Mälardalen University Licentiate Thesis
No.???

Static Timing Analysis of Parallel Software Using Abstract Execution – DRAFT –

Andreas Gustavsson

month year

School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden

ISSN ????-????
ISBN ??-?????-??-?
Printed by Arkitektkopia, Västerås, Sweden
Distribution: Mälardalen University Press

Abstract

The Power Wall has stopped the past trend of increasing processor throughput by increasing the clock frequency and the instruction level parallelism. Therefore, the current trend in computer hardware design is to expose explicit parallelism to the software level. This is most often done using multiple processing cores situated on a single processor chip. The cores usually share some resources on the chip, such as some level of cache memory (which means that they also share the interconnect, e.g., a bus, to that memory and also all higher levels of memory), and to fully exploit this type of parallel processor chip, programs running on it will have to be parallel as well. Since multi-core processors are the new standard, even embedded real-time systems will (and some already do) incorporate this kind of processor and parallel code.

A real-time system is any system whose correctness is dependent both on its functional and temporal output. For some real-time systems, a failure to meet the temporal requirements can have catastrophic consequences. Therefore, it is of utmost importance that methods to analyze and derive safe estimations on the timing properties of parallel computer systems are developed.

This thesis presents an analysis that derives safe (lower and upper) bounds on the execution time of a given parallel program. The interface to the analysis is a rudimentary parallel programming language that is formally (both syntactically and semantically) defined in the thesis. The analysis is based on abstract execution, which is itself based on abstract interpretation techniques that have been commonly used within the field of timing analysis of single-core computer systems, to derive safe timing bounds in an efficient (although, over-approximative) way. Basically, abstract execution simulates the execution of several real executions of the analyzed program in one go.

The thesis proves the soundness of the presented analysis (i.e., that the estimated timing bounds are indeed safe) and also includes three case studies, each showing an important feature or characteristic of the analysis.

Acknowledgment

Thanks!

Andreas Gustavsson
Västerås, January 1, 2004 ??????????????

Contents

Chapter 1

Introduction

This chapter starts by introducing the fundamental concepts used within the field of the thesis. It also states the asked research questions, the approach used to answer the questions and the resulting contributions of the thesis. This chapter also presents the papers included in the thesis and a pilot study on using model-checking for timing analysis of parallel real-time systems.

1.1 Real-Time Systems

As computers have become smaller, faster, cheaper and more reliable, their range of use has rapidly increased. Today, everything from wrist watches to airplanes are computer-controlled. These types of systems are commonly referred to as *embedded* systems; i.e., one or more controller chips with accompanying software are embedded within the product. It has been approximated that over 99 percent of the worldwide production of computer chips are destined for embedded systems [9].

A *real-time* system is often an embedded system for which the timing behavior is of great importance. More formally, the Oxford Dictionary of Computing gives the following definition of a real-time system [40].

> "Any system in which the time at which output is produced is significant. This is usually because the input corresponds to some movement in the physical world, and the output has to relate to that same movement. The lag from input time to output time must be sufficiently small for acceptable timeliness."

The word "timeliness" refers to the total system and can be dependent on mechanical properties like inertia. One example is the compensation of temporary deviations in the supporting structure (e.g., a twisting frame) when firing a missile to keep the missile's exit path constant throughout the process. Another example is to fire the airbag in a colliding car. This should not be done too soon, or the airbag will have lost too much pressure upon the human impact, and not too late, or the airbag could cause additional damage upon impact; i.e., the inertia of the human body and the retardation of the colliding car both impact on the timeliness of the airbag system. It should thus be apparent that the correctness of a real-time system depends both on the logical result of the performed computations and the time at which the result is produced.

Real-time systems can be divided into two categories: hard and soft real-time systems. Hard real-time systems are such that failure to produce the computational result within certain timing bounds could have catastrophic consequences. One example of a hard real-time system is the above-mentioned airbag system. Soft real-time systems, on the other hand, can tolerate missing these deadlines to some extent and still function properly. One example of a soft real-time system is a video displaying device. Missing to display a video frame within the given bounds will not be catastrophic (but perhaps annoying to the viewer if it occurs too often). The video will still continue to play (although, perhaps with reduced displaying quality), assuming that the system is not overloaded in general.

The current trend in computer hardware design is to make parallelism explicitly available to the programmer, often in the form of multiple processing cores on the same chip. This strategy helps increasing the chip's throughput without hitting the power wall since the individual processing cores on the multi-core chip are usually simpler than a single core implemented on the equivalent chip area [70]. The cores typically share some resources, such as some level of on-chip cache memory, which introduces dependencies and conflicts between the cores; e.g., simultaneous accesses from two or more cores to shared resources will introduce delays for some of the cores. Processor chips of this kind of *multi-core* architecture are currently being used in real-time systems within, for example, the automotive industry.

To fully utilize the multi-core architecture, algorithms will have to be parallelized over multiple tasks (e.g., threads). This means that the tasks will have to share resources and communicate and synchronize with each other. There already exist software libraries for explicitly parallelizing sequential code automatically. One example of such a library available for C/C++ and Fortran code running on shared-memory machines is OpenMP [64]. The conclusion is that

parallel software running on parallel hardware is already available today and will probably be the standard way of computing in the future, also for real-time systems.

