ((\mathsf{unlift}\, r^\sharp)\, x')$. Consider for a contradiction a $t' \in T'$ and $t'' \in \eta^k\, [(t(x')), A_1]$ such that $\{t'\} = [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\{t\}, \{t''\})$ as considered before. By induction hypothesis, we have $t''(\mathsf{self}) \in \gamma_{\mathcal{V}^\sharp_{\mathbf{tid}, \mathcal{A}}}\, i_1$ and also $t(x') \in \gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}((\mathsf{unlift}\, r^\sharp)\, x')$. For $[\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\{t\}, \{t''\})$ to yield a non-empty set, $t''(\mathsf{self}) = t(x')$ needs to hold. This implies that $\gamma_{\mathcal{V}^\sharp_{\mathbf{tid}, \mathcal{A}}}\, i_1 \cap (\gamma_{\mathcal{V}^\sharp_{\mathbf{tid}}}((\mathsf{unlift}\, r^\sharp)\, x')) \neq \emptyset$ and thus by Eq. (4.3) from Section 4.2.3, $i_1 \in ((\mathsf{unlift}\, r^\sharp)\, x')$, which is a contradiction.

Next, for $\mathsf{unique}\, i_1 \wedge (i = i_1 \vee i_1 \in J^\sharp)$: Consider for a contradiction a $t' \in T'$ and $t'' \in \eta^k\, [(t(x')), A_1]$ such that $\{t'\} = [\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\{t\}, \{t''\})$ as considered before.

If unique i_1 holds, the concretization of i_1 is a singleton set. By induction hypothesis, we have $t''(\mathsf{self}) \in \gamma_{\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}}\, i_1$ and $t(\mathsf{self}) \in \gamma_{\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}}\, i$. If $i = i_1$ holds, we thus have $t''(\mathsf{self}) = t(\mathsf{self})$. Since a thread cannot be joined into itself, we have $[\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\{t\}, \{t''\}) = \emptyset$ and thus a contradiction. On the other hand unique i_1 and $i_1 \in J^\sharp$, implies $\gamma_{\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}}(i_1) \in J$ by induction hypothesis, i.e., a thread with the concrete thread *id* corresponding to i_1 has already been joined into t at some point. As a consequence, we once more have $[\![(u, x = \mathsf{join}(x'), u')]\!]_{\mathcal{T}}(\{t\}, \{t''\}) = \emptyset$ and thus a contradiction.

Thus, if (1) or (2) hold, the concrete right-hand side causes no contribution to $\eta'_{\mathsf{mm}}[u', S, A']$. Altogether, the return value of $[\![([u, S, A_0], x = \mathsf{join}(x'), u'), A_1]\!]_{\mathsf{mm}}\, \eta^k$ is always subsumed by the value $\eta'_{\mathsf{mm}}[u', S, A']$ in both cases and auxiliary property (a) also holds for the next approximation. As neither constraint causes any side-effects, the statement holds.

For **thread creation**, **return**, **initMT**, **signal**, and **wait** the proof proceeds analogously to the one outlined in Section 6.2.1. We do not detail this here.

This concludes the case distinction for the inductive step and, thus, the soundness proof for this analysis.

□

Remark 28. *An analysis actually using the precise digests $((i, C), H)$ as described in this section would also be sound and would consider all the different interleavings of critical sections. It would thus be very precise. However, it would, e.g., fail to terminate in the presence of (potentially non-terminating) loops in which globals are modified due to the counter components in H. Thus, this instance is ill-suited for usage in an over-approximating analysis and is mostly of interest as an ingredient of the proof — or potentially for analyses where termination is not a concern. Nevertheless, it may be a good base for an instance of the generalized digest framework that allows further abstractions of digests.*

6.2.3 Mutex-Meet with Joins and Clusters

We refer to the constraint system of the analysis from Section 4.2.5 by $\mathcal{C}^\sharp$, and to the constraint system introduced in Section 4.2.2 by $\mathcal{C}_{\mathsf{mm}}$ once again. We further refer to the $\mathcal{A}$ to compute thread *id*s (see Fig. 2.14) by $\mathcal{A}_1$. The proof once again makes use of a modified digest that tracks the same information as before, but now per mutex and associated cluster.

$$\mathcal{A} = (\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \times 2^{\mathcal{P}}) \times ((\mathcal{M} \times \mathcal{Q}_a) \to (\mathbb{N}_0 \times (\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \times 2^{\mathcal{P}}) \times 2^{\mathcal{G}}))$$

The right-hand sides are then given in Fig. 6.4 with the function $[\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}$ for a mutex a given by

$$[\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}((i, C), H)\,((i', C'), H') = \begin{cases} \emptyset & \exists Q \in \mathcal{Q}_a : \neg \mathsf{may\_run}\,(i, C)\,(H'\,(a, Q))_2 \\ \{((i, C), \mathsf{merge}\, H\, H')\} & \text{else} \end{cases}$$

$$\mathsf{init}^\sharp_{\mathcal{A}} = \{(((\mathsf{main}, \emptyset), \emptyset), \{(a, Q) \mapsto (0, ((\mathsf{main}, \emptyset), \emptyset)) \mid a \in \mathcal{M}, Q \in \mathcal{Q}_a\})\}$$

$$[\![u, x{=}\mathsf{create}(u_1)]\!]^\sharp_{\mathcal{A}}\,((i, C), H) = \{((i, C \cup \{\langle u, u_1\rangle\}), H)\}$$

$$\begin{aligned}[\![u, g = x]\!]^\sharp_{\mathcal{A}}\,((i, C), H) = \{&((i, C),\\ &\{(a, Q) \mapsto ((H\,(a, Q))_1, (H\,(a, Q))_2, (H\,(a, Q))_3 \cup (\{g\} \cap \bar{\mathcal{G}}[a]))\\ &\quad \mid a \in \mathcal{M}, Q \in \mathcal{Q}_a\})\}\end{aligned}$$

$$\begin{aligned}[\![u, \mathsf{unlock}(a)]\!]^\sharp_{\mathcal{A}}((i, C), H) = \{&((i, C),\\ &H \oplus \{(a, Q) \mapsto ((H\,(a, Q))_1 + 1, (i, C), \emptyset)\\ &\qquad \mid Q \in \mathcal{Q}_a, (H\,(a, Q))_3 \neq \emptyset\})\}\end{aligned}$$

$$[\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}\,((i, C), H) = \{((i, C), H)\} \quad \text{(other non-observing)}$$

Figure 6.4: Modified instance of $\mathcal{A}$ used in the proof tracking additional information on top of thread *id*s. The definition of $\circ$ is the one given in Fig. 2.14 and the definition of $\mathsf{new}^\sharp_{\mathcal{A}}\,u\,u_1\,A$ is reused from Fig. 6.3.

where

$$\begin{aligned}\mathsf{merge}(H, H') \;=\; &\{(b, Q) \mapsto H\,(b, Q) \mid b \in \mathcal{M}, Q \in \mathcal{Q}_b, (H\,(b, Q))_1 \geq (H'\,(b, Q))_1\}\\ &\cup \{(b, Q) \mapsto \bar{H}'\,(b, Q) \mid b \in \mathcal{M}, Q \in \mathcal{Q}_b, (H\,(b, Q))_1 < (H'\,(b, Q))_1\}\end{aligned}$$

where once more $(\cdot)_k$ is shorthand for accessing the k-th component of a tuple and $\bar{H}$ denotes setting the 3-rd component of all $H\,(b.Q)$ to $\emptyset$. This definition is also used for $[\![u, x = \mathsf{join}(x')]\!]^\sharp_{\mathcal{A}}$ and $[\![u, \mathsf{wait}(s)]\!]^\sharp_{\mathcal{A}}$. One again, the may_run predicate here does not check for the current (i', C') of the thread performing the respective unlock, but instead refers to the (i, C) at the last thread-local write to a global protected by a in cluster Q.

The concretization for abstract values at unknowns $[a, Q, ((i, C), H)]$ is defined such that it contains any local trace ending in an $\mathsf{unlock}(a)$ in which the last thread-local write to a global in the *cluster* Q, was by the thread with the thread *id* $(H\,(a, Q))_2$.

As before, we once again introduce a function β that maps a local trace to some information extracted from it that can then be related to the information computed by the analysis. For this, per cluster counterparts of $\mathsf{jl\_unlock\_after\_write}_a$ and $\mathsf{unlock\_after\_write}_a$ are needed. Let $\mathsf{jl\_unlock\_after\_write}_{a,Q}$ and $\mathsf{unlock\_after\_write}_{a,Q}$ denote these counterparts referring to the respective cluster only.

The function β then has the following type here:

$$\beta : \mathcal{T} \to (2^{\mathcal{V}_{\mathsf{tid}}} \times ((\mathcal{M} \times \mathcal{Q}) \to (\mathcal{V}ars \to \mathcal{V})) \times 2^{\mathcal{G}} \times (\mathcal{V}ars \to \mathcal{V}))$$

We set:

$$\begin{aligned}\beta\,t \;&=\; (J, L, W, r) \text{ where}\\ J \;&=\; \{\sigma_i\,x' \mid ((i, u_i, \sigma_i), x{=}\mathsf{join}(x'), \_) \in \mathsf{jl\_joins}\ t\}\end{aligned}$$

$$
\begin{aligned}
L &= \{(a,Q) \mapsto \mathsf{ex}\, t' \mid a \in \mathcal{M}, Q \in \mathcal{Q}_a, \\
&\qquad \mathsf{jl\_unlock\_after\_write}_{a,Q}\, t = \bot, t' = \downarrow_{(u_0,0,\sigma_0)} (t)\} \cup \\
&\quad \{(a,Q) \mapsto \mathsf{ex}\, t' \mid a \in \mathcal{M}, Q \in \mathcal{Q}_a, \\
&\qquad \mathsf{jl\_unlock\_after\_write}_{a,Q}\, t = (\bar{u}_i, \_, \_), t' = \downarrow_{\bar{u}_i} (t)\} \\
W &= \bigcup_{a \in \mathcal{M}} \{g \mid g \in \bar{\mathcal{G}}[a], \mathsf{last\_tl\_write}_g\, t = (\bar{u}_i, \_, \_), \\
&\qquad ((\mathsf{last\_tl\_unlock}_a\, t = (\bar{u}_j, \_, \_) \wedge \bar{u}_j \leq \bar{u}_i) \vee \mathsf{last\_tl\_unlock}_a\, t = \bot)\} \\
r &= \mathsf{ex}\, t
\end{aligned}
$$

where the helper function ex is defined as in the previous chapter. We remark that J, W, and r are also identical to the definition from the previous section.
The abstraction function β is used to specify concretization functions for the values of unknowns $[u, S, ((i,C),H)]$ for program points, currently held locksets, and digests as well as for the other unknowns, where $\mathcal{A}_1$ refers to the digests as considered in Section 2.8.

$$
\begin{aligned}
\gamma_{u,S,((i,C),H)}(J^\sharp, L^\sharp, W^\sharp, r^\sharp) &= \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u, L_t = S, \alpha_{\mathcal{A}_1}\, t = (i,C), \\
&\qquad (J,L,W,r) = \beta\, t, \\
&\qquad J^\sharp \subseteq_j J, L \sqsubseteq_t L^\sharp, W \subseteq W^\sharp, r \in \gamma_{\mathcal{R}}\, r^\sharp\} \\
\gamma_{i',((i,C),H)}(J^\sharp, L^\sharp, r^\sharp) &= \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{return}, \alpha_{\mathcal{A}_1}\, t = (i,C), \mathsf{id}\, t = i', \\
&\qquad (J,L,W,r) = \beta\, t, \\
&\qquad J^\sharp \subseteq_j J, L \sqsubseteq_t L^\sharp, r \in \gamma_{\mathcal{R}}\, r^\sharp\} \\
\gamma_{a,Q,((i,C),H)}(r^\sharp) &= \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{unlock}(a), \\
&\qquad \mathsf{unlock\_after\_write}_{a,Q}\, t = (\bar{u}_i, \_, \_), \\
&\qquad t' = \downarrow_{\bar{u}_i} (t), \alpha_{\mathcal{A}_1}\, t' = (H\,(a,Q))_2 \\
&\qquad (J,L,W,r) = \beta\, t', r \in \gamma_{\mathcal{R}}\, r^\sharp\} \\
&\quad \cup \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{unlock}(a) \vee \mathsf{last}\, t = \mathsf{initMT}, \\
&\qquad \mathsf{unlock\_after\_write}_{a,Q}\, t = \bot\} \\
\gamma_{s,((i,C),H)}(J^\sharp, L^\sharp, W^\sharp, r^\sharp) &= \{t \in \mathcal{T} \mid \mathsf{last}\, t = \mathsf{signal}(s), \alpha_{\mathcal{A}_1}\, t = (i,C), \\
&\qquad (J,L,W,r) = \beta\, t, \\
&\qquad J^\sharp \subseteq_j J, L \sqsubseteq_t L^\sharp, W \subseteq W^\sharp, r \in \gamma_{\mathcal{R}}\, r^\sharp\}
\end{aligned}
$$

where $\subseteq_j$ is defined as in the previous section. Here, let $L \sqsubseteq_t L^\sharp$ denote checking that

$$
\begin{aligned}
\forall a \in \mathcal{M} : \forall Q \in \mathcal{Q}_a : (&\mathsf{jl\_unlock\_after\_write}_{a,Q}\, t = \mathsf{unlock\_after\_write}_{a,Q}\, t \implies \\
&L\,(a,Q) \in \gamma_{\mathcal{R}}\,(L^\sharp(a,Q)))
\end{aligned}
$$

This predicate thus checks that if the last unlock of a mutex a immediately succeeding a write to a global in Q is join-local to t, $L^\sharp\,(a,Q)$ soundly abstracts the value of locals and globals at the program point corresponding to that unlock. We remark that these concretizations are once more monotonic and that, except for the unknowns for mutexes and the slightly modified meaning of $\sqsubseteq_t$, they are identical to the concretizations proposed in the previous section.

For a solution $\eta^\sharp$ of $\mathcal{C}^\sharp$ as considered in Section 4.2.5, we then construct a mapping η'_{mm} by:

$$\begin{array}{lcll}
\eta'_{\mathsf{mm}}[u,S,((i,C),H)] & = & \gamma_{u,S,((i,C),H)}(\eta^\sharp\,[u,S,(i,C)]) & u \in \mathcal{N}, S \subseteq \mathcal{M}, \\
\eta'_{\mathsf{mm}}[a,Q,((i,C),H)] & = & \gamma_{a,Q,((i,C),H)}(\eta^\sharp\,[a,Q,(H\,(a,Q))_2]) & a \in \mathcal{M}, Q \in \mathcal{Q}_a \\
\eta'_{\mathsf{mm}}[i',((i,C),H)] & = & \gamma_{i',((i,C),H)}(\eta^\sharp\,[(i,C)]) & i' \in \mathcal{V}_{\mathbf{tid}} \\
\eta'_{\mathsf{mm}}[s,((i,C),H)] & = & \gamma_{s,((i,C),H)}(\eta^\sharp\,[s,(i,C)]) & s \in \mathcal{S}
\end{array}$$

for $((i,C),H) \in \mathcal{A}$ where we remark that the difference to the definition in the previous section is that we now consider the information associated with a specific cluster associated with a mutex. Altogether, correctness of $\mathcal{C}^\sharp$ follows from the following theorem:

Theorem 22. *Every solution of $\mathcal{C}^\sharp$ is sound w.r.t. the local trace semantics.*

Proof. Recall from Proposition 28, that the least solution of $\mathcal{C}_{\mathsf{mm}}$ is sound w.r.t. the local trace semantics as specified by the constraint system $\mathcal{C}$. It thus suffices to prove that the mapping η'_{mm} as constructed above, is a solution of the constraint system $\mathcal{C}_{\mathsf{mm}}$. For that, we verify by fixpoint induction that for the k-th approximation η^k to the least solution η of $\mathcal{C}_{\mathsf{mm}}$, $\eta^k \subseteq \eta'_{\mathsf{mm}}$ holds.

As in previous proofs, we require some auxiliary property stating that abstract values only speak about locals and protected globals (for unknowns associated with program points), respectively those globals that form a part of a given cluster (for unknowns associated with a mutex and a cluster). Let us call auxiliary property (a) that

$$\begin{array}{lll}
\forall t \in \eta^k\,[u,S,((i,C),H)] & : \beta\, t = (J,L,W,r), (J^\sharp,L^\sharp,W^\sharp,r^\sharp) = \eta^\sharp\,[u,S,(i,C)] \implies \\
& \quad r|_{\mathcal{X}\cup\{g\in\mathcal{G},\bar{\mathcal{M}}[g]\cap S\neq\emptyset\}} \subseteq \gamma_{\mathcal{R}}\, r^\sharp \wedge \forall a \in \mathcal{M}, \forall Q \in \mathcal{Q}_a : \\
& \quad \mathsf{jl_unlock_after_write}_{a,Q}\, t = \mathsf{unlock_after_write}_{a,Q}\, t \\
& \qquad \implies (L\,(a,Q))|_Q \in \gamma_{\mathcal{R}}\,(L^\sharp\,(a,Q)) \\
& \qquad\qquad (\text{for } u \in \mathcal{N}, S \subseteq \mathcal{M}, ((i,C),H) \in \mathcal{A}) \\
\forall t \in \eta^k\,[a,Q,((i,C),H)] & : \beta\, t = (J,L,W,r) \wedge \mathsf{unlock_after_write}_{a,Q}\, t \neq \bot \implies \\
& \quad r|_Q \in \gamma_{\mathcal{R}}\,(\eta^\sharp\,[a,Q,(H\,(a,Q))_2]) \\
& \qquad\qquad (\text{for } a \in \mathcal{M}, Q \in \mathcal{Q}_a, ((i,C),H) \in \mathcal{A}) \\
\forall t \in \eta^k\,[i',((i,C),H)] & : \beta\, t = (J,L,r), (J^\sharp,L^\sharp,W^\sharp,r^\sharp) = \eta^\sharp\,[(i,C)] \implies \\
& \quad \forall a \in \mathcal{M}, \forall Q \in \mathcal{Q}_a : \\
& \quad \mathsf{jl_unlock_after_write}_{a,Q}\, t = \mathsf{unlock_after_write}_{a,Q}\, t \\
& \qquad \implies (L\,(a,Q))|_Q \in \gamma_{\mathcal{R}}\,(L^\sharp\,(a,Q)) \\
& \qquad\qquad (\text{for } u \in \mathcal{N}, S \subseteq \mathcal{M}, ((i,C),H) \in \mathcal{A})
\end{array}$$

For the zero-th iteration η^k is $\emptyset$ everywhere, and thus auxiliary property (a) hold.

The proof proceeds largely analogously to the proof in the previous section. We only exemplify it for the case of locking a mutex here:

Consider the constraints corresponding to **locking** a mutex a. Consider an edge $(u, \mathsf{lock}(a), u') \in \mathcal{E}$ and digests $A' = ((i, C), H')$, $A_0 = ((i, C), H_0)$, and $A_1 = ((i_1, C_1), H_1)$ such that $((i, C), H') \in [\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}(((i, C), H_0), ((i_1, C_1), H_1))$. We verify that

$$[\![([u, S, ((i, C), H_0)], \mathsf{lock}(a), u'), ((i_1, C_1), H_1)]\!]_{\mathsf{mm}}\, \eta^k \subseteq (\eta'_{\mathsf{mm}}, \eta'_{\mathsf{mm}}\, [u', S \cup \{a\}, ((i, C), H')])$$

We have

$$\begin{array}{l}
[\![([u, S, A_0], \mathsf{lock}(a), u'), A_1]\!]_{\mathsf{mm}}\, \eta_{\mathsf{mm}} = \\
\quad \textbf{let } T_1 = \bigcap\{\eta_{\mathsf{mm}}\, [a, Q, A_1] \mid Q \in \mathcal{Q}_a\} \textbf{ in} \\
\quad \textbf{let } T = [\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}(\eta_{\mathsf{mm}}\, [u, S, A_0], T_1) \textbf{ in} \\
\quad (\emptyset, T)
\end{array}$$

$$\begin{array}{l}
[\![[u, S, (i, C)], \mathsf{lock}(a), \mathbf{I}]\!]^\sharp \eta^\sharp = \\
\quad \textbf{let } (J^\sharp, L^\sharp, W^\sharp, r^\sharp) = \eta^\sharp\, [u, S, (i, C)] \textbf{ in} \\
\quad \textbf{let } B^\sharp(Q) = \bigsqcup\{\textbf{if } \mathsf{unique}\, \bar{i}_1 \wedge (i = \bar{i}_1 \vee \bar{i}_1 \in J^\sharp) \textbf{ then } \bot \textbf{ else } \eta^\sharp\, [a, Q, (\bar{i}_1, \bar{C}_1)] \\
\qquad\qquad\qquad \mid (\bar{i}_1, \bar{C}_1) \in \mathbf{I}\} \sqcup L^\sharp\, (a, Q) \\
\quad \textbf{in} \\
\quad \textbf{let } r^\sharp_m = \bigsqcap_{Q \in \mathcal{Q}_a}(B^\sharp(Q)) \textbf{ in} \\
\quad \textbf{let } r^{\sharp''} = r^\sharp \sqcap r^\sharp_m \textbf{ in} \\
\quad \left(\emptyset, \left(J^\sharp, L^\sharp, W^\sharp, r^{\sharp''}\right)\right)
\end{array}$$

Let $\eta^\sharp\, [u, S, (i, C)] = (J^\sharp, L^\sharp, W^\sharp, r^\sharp)$ and $\eta^\sharp\, [u', S \cup \{a\}, (i, C)] = (J^{\sharp'}, L^{\sharp'}, W^{\sharp'}, r^{\sharp'})$ the value provided by $\eta^\sharp$ for the endpoint of the given control-flow edge and the resulting lockset and digest. Since $\eta^\sharp$ is a solution of $\mathcal{C}^\sharp$, $J^\sharp \sqsubseteq J^{\sharp'}$, $L^\sharp \sqsubseteq L^{\sharp'}$, $W^\sharp \subseteq W^{\sharp'}$, and $r^{\sharp''} \sqsubseteq r^{\sharp'}$ all hold. Then, by definition:

$$\begin{array}{rl}
& \eta'_{\mathsf{mm}}[u', S \cup \{a\}, ((i, C), H')] = \gamma_{u', S \cup \{a\}, ((i,C), H')}((J^{\sharp'}, L^{\sharp'}, W^{\sharp'}, r^{\sharp'})) \\
= & \{t \in \mathcal{T} \mid \mathsf{loc}\, t = u', L_t = S \cup \{a\}, \alpha_{\mathcal{A}_1}\, t = (i, C), (J, L, W, r) = \beta\, t, \\
& \qquad J^{\sharp'} \sqsubseteq_j J, L \sqsubseteq_t L^{\sharp'}, W \subseteq W^{\sharp'}, r \in \gamma_{\mathcal{R}}\, r^{\sharp'}\}
\end{array}$$

For every trace $t \in \eta^k\, [u, S, A_0]$, let $\beta\, t = (J, L, W, r)$. By induction hypothesis, $J^\sharp \sqsubseteq_j J$, $L \sqsubseteq_t L^\sharp$, $W \subseteq W^\sharp$, and $r \in \gamma_{\mathcal{R}}\, (r^\sharp)$. Consider some trace $t_1 \in \bigcap\{\eta^k\, [a, Q, ((i_1, C_1), H_1)] \mid Q \in \mathcal{Q}_a\}$ such that $[\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}(\{t\}, \{t_1\}) \neq \emptyset$ and let $(\_, \_, \_, r_1) = \beta\, t_1$. Consider further a cluster $Q \in \mathcal{Q}_a$, and a $\bar{I} \in \mathcal{A}_1$ and let $\eta^\sharp[a, Q, \bar{I}] = r^\sharp_{Q,\bar{I}}$.

By induction hypothesis, we thus have

(1) either

$$t_1 \in \{t'' \in \mathcal{T} \mid \mathsf{last}\, t'' = \mathsf{unlock}(a) \vee \mathsf{last}\, t'' = \mathsf{initMT}, \mathsf{unlock\_after\_write}_{a,Q}\, t'' = \bot\}$$

Then $\mathsf{jl\_unlock\_after\_write}_{a,Q}\, t = \bot$ as well, and by induction hypothesis, there is an r_0 (namely the one extracted from the local trace ending in $(u_0, 0, \sigma_0)$) such that $r_0 \in \gamma_R\, (L^\sharp\, (a, Q))$ for all $Q \in \mathcal{Q}_a$. r_1 coincides with r_0 on the values of globals from this cluster, and we thus get by auxiliary property (a):

$$r_1|_Q = r_0|_Q \subseteq \gamma_R\, (L^\sharp\, (a, Q))$$

(2) Otherwise, we have

$$\begin{aligned} t_1 \quad \in \quad \{t'' \in \mathcal{T} \mid & \mathsf{last}\, t'' = \mathsf{unlock}(a), \mathsf{unlock\_after\_write}_{a,Q}\, t'' = (\bar{u}_i, \_, \_), \\ & t''' = \downarrow_{\bar{u}_i} (t''), \\ & \bar{I} = (H_1\,(a,Q))_2, \alpha_{\mathcal{A}_1}\, t''' = \bar{I}, (\_, \_, \_, r'''_{1,Q}) = \beta\, t''', r'''_{1,Q} \in \gamma_{\mathcal{R}}\, r^\sharp_{Q,\bar{I}}\} \end{aligned}$$

As $r'''_{1,Q}$ and $r_{1,Q}$ coincide on the values of all globals in cluster Q, we obtain (once again using auxiliary property (a)),

$$r_1|_{\bar{\mathcal{G}}[a]} \subseteq r_1|_Q = r'''_1|_Q \subseteq \gamma_R\,(r_1^\sharp)$$

Let $t' \in [\![(u, \mathsf{lock}(a), u')]\!]_{\mathcal{T}}(\{t\}, \{t_1\})$. Then $\mathsf{loc}\, t' = u'$, $L_{t'} = S \cup \{a\}$, and $\alpha_{\mathcal{A}_1}\, t' = \alpha_{\mathcal{A}_1}\, t = (i, C)$. Let $(J', L', W', r') = \beta\, t'$.

- First, consider the r component: We first observe that once more $\forall x \in \mathcal{X}, r'\, x = r\, x$. For all globals protected by a, we either have
  - $S \cap \bar{\mathcal{M}}[g] \neq \emptyset$, in which case $r'\, g = r\, g = r_{1,Q}\, g$ for all $Q \in \mathcal{Q}_a$ for which $g \in Q$.
  - $S \cap \bar{\mathcal{M}}[g] = \emptyset$, in which case $r'\, g = r_{1,Q}\, g$ for all $Q \in \mathcal{Q}_a$ for which $g \in Q$.

  Next, we relate $B^\sharp(Q)$ to the values of the globals protected by a in cluster Q. First, let us assume that for a cluster Q, no unlock of a succeeding a write to a global in Q has happened in t and thus also in t_1 and t'. Then,

$$\begin{aligned} & \mathsf{jl\_unlock\_after\_write}_{a,Q}\, t = \mathsf{jl\_unlock\_after\_write}_{a,Q}\, t_1 \\ = \quad & \mathsf{unlock\_after\_write}_{a,Q}\, t = \mathsf{unlock\_after\_write}_{a,Q}\, t_1 \\ = \quad & \bot \end{aligned}$$

  and thus $L\,(a,Q) \in \gamma_{\mathcal{R}}(L^\sharp\,(a,Q))$. Also, we have $r'\, g = r\, g = L\,(a,Q)\, g = 0$ for all $g \in Q$. By auxiliary property (a), we have

$$r' \subseteq r'|_Q = L\,(a,Q)|_Q \in \gamma_{\mathcal{R}}\,(L^\sharp(a,Q))$$

  and thus $r' \in \gamma_{\mathcal{R}}(B^\sharp(Q))$.

  Now, assume that such an unlock succeeding a write to a global in Q has happened: Let $\bar{I}_Q = (H_1(a,Q))_2 = (\bar{i}_1, \bar{C}_1)$ be the thread *id* that performed the last unlock of a in t_1 immediately succeeding a write to a global in Q. We remark that $\bar{I}_Q \in \mathbf{I}$ holds by definition of $\mathcal{A}_1$.

  First, assume that $\mathsf{unique}\, \bar{i}_1 \wedge (i = \bar{i}_1 \vee \bar{i}_1 \in J^\sharp)$ does not hold. We have $r_{1,Q} \in \gamma_{\mathcal{R}}\left(r^\sharp_{Q,\bar{I}}\right)$ and thus by auxiliary property (a),

$$r' \subseteq r'|_Q = r_1|_Q \subseteq r' \in \gamma_{\mathcal{R}}\left(r^\sharp_{Q,\bar{I}}\right)$$

  and thus $r' \in \gamma_{\mathcal{R}}(B^\sharp(Q))$.

  Next, for the case where $\mathsf{unique}\, \bar{i}_1 \wedge (i = \bar{i}_1 \vee \bar{i}_1 \in J^\sharp)$ holds.

– $\mathsf{unique}\,\bar{i}_1 \wedge i = \bar{i}_1$ implies that it was the ego thread of t itself that performed the last write to a global protected by a in Q and the following unlock or that there was no such write. As a consequence, we have $r'\,g = r\,g$ for $g \in Q$, and remark that in this case $r\,g = L\,(a, Q)\,g$. By induction hypothesis and the auxiliary property (a) we also have $L\,(a, Q)|_Q = r|_Q \in \gamma_R(L^\sharp\,(a, Q))$.

– In the other case, we have $\gamma_{\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}}\,\bar{i} \subseteq J$, and thus the last write to a global in Q and the following unlock is join-local to t. The same argument holds here.

Thus, we have $r' \in \gamma_{\mathcal{R}}(B^\sharp(Q))$ for all $Q \in \mathcal{Q}_a$, and thus, by the property of meets of $\mathcal{R}$,

$$r' \in r'\big|_{\bar{\mathcal{G}}[a]} \subseteq \gamma_{\mathcal{R}}\left(\prod_{Q\in\mathcal{Q}_a}(B^\sharp(Q))\right) = \gamma_{\mathcal{R}}(r^\sharp_m)$$

By induction hypothesis, auxiliary property (a), and the soundness of $\sqcap$ w.r.t. the intersection of concretizations, we have

$$\begin{array}{rcl} r' & \in & (r')|_{\mathcal{X}\cup\{g\in\mathcal{G},\bar{\mathcal{M}}[g]\cap(S\cup\{a\})\neq\emptyset\}} \\ & \subseteq & r|_{\mathcal{X}\cup\{g\in\mathcal{G},\bar{\mathcal{M}}[g]\cap S\neq\emptyset\}} \cap r'|_{\bar{\mathcal{G}}[a]} \\ & \subseteq & \gamma_{\mathcal{R}}\left(r^\sharp\right) \cap \gamma_{\mathcal{R}}\left(r^\sharp_m\right) \\ & \subseteq & \gamma_{\mathcal{R}}\left(r^\sharp \sqcap r^\sharp_m\right) \\ & = & \gamma_{\mathcal{R}}\,r^{\sharp''} \subseteq \gamma_{\mathcal{R}}\,r^{\sharp'} \end{array}$$

and thus auxiliary property (a) continues to hold w.r.t. r.

- As t' and t coincide on their join-local parts (save for the new edge labeled $\mathsf{lock}(a)$ appearing in t' only), $\mathsf{jl\_joins}\ t = \mathsf{jl\_joins}\ t'$ holds. Thus, we have $J^\sharp \subseteq_j J = J'$, and as $J^\sharp \sqsubseteq J^{\sharp'}$, also $J^{\sharp'} \subseteq_j J'$.

- By the same argument, we have that for all $a \in \mathcal{M}$ and $Q \in \mathcal{Q}_a$, for which the last unlock directly following a write to a global protected by a is join-local to t', i.e., $\mathsf{jl\_unlock\_after\_write}_{a,Q}\,t' = \mathsf{unlock\_after\_write}_{a,Q}\,t'$, that unlock was also join-local to t, and $L'\,a = L\,a$. Thus, for all such mutexes and clusters, we have $L'\,(a, Q) = L\,(a, Q) \in \gamma_{\mathcal{R}}\,L^\sharp\,(a, Q) \subseteq \gamma_{\mathcal{R}}\,L^{\sharp'}\,(a, Q)$ and the auxiliary property (a) continues to hold w.r.t. L.

- Finally, using the same argument once more, we have $W' = W \subseteq W^\sharp \subseteq W^{\sharp'}$.

Altogether, $t' \in \eta'_{\mathsf{mm}}\,[u', S \cup \{a\}, A']$ holds for $t \in \eta^k\,[u, S, A_0]$ and $t_1 \in \bigcap\{\eta^k\,[a, Q, A_1] \mid Q \in \mathcal{Q}_a\}$. We conclude that the return value of $[\![([u, S, A_0], \mathsf{lock}(a), u'), A_1]\!]_{\mathsf{mm}}\,\eta^k$ is subsumed by the value $\eta'_{\mathsf{mm}}\,[u', S \cup \{a\}, A']$ and that auxiliary property (a) holds for the next approximation as well. Since the constraint causes no side-effects, the claim holds.

The proof proceeds analogously to the proof from the previous section for the other constraints. The case for unlocking a mutex once again relies on Proposition 29. We do not detail this here. □

7 Related Work

To allow for easier reference, we order our related work thematically. This leads to work being listed more than once in some instances, but makes for a more consistent overall structure.

Local Traces. To prove our thread-modular analyses correct, we rely on a trace semantics of the concurrent system. Here, we insist on maintaining the *local perspectives* of executing threads (*ego threads*) only. The idea of tracking the events possibly affecting a particular local thread configuration goes back to Lamport [74] (see also [125]), and is also used extensively for the verification of concurrent systems via separation logic [21–23, 71, 76, 90, 91, 116, 117]. Accordingly, we collect all attained local configurations of threads affecting a thread-local configuration $\bar{u}$ of an ego thread into the *local trace* reaching $\bar{u}$. A *thread-local* concrete semantics was also used in Mukherjee et al. [89] for proving the correctness of their thread-modular analyses. The semantics there, however, is based on *interleaving* and permits *stale* values for unread globals. In contrast, we consider a *partial order* of past events and explicitly exclude the *values* of globals from local traces. These are instead recovered from the local trace by searching for the *last preceding write* at the point when the value of the global is accessed. Miné [85] proposes a thread-modular concrete semantics which also is used as the basis of later work [87]. There, thread-modularity is attained by introducing auxiliary variables tracking the control locations of other threads. The local trace semantics does not introduce such auxiliary variables, and instead only expresses facts about the execution that the ego thread has observed by executing observable actions. In this way, the exact program point that some other thread is at is information that cannot be expressed in the local trace semantics — unless it is of relevance for the local perspective of the ego thread. We do not show the equivalence between our local trace semantics and an interleaving semantics; such a proof is provided by Erhard et al. [42].

(Weakly) Relational Domains and Clusters. Since its introduction, the weakly relational numerical domain of *Octagons* [81, 82] has found wide-spread application for the analysis and verification of programs [18, 33]. Since tracking relations between *all* variables may be expensive, pre-analyses have been suggested to identify *clusters* of numerical variables whose relationships may be of interest [18, 33, 59, 95]. A *dynamic* approach to decomposing relational domains into non-overlapping clusters based on learning is proposed by Singh et al. [122]. While these approaches trade (unnecessary) precision for efficiency, others try to *partition* the variables into clusters without compro-

mising precision [34, 56, 57, 94, 121, 123]. These types of clustering are orthogonal to the clustering in our relational analyses and could, perhaps, be combined with it.

Inspired by the notion of 2-decomposability, we have recently also worked on more elegant normalization for Octagons, identifying the stronger notion of 2-projectivity [109] as well as on other, non-numerical, weakly relational domains [112]. Their integration into our thread-modular analysis of globals remains an open topic, in particular for our weakly relational heap domain [52, 110], as such an integration would require developing a different notion of *protection*.

Thread-Modular Static Analyses of Values of Globals. As there is a wealth of related work, we limit ourselves to discussing automatic thread-modular approaches based on abstract interpretation endeavoring to compute values of globals in a shared memory setting. We do thus not outline work that targets similar problems but is not thread-modular (e.g., [16, 20, 63]), is not based on abstract interpretation (e.g., [16, 124]), does not target a shared memory setting (e.g., [80]), or is only concerned with other properties such as data-race freedom, determinism, or memory safety.

The thread-modular value analyses by Miné [83, 84] are based on interferences: *Synchronized interferences* are propagated from a lock to an unlock, provided that some side conditions, e.g., on the background locksets, are met. *Weak interferences* are used to account for asynchronous variable accesses. For a more detailed account of this work, see also Section 4.1.3. Stiévenart et al. [128] also use the concept of interferences [25, 83, 84] and nested fixpoints, though their abstractions are coarser, and accumulate all writes to globals, regardless of any mutexes held. Their analysis supports thread joins and dynamic thread creation — as do our analyses. The approach suggested by Vojdani [133, 134] is similar in that it also accumulates written values per global variable. However, instead of accumulating all writes, it uses *privatization*, a concept inspired by the thread-modular shape analysis by Gotsman et al. [55], to only publish those writes that are potentially visible to other threads. To this end, for each global, a set of mutexes is computed that is held whenever this global is accessed. The inferred invariants for globals thus are *resource invariants* associated with the mutex in the sense of Owicki and Gries [98], as used later in *concurrent separation logic* [96].

Turning from inherently non-relational approaches to those that allow for tracking relational abstractions of globals, Miné [85] generalizes the setting using synchronized and weak interferences to analyses that can be relational for global variables as well. There, background locksets are abandoned and the synchronized interferences thus roughly correspond to *lock invariants*, determining for each mutex a relation which holds whenever the mutex is *not* held, which is similar to our approach. While, in principle, relational abstractions of both weak interferences and the lock invariants are supported, for practical analyses, a relational abstraction only for lock invariants is proposed, while using a coarse, non-relational abstraction for the weak interferences. Our relational analysis, on the other hand, maintains, at each mutex a, only relations between variables write-protected by a. For these relations more precise results can be obtained, since they

are incorporated into the local state at locks by *meet* (while [85] uses *join*).

Monat and Miné [87] present an analysis framework which is orthogonal to our approach. It is tailored to the verification of algorithms that do not rely on explicit synchronization via mutexes such as the *Bakery* algorithm. Such an analysis is, e.g., useful when one wants to verify that an algorithm actually ensures mutual exclusion, while our analyses target higher-level code where mutexes as provided by the system are used to ensure mutual exclusion. Suzanne and Miné [130] extend [87] to handle weak memory effects (PSO, TSO) by incorporating memory buffers into the thread-local semantics. The notion of interferences is also used by Sharma and Sharma [119] for the analysis of programs under the Release/Acquire Memory Model of C11 by additionally tracking abstractions of *modification sequences* for global variables. They consider fixed finite sets of threads only, and do not deal with thread creation or joining.

Earlier work by Ferrara [48, 50] considers the (non-relational) analysis of programs under the *Happens-Before Memory Model*, which is an overapproximation of the Java memory model. The analysis description is generic in the programming language and the semantics of local computations, and focuses on the effects of concurrency, where a key step is reasoning about which values written by other threads may be *visible* to the current thread. The implementation [49, 50] supports the dynamic creation of threads as well as the dynamic creation of monitors, whereas our analyses support dynamic thread creation but assume a fixed set of mutexes.

Another line of work on thread-modular relational analysis combines different techniques: It relies on Datalog rules to model interferences in combination with abstract interpretation applied to the Data-Flow Graph [45] or the Control-Flow Graph [72] (later extended to weak memory [73]), respectively. Both settings are more specific than ours: [45] considers parameterized systems (where multiple instances of each given thread may exist), whereas [72] assumes a fixed finite set of threads. In our setting, thread *id*s are analyzed to deal with both cases appropriately. [72] treats interferences flow-sensitively, by identifying pairs of interferences that may not coexist in a single execution.

In all these approaches clusters of variables, if there are any, are predefined and not treated specially by the analysis. This is different in the thread-modular analysis proposed by Mukherjee et al. [89]. It propagates information from unlocks to locks. It is relational for the locals of each thread, and within *disjoint* subsets of globals, called *regions*. These regions must be determined beforehand and must satisfy *region-race freedom*. In contrast, the only extra a priori information required by our analysis, are the sets of (write-) protecting mutexes of globals – which can be computed during the analysis itself. The closest concept within our approach to a *region* is the set of globals jointly protected by mutexes. However, unlike regions, these sets may overlap — which the analysis explicitly exploits.

Digests. In a sequential setting, splitting control locations according to some abstraction of reaching traces, e.g., which branch was taken at a conditional, or having a

domain maintain disjunctions of abstract values based on such trace information, is a well-known technique for improving the precision of dataflow analyses [19, 37, 60] or abstract interpretation [11, 58, 79, 88, 97, 101] where it is usually referred to as *trace partitioning*. Associating different information to different copies of program points can — instead of being understood as creating new unknowns — alternatively be described as an instance of the reduced cardinal power domain [28, 32, 53], and potentially also be implemented this way. Our implementation uses this insight for unknowns associated with mutexes. The reduced cardinal power domain also lies at the foundation of the view-sensitivity framework proposed by Kim et al. [69] for a sequential setting. In this setting, the notion of *view*s as proposed there is closely related to digests. While the view-sensitivity framework has an operation similar to computing the successor digest for a unary operation, there are no operations that incorporate the histories of other threads for single-threaded programs, and thus there is also no notion of compatibility. For the analysis of multi-threaded programs, Miné [85] applies the techniques of Mauborgne and Rival [79] to *single* threads, i.e., independently of the actions of all other threads. Our approach, on the other hand, may take arbitrary properties of *local* traces into account, and thus is more general. We have recently proposed an extension of the digest framework [105], where we coin the term *concurrency-sensitivity* for digests and propose several novel digests, e.g., for a more detailed analysis of condition variables. That work builds on the local trace semantics with support for signal/wait, which is proposed in this thesis.

Thread *Id*s & MHP. While we cast thread *id*s and MHP analysis as an instance of our digest framework, there is a wealth of independent work on this specific topic: The thread *id* analysis perhaps most closely related to ours, is by Feret [46] who computes *id*s for agents in the π-calculus as abstractions of sequences of encountered create edges and in later work proposes techniques to partition threads into different classes [47]. Might and Horn [80] propose techniques to identify certain threads as *unique* in the context of a small-step abstract interpreter for the static analysis of high-order functional programs. Another line of analysis of concurrent programs deals with determining which critical events may happen in parallel (MHP) [1, 3–5, 13, 39, 92, 136] to detect programming errors such as, e.g., data races, or — in the context of compilers — for identifying opportunities for optimization. Mostly, MHP analyses are obtained as abstractions of a *global* trace semantics [41]. We apply related techniques for improving thread-modular analyses – but based on a *local* trace semantics. Like MHP analyses, we take thread creation and joining histories as well as sets of held mutexes into account.

8 Conclusion and Future Work

We have presented a concrete semantics based on local traces, which consider program execution from the *local perspective* of one special thread, the *ego thread*. In this semantics, communication happens based on executing observable and observing actions. We have described a way in which the concrete semantics can be refined by taking abstractions (*digests*) of the history of the thread into account and have proposed a limited version of such digests, which can be computed based only on the information available along the (creation-extended) *ego lane*. We have provided instances of this scheme to, e.g., compute thread *id*s in a setting with dynamic thread creation. When turning to the analysis of the values of global variables, we have first investigated thread-modular non-relational analyses based on the observation that a relatively simplistic analysis offered in the Goblint system is incomparable to a more sophisticated analysis in literature. We have offered improved versions of either of these analyses and have described how additional precision can be gained by taking *ego-lane digests* into account. We have further presented relational analyses of the values of global variables tailored to decomposable domains, where, in some instances, more precise results can be achieved by considering smaller clusters. For 2-decomposable domains, however, we have shown that the *optimal* result can already be obtained by considering clusters of size at most 2. These relational analyses also admit refinement according to digests, and we have used this framework to justify analyses that take the creation as well as joining of threads into account. We have provided a comprehensive evaluation of the precision and runtimes of all the analyses presented in this thesis, showing our analyses to be competitive with analyses from literature regarding precision and performance. Furthermore, we have provided novel principled soundness proofs w.r.t. the local trace semantics.

Subsequently, we outline some possible future directions in which the work presented in this thesis can be extended. Here, we focus mostly on discussing extensions that go beyond enhancing the framework for local traces to handle features such as procedures, pointers, or dynamically allocated memory that are already supported by our implementation but not supported by the local trace semantics yet. The list of ways in which the implementation extends on the local trace semantics provided in the introduction of Chapter 5 gives an overview of such extensions and also lists some challenges that would be encountered there. By the same token, we omit a discussion of how the analyses could be extended with context-sensitivity after enhancing the language with procedures.

Weak Memory. Unlike some earlier work by others (e.g., [48, 73, 119, 129, 130]), we do not consider the effects of weak memory for this thesis. In fact, by introducing additional m_g mutexes for each global g to enforce atomicity of accesses, we have effectively encoded *sequential consistency* into the local trace semantics. To consider weaker memory models, one would have to give up the notion of a uniquely identifiable *last* write and replace this notion with a weaker notion of *possibly* last writes. Similarly, the consistency conditions on the local traces and their corresponding orders would also need to be relaxed.

When it comes to the analyses, the situation is not the same for all of them: The non-relational analyses based on the notion of protection (e.g., Protection-based Reading) should naturally be sound also w.r.t. weaker memory models, following the results of Alglave et al. [6] on the soundness of non-relational analyses w.r.t. weak memory semantics. Since ego-lane digests only track the (creation-extended) history of the actions taken by the ego thread, and most weak memory models do still guarantee that each thread observes its own actions in sequence, the refined analyses should arguably also still be sound for weak memory models, in so far as they guarantee that created threads have a sequentially consistent view of the actions of their parents up to their creation. This claim, though, does not directly follow from the results of Alglave et al. [6] and would require a dedicated argument.

For the relational analyses, such a guarantee no longer holds in general: The digest framework can be used to encode arbitrary abstractions of the history, such as sequences of already observed actions of other threads. Exclusions based on this information will, while sound w.r.t. the local trace semantics and thus for sequential consistency, in general, be unsound for weaker memory models. The analyses without refinement, on the other hand, only relate globals that are commonly protected by at least one mutex. Provided that locks, unlocks, and thread joins act as fences, these analyses should also be sound w.r.t. at least some forms of weak memory, though dedicated proofs would need to be conducted to confirm this intuition.

Using Local Traces for Model Checking. Another interesting use case for the local trace semantics is as a basis for performing (bounded) model checking. The appeal of this approach is that the local trace semantics provides some (simple) notion of partial order reduction [99] for free, as it does not consider interleavings but only those parts of the computation history that may influence a given configuration. Basing a model checker on the local trace semantics would also allow for fine-grained interfacing with Goblint and thus benefitting from the information the abstract interpretation has established about the program to guide the model checker towards possible violations. Such an approach could be considered an instance of *co-operative verification* in the sense of Beyer [15]. We have advised two student theses [36, 44] on this idea. However, the preliminary results hinted at significant engineering challenges that would need to be overcome before the approach can be evaluated in practice.

Generalization of the Digest Framework. The digest framework, while useful as demonstrated in the previous sections, is still quite rigid in the sense that we demand that digests are computed as an abstraction of a *single* local trace, and that information associated with different digests is always kept apart. This effectively means that the analysis is required to be path-sensitive in the digest information, limiting which digests can be used in practice without running into scalability issues. At times, it would thus be convenient to further abstract digests — while still using the insights preserved after abstraction to gain precision over an analysis that does not use digests at all. We propose such an abstract version of the digest framework and elaborate on the corresponding design space as part of a forthcoming paper [105]. To allow leveraging such an extension in analyses that admit only ego-lane digests, an ego-lane version of abstract digests is conceivable as well, though such a generalization is not discussed in that paper.

Precision Improvements by Better Widening. One of the practical hurdles for achieving a higher precision is that the solvers available in the Goblint framework apply widening to global unknowns, i.e., unknowns that receive their values by side-effects, quite eagerly. This situation is exasperated by the lack of a *narrowing* iteration for these global variables in the solvers currently supported by Goblint. While some precision may be retained by employing techniques that delay the application of widening or prevent the loss of relationships of interest (e.g., threshold widening using the constants or expressions appearing in the program as thresholds), the analyses presented in this thesis would potentially greatly benefit from principled approaches that either allow for a narrowing iteration on globals (as was, to some extent, realized in some earlier variants of the SLR solvers [7, 10]), or ensuring the widening operator is applied less often. It may be of interest to design and study such extensions of the solvers, whether such strategies may be considered applications of the A^2I framework by Cousot et al. [34] (along the lines of [131]), and their practical impact on precision. We have co-advised a thesis on such techniques [126]. It resulted in approaches [127] for applying narrowing to globals in a way that is applicable not only to side-effecting constraint systems, but to other frameworks for mixed-flow sensitivity as well.

List of Figures

List of Tables

Symbols

X	Set of Unknowns
$\mathcal{C}$	Constraint System
$\mathcal{V}ars$	Set of variables
$\mathcal{X}$	Set of local variables
$\mathcal{G}$	Set of global variables
self	Local variable containing thread *id* of ego thread
ret	Local variable used for return values
$\mathcal{V}$	Set of concrete values
$\mathcal{V}_{\mathbf{tid}}$	Set of concrete values of type thread *id*
$\mathcal{V}_{\mathbf{int}}$	Set of concrete integer values
i_0	Concrete thread *id* of initial thread ($i_0 \in \mathcal{V}_{\mathbf{tid}}$)
Σ	Local program states ($\sigma : (\mathcal{X} \to \mathcal{V}) \in \Sigma$)
$\mathcal{A}ct$	Set of actions
$\mathcal{E}xp$	Expression language
$\mathcal{N}$	Set of program points
$\mathcal{E}$	Set of CFG edges ($\mathcal{E} = \mathcal{N} \times \mathcal{A}ct \times \mathcal{N}$)
$\mathcal{T}$	Set of local traces
$\mathcal{M}$	Set of mutexes
$\mathcal{U}_{\mathcal{M}}$	Set of all upwards-closed subsets of $\mathcal{M}$
$\mathcal{S}$	Set of signals
ν	Computes a new thread *id* based on the local trace reaching the predecessor node of the create action ($\nu : \mathcal{T} \to \mathcal{V}_{\mathbf{tid}}$)
$\tau_{\mathcal{A}ct}$	Function to get type of an action ($\tau_{\mathcal{A}ct} : \mathcal{A}ct \to \{\mathsf{local}, \mathsf{creating}, \mathsf{observing}, \mathsf{observable}\}$)
$\mathcal{A}ct_{\mathsf{local}}$	Set of *local* actions, i.e., of type local ($\mathcal{A}ct_{\mathsf{local}} \subset \mathcal{A}ct$)
$\mathcal{A}ct_{\mathsf{creating}}$	Set of *creating* actions
$\mathcal{A}ct_{\mathsf{observing}}$	Set of *observing* actions
$\mathcal{A}ct_{\mathsf{observable}}$	Set of *observable* actions
main	Thread template with which execution begins
sink	Extract sink from local trace ($\mathsf{sink} : \mathcal{T} \to \mathcal{N} \times \Sigma$)
last	Extract the last action (of the ego thread) from local trace ($\mathsf{last} : \mathcal{T} \to \{\mathcal{A}ct \cup \bot\}$)
id	Extract thread *id* of ego thread from local trace ($\mathsf{id} : \mathcal{T} \to \mathcal{V}_{\mathbf{tid}}$)

loc	Extract program point of sink from local trace (loc : $\mathcal{T} \to \mathcal{N}$)
init	Set of initial local traces (init $\subseteq \mathcal{T}$)
observes	Set of *observable* actions potentially observed by an *observing* action (observes : $\mathcal{A}ct \to 2^{\mathcal{A}ct}$)
$\downarrow$	$\downarrow_w (t) \in \mathcal{T}$ refers to the local sub-trace from $t \in \mathcal{T}$ that ends in node w
$[\![\cdot]\!]_{\mathcal{T}}$	$[\![e]\!]_{\mathcal{T}}$ is the right-hand side for an edge e
$[\![\cdot]\!]_{\mathcal{LT}}$	$[\![e]\!]_{\mathcal{LT}}$ is the right-hand side for an edge e in the localized constraint system
$[\![\cdot]\!]_{\mathcal{LTA}}$	$[\![e]\!]_{\mathcal{LTA}}$ is the right-hand side for an edge e in the constraint system refined according to a digest
$[\![\cdot]\!]_{\mathcal{E}xp}$	$[\![e]\!]_{\mathcal{E}xp}$ is the function for concrete evaluation of expressions
$\mathcal{A}$	Set of digests
$\mathsf{new}^\sharp_{\mathcal{A}}$	Function to compute a new digest for a thread being created along an edge originating at u being execution at u_1 from initial digest A_0 ($\mathsf{new}^\sharp_{\mathcal{A}} : \mathcal{N} \to \mathcal{N} \to \mathcal{A} \to 2^{\mathcal{A}}$)
$[\![\cdot]\!]^\sharp_{\mathcal{A}}$	Right-hand side for a digest
$\mathsf{init}^\sharp_{\mathcal{A}}$	Initial digests ($\mathsf{init}^\sharp_{\mathcal{A}} \subseteq \mathcal{A}$)
$\mathsf{compat}^\sharp_{\mathcal{A}}$	$\mathsf{compat}^\sharp_{\mathcal{A}} : \mathcal{A} \to \mathcal{A} \to \mathbf{bool}$ checks whether two digests are compatible, where the digest of the ego-thread is the first argument; Usually not symmetric.
acc	acc : $(\mathcal{A} \times \mathcal{D}_S) \to \mathcal{A} \to \mathbf{bool}$ takes the digest of the ego-thread and the abstract value associated with that program point and decides if the contributions associated with another digest are already taken into account.
$\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}$	Thread *Id*s as digests
$\mathcal{P}$	Set of pairs relating thread creation nodes with the starting points of the created threads
single	Turns a thread *id* computed as digest into an abstract thread *id* of the value domain ($\mathsf{single} : \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \to \mathcal{V}^\sharp_{\mathbf{tid}}$)
unique	Checks whether a thread *id* computed as a digest represents one concrete thread only ($\mathsf{unique} : \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \to \mathbf{bool}$)
lcu_anc	Computes the last common unique ancestor for two thread *id*s ($\mathsf{lcu\_anc} : \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \to \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \to \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}}$)

may_create	Checks whether a thread with the thread *id* provided as the first argument may start a thread with the thread *id* provided as the second argument. (may_create : $\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \to \mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \to \mathbf{bool}$)
may_run	Overapproximates whether a given thread may already be started and have preceded to the encountered set of create edges have been started based on a thread *id* and encountered create edges of the ego thread. (may_run : $(\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \times \mathcal{P}) \to (\mathcal{V}^\sharp_{\mathbf{tid},\mathcal{A}} \times \mathcal{P}) \to \mathbf{bool}$)
$d\|_Q$	$d\|_Q$ restricts an element d from an abstract domain to only talk about variables in Q
$\mathcal{V}^\sharp$	Abstract value domain
$\mathcal{V}^\sharp_\tau$	Abstract value domain for type τ
$\mathcal{V}^\sharp_{\mathbf{tid}}$	Abstract value domain (powerset domain) for variables of type thread *id*
$S_{\mathcal{V}^\sharp_{\mathbf{tid}}}$	Finite carrier set of $\mathcal{V}^\sharp_{\mathbf{tid}}$
$\bar{\mathcal{V}}^\sharp$	Non-relational domain (Mappings from variables to abstract values extended with bottom ($\mathcal{V}ars \to_\bot \mathcal{V}^\sharp$))
$[\![\cdot]\!]^\sharp_{\bar{\mathcal{V}}^\sharp}$	Abstract transfer function for assignments and guards a ($[\![a]\!]^\sharp_{\bar{\mathcal{V}}^\sharp} : \bar{\mathcal{V}}^\sharp \to \bar{\mathcal{V}}^\sharp$)
$[\![\cdot]\!]^\sharp_{\mathcal{E}xp}$	Abstract expression evaluation ($[\![e]\!]^\sharp_{\mathcal{E}xp} : \bar{\mathcal{V}}^\sharp \to \mathcal{V}^\sharp$)
$\mathcal{R}$	Relational domain
$[\![\cdot]\!]^\sharp_{\mathcal{R}}$	Abstract transfer function for assignments and guards a ($[\![a]\!]^\sharp_{\mathcal{R}} : \mathcal{R} \to \mathcal{R}$)
lift	Turns a non-relational abstract value from $\bar{\mathcal{V}}^\sharp$ into a corresponding relational abstract value from $\mathcal{R}$ (lift : $\bar{\mathcal{V}}^\sharp \to \mathcal{R}$)
unlift	Turns a relational abstract value from $\mathcal{R}$ into a corresponding non-relational abstract value from $\bar{\mathcal{V}}^\sharp$ (unlift : $\mathcal{R} \to \bar{\mathcal{V}}^\sharp$)
$[\mathcal{V}ars]_k$	Set of all subsets of $\mathcal{V}ars$ of size at most k ($\{Y \mid Y \subseteq \mathcal{V}ars, 1 \leq \|Y\| \leq k\}$)
$\bar{\mathcal{M}}[g]$	Set of mutexes protecting some global g. Contains at least m_g
$\bar{\mathcal{G}}[a]$	Globals that are protected by a mutex a
$\mathcal{Q}_a$	Clusters associated with some mutex a ($\mathcal{Q}_a \subseteq 2^{\bar{\mathcal{G}}[a]}$)
$\underline{\bot}$	$\bot$-element to denote dead code

$\nu^\sharp$	Computes a new abstract thread *id* based on the program point of the creator, the abstract state, and the program point at which program execution starts. ($\nu^\sharp : \mathcal{N} \to \bar{\mathcal{V}}^\sharp \to \mathcal{N} \to \mathcal{V}^\sharp_{\mathbf{tid}}$)
$\mathbf{V}$	Vertices of a (raw) global trace
$\mathbf{E}$	Edges of (raw) global trace
last_write	$\mathsf{last\_write}_g : \mathcal{T} \to \mathbf{E}$ extracts the last write to g appearing in a local trace
last_tl_write	$\mathsf{last\_tl\_write}_g : \mathcal{T} \to \mathbf{E}$ extracts the last *thread-local* write to g appearing in a local trace
eval_tl	$\mathsf{eval\_tl}_g : \mathcal{T} \to \mathcal{V}$ extracts the value written at the last thread-local write to g appearing in a local trace
init_v	$\mathsf{init\_v}\, t : \mathcal{T} \to \mathbf{V} \cup \{\bot\}$ extracts the node from a local trace in which the call to initMT ends if it exists, and returns $\bot$ otherwise
$L_t[\bar{u}]$	Extracts the lockset held by in local trace t at node $\bar{u}$ by the respective thread
L_t	Extracts the lockset held by the ego thread at the sink node in a local trace t
min_lockset_since	$\mathsf{min\_lockset\_since} : \mathcal{T} \to \mathbf{V} \to \mathcal{U}_\mathcal{M}$ extracts the upwards-closed set of minimal locksets the ego thread has held since a given node of the raw ego trace
last_tl_lock	$\mathsf{last\_tl\_lock}_a : \mathcal{T} \to \mathbf{V} \cup \{\bot\}$ extracts the last thread-local lock of a if it exists, and returns $\bot$ otherwise
last_tl_unlock	$\mathsf{last\_tl\_unlock}_a : \mathcal{T} \to \mathbf{V} \cup \{\bot\}$ extracts the last thread-local unlock of a if it exists, and returns $\bot$ otherwise
jl_joins	$\mathsf{jl\_joins} : \mathcal{T} \to 2^\mathbf{E}$ extracts from a local trace all calls to join that are *join-local*
jl_unlock_after_write	$\mathsf{jl\_unlock\_after\_write}_a : \mathcal{T} \to \mathbf{E} \cup \{\bot\}$ extracts, for a mutex a, the first *join-local* $\mathsf{unlock}(a)$ action that immediately succeeds the last *join-local* write to a global in $\bar{\mathcal{G}}[a]$. $\mathsf{jl\_unlock\_after\_write}_{a,Q}$ extracts this unlock for a global in the cluster $Q \in \mathcal{Q}_a$
unlock_after_write	$\mathsf{unlock\_after\_write}_a : \mathcal{T} \to \mathbf{E} \cup \{\bot\}$ extracts, for a mutex a, the first $\mathsf{unlock}(a)$ action that immediately succeeds the last write to a global in $\bar{\mathcal{G}}[a]$. $\mathsf{unlock\_after\_write}_{a,Q}$ extracts this unlock for a global in the cluster $Q \in \mathcal{Q}_a$

Bibliography

[1] Agarwal, S., Barik, R., Sarkar, V., Shyamasundar, R.K.: May-happen-in-parallel analysis of x10 programs. In: PPoPP '07, p. 183–193, ACM (2007), doi: 10.1145/1229428.1229471

[2] Airlines Electronic Engineering Committee: Avionics application software standard interface ARINC653P1-5. Tech. rep., Airlines Electronic Engineering Committee (2019)

[3] Albert, E., Flores-Montoya, A., Genaim, S.: Analysis of may-happen-in-parallel in concurrent objects. In: Giese, H., Rosu, G. (eds.) Formal Techniques for Distributed Systems - Joint 14th IFIP WG 6.1 International Conference, FMOODS 2012 and 32nd IFIP WG 6.1 International Conference, FORTE 2012, Stockholm, Sweden, June 13-16, 2012. Proceedings, Lecture Notes in Computer Science, vol. 7273, pp. 35–51, Springer (2012), doi: 10.1007/978-3-642-30793-5_3

[4] Albert, E., Genaim, S., Gordillo, P.: May-happen-in-parallel analysis for asynchronous programs with inter-procedural synchronization. In: Blazy, S., Jensen, T.P. (eds.) Static Analysis - 22nd International Symposium, SAS 2015, Saint-Malo, France, September 9-11, 2015, Proceedings, Lecture Notes in Computer Science, vol. 9291, pp. 72–89, Springer (2015), doi: 10.1007/978-3-662-48288-9_5

[5] Albert, E., Genaim, S., Gordillo, P.: May-happen-in-parallel analysis with returned futures. In: D'Souza, D., Kumar, K.N. (eds.) Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings, Lecture Notes in Computer Science, vol. 10482, pp. 42–58, Springer (2017), doi: 10.1007/978-3-319-68167-2_3

[6] Alglave, J., Kroening, D., Lugton, J., Nimal, V., Tautschnig, M.: Soundness of data flow analyses for weak memory models. In: Yang, H. (ed.) Programming Languages and Systems - 9th Asian Symposium, APLAS 2011, Kenting, Taiwan, December 5-7, 2011. Proceedings, Lecture Notes in Computer Science, vol. 7078, pp. 272–288, Springer (2011), doi: 10.1007/978-3-642-25318-8_21

[7] Amato, G., Scozzari, F., Seidl, H., Apinis, K., Vojdani, V.: Efficiently intertwining widening and narrowing. Sci. Comput. Program. **120**, 1–24 (2016), doi: 10.1016/J.SCICO.2015.12.005

[8] Apinis, K.: Frameworks for analyzing multi-threaded C. Ph.D. thesis, Technical University Munich (2014), URL `https://nbn-resolving.org/urn:nbn:de:bvb:91-diss-20140606-1189191-0-9`

[9] Apinis, K., Seidl, H., Vojdani, V.: Side-effecting constraint systems: a swiss army knife for program analysis. In: APLAS '12, pp. 157–172, Springer (2012)

[10] Apinis, K., Seidl, H., Vojdani, V.: How to combine widening and narrowing for non-monotonic systems of equations. In: Boehm, H., Flanagan, C. (eds.) ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013, pp. 377–386, ACM (2013), doi: 10.1145/2491956.2462190

[11] Babu M, C., Lemerre, M., Bardin, S., Marion, J.Y.: Trace partitioning as an optimization problem. In: Static Analysis - 31st International Symposium, SAS 2024, Pasadena, CA, USA, October 20-22, 2024, Proceedings, Lecture Notes in Computer Science, Springer (2024)

[12] Ballou, K., Sherman, E.: Minimally comparing relational abstract domains. In: André, É., Sun, J. (eds.) Automated Technology for Verification and Analysis - 21st International Symposium, ATVA 2023, Singapore, October 24-27, 2023, Proceedings, Part II, Lecture Notes in Computer Science, vol. 14216, pp. 159–175, Springer (2023), doi: 10.1007/978-3-031-45332-8_8

[13] Barik, R.: Efficient computation of may-happen-in-parallel information for concurrent Java programs. In: LCPC '06, vol. 4339 LNCS, pp. 152–169, Springer (2006), doi: 10.1007/978-3-540-69330-7_11

[14] Beyer, D.: Software verification: 10th comparative evaluation (sv-comp 2021). In: Groote, J.F., Larsen, K.G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, pp. 401–422, Springer International Publishing, Cham (2021), ISBN 978-3-030-72013-1

[15] Beyer, D.: Cooperative verification: Towards reliable safety-critical systems (invited talk). In: Artho, C., Ölveczky, P.C. (eds.) Proceedings of the 8th ACM SIGPLAN International Workshop on Formal Techniques for Safety-Critical Systems, FTSCS 2022, Auckland, New Zealand, 7 December 2022, pp. 1–2, ACM (2022), doi: 10.1145/3563822.3572548

[16] Beyer, D., Friedberger, K.: A light-weight approach for verifying multi-threaded programs with cpachecker. In: Bouda, J., Holík, L., Kofron, J., Strejcek, J., Rambousek, A. (eds.) Proceedings 11th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, MEMICS 2016, Telč, Czech Republic, 21st-23rd October 2016, EPTCS, vol. 233, pp. 61–71 (2016), doi: 10.4204/EPTCS.233.6

[17] Beyer, D., Löwe, S., Wendler, P.: Reliable benchmarking: requirements and solutions. Int. J. Softw. Tools Technol. Transf. **21**(1), 1–29 (2019), doi: 10.1007/S10009-017-0469-Y

[18] Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: A static analyzer for large safety-critical software. In: Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation, p. 196–207, PLDI '03, Association for Computing Machinery, New York, NY, USA (2003), ISBN 1581136625, doi: 10.1145/781131.781153

[19] Bodík, R., Gupta, R., Soffa, M.L.: Refining data flow information using infeasible paths. SIGSOFT Softw. Eng. Notes **22**(6), 361–377 (1997), ISSN 0163-5948, doi: 10.1145/267896.267921

[20] Botbol, V., Chailloux, E., Gall, T.L.: Static analysis of communicating processes using symbolic transducers. In: Bouajjani, A., Monniaux, D. (eds.) Verification, Model Checking, and Abstract Interpretation - 18th International Conference, VMCAI 2017, Paris, France, January 15-17, 2017, Proceedings, Lecture Notes in Computer Science, vol. 10145, pp. 73–90, Springer (2017), doi: 10.1007/978-3-319-52234-0_5

[21] Brookes, S.: A grainless semantics for parallel programs with shared mutable data. In: Escardó, M.H., Jung, A., Mislove, M.W. (eds.) Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics, MFPS 2005, Birmingham, UK, May 18-21, 2005, Electronic Notes in Theoretical Computer Science, vol. 155, pp. 277–307, Elsevier (2005), doi: 10.1016/J.ENTCS.2005.11.060

[22] Brookes, S.: A semantics for concurrent separation logic. Theoretical Computer Science **375**(1-3), 227–270 (2007), doi: 10.1016/j.tcs.2006.12.034

[23] Brookes, S.: On grainless footprint semantics for shared-memory programs. In: Jacobs, B., Silva, A., Staton, S. (eds.) Proceedings of the 30th Conference on the Mathematical Foundations of Programming Semantics, MFPS 2014, Ithaca, NY, USA, June 12-15, 2014, Electronic Notes in Theoretical Computer Science, vol. 308, pp. 65–86, Elsevier (2014), doi: 10.1016/J.ENTCS.2014.10.005

[24] Butenhof, D.R.: Programming with POSIX threads. Addison-Wesley Professional (1993)

[25] Carre, J., Hymans, C.: From single-thread to multithreaded: An efficient static analysis algorithm. CoRR **abs/0910.5833** (2009), URL `http://arxiv.org/abs/0910.5833`

[26] Chase, D.R., Wegman, M.N., Zadeck, F.K.: Analysis of pointers and structures. In: Fischer, B.N. (ed.) Proceedings of the ACM SIGPLAN'90 Conference on Programming Language Design and Implementation (PLDI), White Plains, New York, USA, June 20-22, 1990, pp. 296–310, ACM (1990), doi: 10.1145/93542.93585

[27] Chen, L., Liu, J., Miné, A., Kapur, D., Wang, J.: An abstract domain to infer octagonal constraints with absolute value. In: Müller-Olm, M., Seidl, H. (eds.) Static Analysis - 21st International Symposium, SAS 2014, Munich, Germany, September 11-13, 2014. Proceedings, Lecture Notes in Computer Science, vol. 8723, pp. 101–117, Springer (2014), doi: 10.1007/978-3-319-10936-7_7

[28] Cortesi, A., Costantini, G., Ferrara, P.: A survey on product operators in abstract interpretation. In: Banerjee, A., Danvy, O., Doh, K., Hatcliff, J. (eds.) Semantics, Abstract Interpretation, and Reasoning about Programs: Essays Dedicated to David A. Schmidt on the Occasion of his Sixtieth Birthday, Manhattan, Kansas, USA, 19-20th September 2013, EPTCS, vol. 129, pp. 325–336 (2013), doi: 10.4204/EPTCS.129.19

[29] Cousot, P.: Principles of abstract interpretation. MIT Press (2021)

[30] Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Graham, R.M., Harrison, M.A., Sethi, R. (eds.) Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977, pp. 238–252, ACM (1977), doi: 10.1145/512950.512973

[31] Cousot, P., Cousot, R.: Automatic synthesis of optimal invariant assertions: Mathematical foundations. In: Low, J. (ed.) Proceedings of the 1977 Symposium on Artificial Intelligence and Programming Languages, USA, August 15-17, 1977, pp. 1–12, ACM (1977), doi: 10.1145/800228.806926

[32] Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Aho, A.V., Zilles, S.N., Rosen, B.K. (eds.) Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages, San Antonio, Texas, USA, January 1979, pp. 269–282, ACM Press (1979), doi: 10.1145/567752.567778

[33] Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Rival, X.: Why does astrée scale up? Form. Methods Syst. Des. **35**(3), 229–264 (2009), ISSN 0925-9856, doi: 10.1007/s10703-009-0089-6

[34] Cousot, P., Giacobazzi, R., Ranzato, F.: A^2i: abstract^2 interpretation. Proc. ACM Program. Lang. **3**(POPL), 42:1–42:31 (2019), doi: 10.1145/3290355

[35] Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: Aho, A.V., Zilles, S.N., Szymanski, T.G. (eds.) Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, Tucson, Arizona, USA, January 1978, pp. 84–96, ACM Press (1978), doi: 10.1145/512760.512770

[36] Danylchenko, S.: Witnessing Violations of Safety Properties for Concurrent Programs. Bachelor's thesis, Technical University of Munich (2023)

[37] Das, M., Lerner, S., Seigle, M.: ESP: path-sensitive program verification in polynomial time. In: Knoop, J., Hendren, L.J. (eds.) Proceedings of the 2002 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Berlin, Germany, June 17-19, 2002, pp. 57–68, ACM (2002), doi: 10.1145/512529.512538

[38] De, A., D'Souza, D., Nasre, R.: Dataflow analysis for datarace-free programs. In: ESOP, vol. LNCS 6602, pp. 196–215, Springer (2011), doi: 10.1007/978-3-642-19718-5_11

[39] Di, P., Sui, Y., Ye, D., Xue, J.: Region-based may-happen-in-parallel analysis for C programs. In: ICPP, pp. 889–898, IEEE (2015), ISBN 978-1-4673-7587-0, doi: 10.1109/ICPP.2015.98

[40] Durumeric, Z., Wustrow, E., Halderman, J.A.: ZMap: Fast internet-wide scanning and its security applications. In: 22nd USENIX Security Symposium (2013)

[41] Dwyer, M.B., Clarke, L.A.: Data flow analysis for verifying properties of concurrent programs. ACM SIGSOFT Software Engineering Notes **19**(5), 62–75 (1994), doi: 10.1145/195274.195295

[42] Erhard, J., Bentele, M., Heizmann, M., Klumpp, D., Saan, S., Schüssele, F., Schwarz, M., Seidl, H., Tilscher, S., Vojdani, V.: Correctness witnesses for concurrent programs: Bridging the semantic divide with ghosts. In: Verification, Model Checking, and Abstract Interpretation, Lecture Notes in Computer Science (2025), URL `https://arxiv.org/abs/2411.16612`, to appear

[43] Erhard, J., Schinabeck, J.F., Schwarz, M., Seidl, H.: When to stop going down the rabbit hole: Taming context-sensitivity on the fly. In: Monat, R., Rubio-González, C. (eds.) Proceedings of the 13th ACM SIGPLAN International Workshop on the State Of the Art in Program Analysis, SOAP 2024, Copenhagen, Denmark, 25 June 2024, pp. 35–44, ACM (2024), doi: 10.1145/3652588.3663321

[44] Faddeenkov, O.: Local Traces: Partial Order Reduction for Free. Master's thesis, Technical University of Munich (2023)

[45] Farzan, A., Kincaid, Z.: Verification of parameterized concurrent programs by modular reasoning about data and control. In: Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, p. 297–308, POPL '12, Association for Computing Machinery, New York, NY, USA (2012), ISBN 9781450310833, doi: 10.1145/2103656.2103693

[46] Feret, J.: Abstract interpretation of mobile systems. J. Log. Algebraic Methods Program. **63**(1), 59–130 (2005), doi: 10.1016/j.jlap.2004.01.005

[47] Feret, J.: Partitioning the threads of a mobile system. CoRR **abs/0802.0188** (2008), URL `http://arxiv.org/abs/0802.0188`

[48] Ferrara, P.: Static analysis via abstract interpretation of the happens-before memory model. In: Beckert, B., Hähnle, R. (eds.) Tests and Proofs - 2nd International Conference, TAP 2008, Prato, Italy, April 9-11, 2008. Proceedings, Lecture Notes in Computer Science, vol. 4966, pp. 116–133, Springer (2008), doi: 10.1007/978-3-540-79124-9_9

[49] Ferrara, P.: Checkmate: A generic static analyzer of java multithreaded programs. In: Hung, D.V., Krishnan, P. (eds.) Seventh IEEE International Conference on Software Engineering and Formal Methods, SEFM 2009, Hanoi, Vietnam, 23-27 November 2009, pp. 169–178, IEEE Computer Society (2009), doi: 10.1109/SEFM.2009.20

[50] Ferrara, P.: Static analysis via abstract interpretation of multithreaded programs. Ph.D. thesis, École Polytechnique, Palaiseau, France (2009), URL `https://tel.archives-ouvertes.fr/tel-00417502`

[51] Fulara, J., Durnoga, K., Jakubczyk, K., Schubert, A.: Relational abstract domain of weighted hexagons. Electron. Notes Theor. Comput. Sci. **267**(1), 59–72 (2010), doi: 10.1016/j.entcs.2010.09.006

[52] Ghidini, R., Erhard, J., Schwarz, M., Seidl, H.: C-2po: A weakly relational pointer domain: "these are not the memory cells you are looking for". In: Proceedings of the 10th ACM SIGPLAN International Workshop on Numerical and Symbolic Abstract Domains, pp. 2–9, NSAD '24, Association for Computing Machinery, New York, NY, USA (2024), ISBN 9798400712173, doi: 10.1145/3689609.3689994

[53] Giacobazzi, R., Ranzato, F.: The reduced relative power operation on abstract domains. Theor. Comput. Sci. **216**(1-2), 159–211 (1999), doi: 10.1016/S0304-3975(98)00194-7

[54] Gierz, G., Hofmann, K.H., Keimel, K., Lawson, J.D., Mislove, M.W., Scott, D.S.: Topology of Continuous Lattices: The Scott Topology, pp. 97–140. Springer Berlin Heidelberg, Berlin, Heidelberg (1980), ISBN 978-3-642-67678-9, doi: 10.1007/978-3-642-67678-9_3

[55] Gotsman, A., Berdine, J., Cook, B., Sagiv, M.: Thread-modular shape analysis. In: PLDI '07, pp. 266–277, ACM (2007), doi: 10.1145/1250734.1250765

[56] Halbwachs, N., Merchat, D., Gonnord, L.: Some ways to reduce the space dimension in polyhedra computations. Formal Methods in System Design **29**(1), 79–95 (2006), ISSN 1572-8102, doi: 10.1007/s10703-006-0013-2

[57] Halbwachs, N., Merchat, D., Parent-Vigouroux, C.: Cartesian factoring of polyhedra in linear relation analysis. In: Cousot, R. (ed.) Static Analysis, pp. 355–365, Springer Berlin Heidelberg, Berlin, Heidelberg (2003), ISBN 978-3-540-44898-3

[58] Handjieva, M., Tzolovski, S.: Refining static analyses by trace-based partitioning using control flow. In: Levi, G. (ed.) Static Analysis, pp. 200–214, Springer Berlin Heidelberg, Berlin, Heidelberg (1998), ISBN 978-3-540-49727-1

[59] Heo, K., Oh, H., Yang, H.: Learning a variable-clustering strategy for octagon from labeled data generated by a static analysis. In: Rival, X. (ed.) Static Analysis, pp. 237–256, Springer Berlin Heidelberg, Berlin, Heidelberg (2016), ISBN 978-3-662-53413-7

[60] Holley, L.H., Rosen, B.K.: Qualified data flow problems. In: Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, p. 68–82, POPL '80, Association for Computing Machinery, New York, NY, USA (1980), ISBN 0897910117, doi: 10.1145/567446.567454

[61] Hong, J., Ryu, S.: Concrat: An automatic c-to-rust lock API translator for concurrent programs. In: 45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023, pp. 716–728, IEEE (2023), doi: 10.1109/ICSE48619.2023.00069

[62] Howe, J.M., King, A.: Logahedra: A new weakly relational domain. In: Liu, Z., Ravn, A.P. (eds.) Automated Technology for Verification and Analysis, 7th International Symposium, ATVA 2009, Macao, China, October 14-16, 2009. Proceedings, Lecture Notes in Computer Science, vol. 5799, pp. 306–320, Springer (2009), doi: 10.1007/978-3-642-04761-9_23

[63] Jeannet, B.: Relational interprocedural verification of concurrent programs. Softw. Syst. Model. **12**(2), 285–306 (2013), doi: 10.1007/S10270-012-0230-7

[64] Jeannet, B., Miné, A.: Apron: A library of numerical abstract domains for static analysis. In: Bouajjani, A., Maler, O. (eds.) Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings, LNCS, vol. 5643, pp. 661–667, Springer (2009), doi: 10.1007/978-3-642-02658-4_52

[65] Journault, M., Miné, A., Ouadjaout, A.: An abstract domain for trees with numeric relations. In: Caires, L. (ed.) Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Lecture Notes in Computer Science, vol. 11423, pp. 724–751, Springer (2019), doi: 10.1007/978-3-030-17184-1_26

[66] Kahlon, V., Ivančić, F., Gupta, A.: Reasoning about threads communicating via locks. In: CAV '05, vol. LNCS 3576, pp. 505–518, Springer (2005), doi: 10.1007/11513988_49

[67] Kahlon, V., Yang, Y., Sankaranarayanan, S., Gupta, A.: Fast and accurate static data-race detection for concurrent programs. In: CAV '07, vol. LNCS 4590, pp. 226–239, Springer (2007), doi: 10.1007/978-3-540-73368-3_26

[68] Karr, M.: Affine relationships among variables of a program. Acta Informatica **6**, 133–151 (1976), doi: 10.1007/BF00268497

[69] Kim, S., Rival, X., Ryu, S.: A theoretical foundation of sensitivity in an abstract interpretation framework. ACM Trans. Program. Lang. Syst. **40**(3), 13:1–13:44 (2018), doi: 10.1145/3230624

[70] Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: Frama-c: A software analysis perspective. Formal Aspects Comput. **27**(3), 573–609 (2015), doi: 10.1007/S00165-014-0326-7

[71] Krebbers, R., Jung, R., Bizjak, A., Jourdan, J., Dreyer, D., Birkedal, L.: The essence of higher-order concurrent separation logic. In: Yang, H. (ed.) Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Lecture Notes in Computer Science, vol. 10201, pp. 696–723, Springer (2017), doi: 10.1007/978-3-662-54434-1_26

[72] Kusano, M., Wang, C.: Flow-sensitive composition of thread-modular abstract interpretation. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, p. 799–809, FSE 2016, Association for Computing Machinery, New York, NY, USA (2016), ISBN 9781450342186, doi: 10.1145/2950290.2950291

[73] Kusano, M., Wang, C.: Thread-modular static analysis for relaxed memory models. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, p. 337–348, ESEC/FSE 2017, Association for Computing Machinery, New York, NY, USA (2017), ISBN 9781450351058, doi: 10.1145/3106237.3106243

[74] Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM **21**(7), 558–565 (1978)

[75] Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Trans. Computers **28**(9), 690–691 (1979), doi: 10.1109/TC.1979.1675439

[76] Ley-Wild, R., Nanevski, A.: Subjective auxiliary state for coarse-grained concurrency. In: Giacobazzi, R., Cousot, R. (eds.) The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013, pp. 561–574, ACM (2013), doi: 10.1145/2429069.2429134

[77] Li, H.: Minimap2: pairwise alignment for nucleotide sequences. Bioinform. **34**(18), 3094–3100 (2018), doi: 10.1093/BIOINFORMATICS/BTY191

[78] Logozzo, F., Fähndrich, M.: Pentagons: A weakly relational abstract domain for the efficient validation of array accesses. In: Proceedings of the 2008 ACM Symposium on Applied Computing, p. 184–188, SAC '08, Association for Computing Machinery, New York, NY, USA (2008), ISBN 9781595937537, doi: 10.1145/1363686.1363736

[79] Mauborgne, L., Rival, X.: Trace partitioning in abstract interpretation based static analyzers. In: Sagiv, M. (ed.) Programming Languages and Systems, pp. 5–20, Springer Berlin Heidelberg, Berlin, Heidelberg (2005), ISBN 978-3-540-31987-0

[80] Might, M., Horn, D.V.: A family of abstract interpretations for static analysis of concurrent higher-order programs. In: Yahav, E. (ed.) Static Analysis - 18th International Symposium, SAS 2011, Venice, Italy, September 14-16, 2011. Proceedings, Lecture Notes in Computer Science, vol. 6887, pp. 180–197, Springer (2011), doi: 10.1007/978-3-642-23702-7_16

[81] Miné, A.: The octagon abstract domain. In: WCRE' 01, p. 310, IEEE Computer Society (2001), doi: 10.1109/WCRE.2001.957836

[82] Miné, A.: The octagon abstract domain. Higher Order Symbol. Comput. **19**(1), 31–100 (2006), ISSN 1388-3690, doi: 10.1007/s10990-006-8609-1

[83] Miné, A.: Static analysis of run-time errors in embedded critical parallel C programs. In: Barthe, G. (ed.) Programming Languages and Systems - 20th European Symposium on Programming, ESOP 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings, Lecture Notes in Computer Science, vol. 6602, pp. 398–418, Springer (2011), doi: 10.1007/978-3-642-19718-5_21

[84] Miné, A.: Static analysis of run-time errors in embedded real-time parallel C programs. Logical Methods in Computer Science **8**(1), 1–63 (2012), doi: 10.2168/LMCS-8(1:26)2012

[85] Miné, A.: Relational thread-modular static value analysis by abstract interpretation. In: VMCAI '14, vol. 8318 LNCS, pp. 39–58, Springer (2014), doi: 10.1007/978-3-642-54013-4_3

[86] Miné, A.: Tutorial on static inference of numeric invariants by abstract interpretation. Found. Trends Program. Lang. **4**(3-4), 120–372 (2017), doi: 10.1561/2500000034

[87] Monat, R., Miné, A.: Precise thread-modular abstract interpretation of concurrent programs using relational interference abstractions. In: VMCAI '17, vol. 10145 LNCS, pp. 386–404, Springer (2017), doi: 10.1007/978-3-319-52234-0_21

[88] Montagu, B., Jensen, T.: Trace-based control-flow analysis. In: PLDI '21, p. 482–496, ACM (2021), doi: 10.1145/3453483.3454057

[89] Mukherjee, S., Padon, O., Shoham, S., D'Souza, D., Rinetzky, N.: Thread-local semantics and its efficient sequential abstractions for race-free programs. In: SAS '17, vol. LNCS 10422, pp. 253–276, Springer (2017), doi: 10.1007/978-3-319-66706-5_13

[90] Nanevski, A., Banerjee, A., Delbianco, G.A., Fábregas, I.: Specifying concurrent programs in separation logic: Morphisms and simulations. PACMPL **3**(OOPSLA), 1–30 (2019), doi: 10.1145/3360587

[91] Nanevski, A., Ley-Wild, R., Sergey, I., Delbianco, G.A.: Communicating state transition systems for fine-grained concurrent resources. In: ESOP '14, vol. LNCS 8410, pp. 290–310, Springer (2014), doi: 10.1007/978-3-642-54833-8_16

[92] Naumovich, G., Avrunin, G.S., Clarke, L.A.: An efficient algorithm for computing mhp information for concurrent Java programs. In: ESEC/FSE '99, vol. 1687 LNCS, pp. 338–354, Springer (1999), doi: 10.1007/3-540-48166-4_21

[93] Necula, G.C., McPeak, S., Rahul, S.P., Weimer, W.: CIL: intermediate language and tools for analysis and transformation of C programs. In: Horspool, R.N. (ed.) Compiler Construction, 11th International Conference, CC 2002, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002, Proceedings, Lecture Notes in Computer Science, vol. 2304, pp. 213–228, Springer (2002), doi: 10.1007/3-540-45937-5_16

[94] Oh, H., Heo, K., Lee, W., Lee, W., Park, D., Kang, J., Yi, K.: Global sparse analysis framework. ACM Trans. Program. Lang. Syst. **36**(3) (2014), ISSN 0164-0925, doi: 10.1145/2590811

[95] Oh, H., Lee, W., Heo, K., Yang, H., Yi, K.: Selective x-sensitive analysis guided by impact pre-analysis. ACM Trans. Program. Lang. Syst. **38**(2) (2015), ISSN 0164-0925, doi: 10.1145/2821504

[96] O'Hearn, P.W.: Resources, concurrency, and local reasoning. Theoretical Computer Science **375**(1), 271–307 (2007), doi: 10.1016/j.tcs.2006.12.035

[97] Olesen, M.C., Hansen, R.R., Larsen, K.G.: An automata-based approach to trace partitioned abstract interpretation. In: Probst, C.W., Hankin, C., Hansen, R.R. (eds.) Semantics, Logics, and Calculi - Essays Dedicated to Hanne Riis Nielson and Flemming Nielson on the Occasion of Their 60th Birthdays, Lecture Notes in Computer Science, vol. 9560, pp. 88–110, Springer (2016), doi: 10.1007/978-3-319-27810-0_5

[98] Owicki, S.S., Gries, D.: Verifying properties of parallel programs: An axiomatic approach. Commun. ACM **19**(5), 279–285 (1976), doi: 10.1145/360051.360224

[99] Peled, D.: Partial-order reduction. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 173–190, Springer (2018), doi: 10.1007/978-3-319-10575-8_6

[100] Péron, M., Halbwachs, N.: An abstract domain extending difference-bound matrices with disequality constraints. In: Cook, B., Podelski, A. (eds.) Verification, Model Checking, and Abstract Interpretation, 8th International Conference, VMCAI 2007, Nice, France, January 14-16, 2007, Proceedings, Lecture Notes in Computer Science, vol. 4349, pp. 268–282, Springer (2007), doi: 10.1007/978-3-540-69738-1_20

[101] Rival, X., Mauborgne, L.: The trace partitioning abstract domain. ACM Trans. Program. Lang. Syst. **29**(5) (2007), ISSN 0164-0925, doi: 10.1145/1275497.1275501

[102] Rival, X., Yi, K.: Introduction to static analysis: an abstract interpretation perspective. Mit Press (2020)

[103] Saan, S., Erhard, J., Schwarz, M., Bozhilov, S., Holter, K., Tilscher, S., Vojdani, V., Seidl, H.: Goblint: Abstract interpretation for memory safety and termination (competition contribution). In: TACAS '24, Springer (2024)

[104] Schwarz, M.: Thread-Modular Abstract Interpretation: The Local Perspective (2024), doi: 10.5281/zenodo.14231262

[105] Schwarz, M., Erhard, J.: The digest framework: concurrency-sensitivity for abstract interpretation. Int. J. Softw. Tools Technol. Transf. **26**(6), 727–746 (2024), doi: 10.1007/S10009-024-00773-Y

[106] Schwarz, M., Erhard, J., Vojdani, V., Saan, S., Seidl, H.: When long jumps fall short: Control-flow tracking and misuse detection for non-local jumps in C. In: Ferrara, P., Hadarean, L. (eds.) Proceedings of the 12th ACM SIGPLAN International Workshop on the State Of the Art in Program Analysis, SOAP 2023, Orlando, FL, USA, 17 June 2023, pp. 20–26, ACM (2023), doi: 10.1145/3589250.3596140

[107] Schwarz, M., Saan, S., Seidl, H., Apinis, K., Erhard, J., Vojdani, V.: Improving thread-modular abstract interpretation. In: Dragoi, C., Mukherjee, S., Namjoshi, K.S. (eds.) Static Analysis - 28th International Symposium, SAS 2021, Chicago, IL, USA, October 17-19, 2021, Proceedings, Lecture Notes in Computer Science, vol. 12913, pp. 359–383, Springer (2021)

[108] Schwarz, M., Saan, S., Seidl, H., Erhard, J., Vojdani, V.: Clustered relational thread-modular abstract interpretation with local traces. In: Wies, T. (ed.) Programming Languages and Systems - 32nd European Symposium on Programming, ESOP 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings, Lecture Notes in Computer Science, vol. 13990, pp. 28–58, Springer (2023)

[109] Schwarz, M., Seidl, H.: Octagons revisited - elegant proofs and simplified algorithms. In: Hermenegildo, M.V., Morales, J.F. (eds.) Static Analysis - 30th International Symposium, SAS 2023, Cascais, Portugal, October 22-24, 2023, Proceedings, Lecture Notes in Computer Science, vol. 14284, pp. 485–507, Springer (2023), doi: 10.1007/978-3-031-44245-2_21

[110] Seidl, H., Erhard, J., Schwarz, M., Tilscher, S.: 2-pointer logic. In: Kiefer, S., Kretínský, J., Kucera, A. (eds.) Taming the Infinities of Concurrency - Essays Dedicated to Javier Esparza on the Occasion of His 60th Birthday, Lecture Notes in Computer Science, vol. 14660, pp. 281–307, Springer (2024), doi: 10.1007/978-3-031-56222-8_16

[111] Seidl, H., Erhard, J., Tilscher, S., Schwarz, M.: Non-numerical weakly relational domains. International Journal on Software Tools for Technology Transfer pp. 1–16 (2024), doi: 10.1007/s10009-024-00755-0

[112] Seidl, H., Erhard, J., Tilscher, S., Schwarz, M.: Non-numerical weakly relational domains. Int. J. Softw. Tools Technol. Transf. **26**(4), 479–494 (2024), doi: 10.1007/S10009-024-00755-0

[113] Seidl, H., Vene, V., Müller-Olm, M.: Global invariants for analysing multi-threaded applications. In: Proceedings — Estonian Academy of Sciences Physics Mathematics, vol. 52, pp. 413–436, Estonian Academy Publishers (2003)

[114] Seidl, H., Vogler, R.: Three improvements to the top-down solver. Math. Struct. Comput. Sci. **31**(9), 1090–1134 (2021), doi: 10.1017/S0960129521000499

[115] Seidl, H., Wilhelm, R., Hack, S.: Compiler Design - Analysis and Transformation. Springer (2012), ISBN 978-3-642-17547-3, doi: 10.1007/978-3-642-17548-0

[116] Sergey, I., Nanevski, A., Banerjee, A.: Mechanized verification of fine-grained concurrent programs. In: PLDI '15, pp. 77–87, ACM (2015), doi: 10.1145/2737924.2737964

[117] Sergey, I., Nanevski, A., Banerjee, A.: Mechanized verification of fine-grained concurrent programs. In: Grove, D., Blackburn, S.M. (eds.) Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015, pp. 77–87, ACM (2015), doi: 10.1145/2737924.2737964

[118] Sharir, M., Pnueli, A.: Two approaches to interprocedural data flow analysis. New York University. Courant Institute of Mathematical Sciences (1978)

[119] Sharma, D., Sharma, S.: Thread-modular analysis of release-acquire concurrency. In: Dragoi, C., Mukherjee, S., Namjoshi, K.S. (eds.) Static Analysis - 28th International Symposium, SAS 2021, Chicago, IL, USA, October 17-19, 2021, Proceedings, LNCS, vol. 12913, pp. 384–404, Springer (2021), doi: 10.1007/978-3-030-88806-0_19

[120] Simon, A., King, A., Howe, J.M.: Two variables per linear inequality as an abstract domain. In: Leuschel, M. (ed.) Logic Based Program Synthesis and Transformation, 12th International Workshop, LOPSTR 2002, Madrid, Spain, September 17-20,2002, Revised Selected Papers, LNCS, vol. 2664, pp. 71–89, Springer (2002), doi: 10.1007/3-540-45013-0_7

[121] Singh, G., Püschel, M., Vechev, M.: Fast polyhedra abstract domain. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, p. 46–59, POPL 2017, Association for Computing Machinery, New York, NY, USA (2017), ISBN 9781450346603, doi: 10.1145/3009837.3009885

[122] Singh, G., Püschel, M., Vechev, M.: Fast numerical program analysis with reinforcement learning. In: Chockler, H., Weissenbacher, G. (eds.) Computer Aided Verification, pp. 211–229, Springer International Publishing, Cham (2018), ISBN 978-3-319-96145-3

[123] Singh, G., Püschel, M., Vechev, M.: A practical construction for decomposing numerical abstract domains. Proc. ACM Program. Lang. **2**(POPL) (2018), doi: 10.1145/3158143

[124] Sinha, N., Wang, C.: On interference abstractions. In: Ball, T., Sagiv, M. (eds.) Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011, pp. 423–434, ACM (2011), doi: 10.1145/1926385.1926433

[125] van Steen, M., Tanenbaum, A.S.: Distributed Systems. distributed-systems.net, 3rd edn. (2017)

[126] Stemmler, F.: Combating Precision Loss in Side-Effecting Constraint Systems. Master's thesis, Technical University of Munich (2024)

[127] Stemmler, F., Schwarz, M., Erhard, J., Tilscher, S., Seidl, H.: Taking out the toxic trash: Recovering precision in mixed flow-sensitive static analyses. Proc. ACM Program. Lang. **9**(PLDI) (2025), doi: 10.1145/3729297

[128] Stiévenart, Q., Nicolay, J., Meuter, W.D., Roover, C.D.: A general method for rendering static analyses for diverse concurrency models modular. J. Syst. Softw. **147**, 17–45 (2019), doi: 10.1016/J.JSS.2018.10.001

[129] Suzanne, T., Miné, A.: From array domains to abstract interpretation under store-buffer-based memory models. In: SAS '16, vol. LNCS 9837, pp. 469–488, Springer (2016), doi: 10.1007/978-3-662-53413-7_23

[130] Suzanne, T., Miné, A.: Relational thread-modular abstract interpretation under relaxed memory models. In: APLAS '18, vol. LNCS 11275, pp. 109–128, Springer (2018), doi: 10.1007/978-3-030-02768-1_6

[131] Tilscher, S., Stade, Y., Schwarz, M., Vogler, R., Seidl, H.: The top-down solver—an exercise in A^2I. In: Arceri, V., Cortesi, A., Ferrara, P., Olliaro, M. (eds.) Challenges of Software Verification, pp. 157–179, Springer Nature Singapore, Singapore (2023), ISBN 978-981-19-9601-6, doi: 10.1007/978-981-19-9601-6_9

[132] Valve Corporation: Steam hardware & software survey: November 2024: Pc physical cpu details (2024), URL `https://store.steampowered.com/hwsurvey/cpus/`, accessed version (as of 12/11/2024) archived at `https://archive.ph/AOr4K`

[133] Vojdani, V.: Static Data Race Analysis of Heap-Manipulating C Programs. Ph.D. thesis, University of Tartu. (2010)

[134] Vojdani, V., Apinis, K., Rõtov, V., Seidl, H., Vene, V., Vogler, R.: Static Race Detection for Device Drivers: The Goblint Approach. In: ASE '16, pp. 391–402, ACM (2016), doi: 10.1145/2970276.2970337

[135] Zakharov, I.S., Mandrykin, M.U., Mutilin, V.S., Novikov, E., Petrenko, A.K., Khoroshilov, A.V.: Configurable toolset for static verification of operating systems kernel modules. Program. Comput. Softw. **41**(1), 49–64 (2015), doi: 10.1134/S0361768815010065

[136] Zhou, Q., Li, L., Wang, L., Xue, J., Feng, X.: May-happen-in-parallel analysis with static vector clocks. In: CGO '18, pp. 228–240, ACM (2018), doi: 10.1145/3168813