When proving the correctness of, and/or the schedulability of the tasks in, a real-time system, it is, as far as the author knows, always assumed that *safe* (i.e., not under-approximated) bounds on the timing behavior of all tasks in the system are known. The timing bounds are, for example, used as input to algorithms that prove or falsify the schedulability of the tasks in the system [4, 22, 54]. Therefore, it is of crucial importance that methods for deriving safe timing bounds (referred to as estimates) for this type of parallel computational systems are defined.

This thesis presents a method that derives safe estimates on the timing bounds for the described type of systems. The method mainly targets hard real-time systems. However, it can be applied to any computer system fitting the assumptions made in the upcoming chapters.

1.2 Timing Analysis of Real-Time Systems

A program's execution time (i.e., the amount of time it takes to execute the entire program from its entry point to its exit point) is not constant; it is dependent on the initial system state. This state includes the input to the program (i.e., the values of its arguments), the hardware state (e.g., cache memory contents) and the state of any other software that is executing on the same hardware. However, for any program and any set of initial states, at least one of the resulting execution times will be equal to the shortest execution time for the given program and set of initial states. The shortest execution time is referred to as the Best-Case Execution Time (*BCET*). Likewise, at least one of the resulting execution times will be equal to the longest execution time for the given program and set of initial states. The longest execution time is referred to as the Worst-Case Execution Time (*WCET*).

Traditionally, when computers were purely sequential, the main focus of timing analysis only targeted estimations of the WCET. This was possible because the analyzed systems did not suffer from any timing anomalies, which meant that the local worst-case scenario (i.e., the longest possible execution time for a single instruction, or a block of instructions) always resulted in the global worst-case. However, when introducing multi-core architectures with shared memory, this is no longer the case [1, 56, 76]. This means that the only safe option is to take all the possible execution times between, and including,

the local BCET and WCET into account when deriving the global (BCET and) WCET.

Today, there exist several algorithms and tools that strive to derive a *safe* and *tight* (i.e., not too over-approximate) estimate of the WCET of a sequential task targeted for sequential hardware. Some examples of such tools are aiT [20, 91], Bound-T [37, 91], Chronos [49, 91], Heptane [91], OTAWA [6], RapiTime [75, 91], SWEET [17, 91], SymTA/P [91] and TuBound [72, 91]. aiT, Bound-T and RapiTime are commercial tools while the others are primarily research prototypes. aiT, Bound-T, Chronos, Heptane, OTAWA and TuBound are purely static tools while SWEET and SymTA/P mainly use static WCET analysis techniques, but also dynamic techniques to some extent. RapiTime is heavily based on dynamic techniques.

In *dynamic* WCET analysis, measurements of the actual execution time of the software running on the target hardware are performed. This method is not guaranteed to execute the program's worst-case path, though, which could, for example, include some error-handling routine that is only rarely executed. Thus, the WCET might be gravely under-estimated; i.e., there might exist paths through the code with considerably worse (longer) execution times than the worst execution time detected by the measurements.

In *static* WCET analysis, the program code and the properties of the target hardware are analyzed without actually executing the program. Instead, the analysis is based on the semantics of the programming language constructs used to define the program and a (timing) model of the target hardware. Static methods usually try to find a tight estimation of the WCET, but always safely over-estimate it.

Static WCET analyses are normally split into three subtasks: the *low-level analysis*, which attempts to find safe timing estimates for executions of code sequences, the *flow analysis*, which constrains the possible paths through the code, and the *calculation*, where the most time-consuming path is found, using information derived in the first two phases. This traditional approach assumes that the analyzed program consists of a single flow of control; i.e., is sequential. In a parallel program, there are several flows of control, possibly with dependencies among them. The consequence is that the traditional three-phase approach is not directly applicable when analyzing arbitrary parallel programs executing on parallel shared-memory architectures.

This thesis presents a static method that derives safe estimations of the BCET and WCET of a parallel program by combining the three phases into one single phase; i.e., the method directly calculates the timing bound estimates while analyzing the semantic behavior of the program, based on a (safe) timing

model of the underlying architecture.

Note that solving the problem of finding the actual WCET in the general case is comparable to solving the halting-problem (i.e., determining whether the program will terminate), which is an undecidable problem (c.f., [45]). Thus, the space of possible system states that a WCET analysis must search through could be extremely large, or even infinite, in the general case. This means that the analysis itself might not terminate in the general case. Therefore, techniques to increase the probability of, or even more desirable, guarantee, analysis termination must be derived. For many of the traditional methods for analyzing sequential programs, there are ways to guarantee termination using widening/narrowing techniques [62]. These techniques are not directly applicable to the method presented in this thesis, though.

1.3 Research Questions

This thesis mainly tries to answer the following questions.

Question 1: *"What are the distinguishing features of a parallel computer system (i.e., the hardware and software combination) that must be taken into account in a timing analysis on the code level?"*

Question 2: *"How can a parallel computer system be analyzed to derive safe and tight estimations on its timing bounds?"*

Question 3: *"How can analysis termination be guaranteed?"*

1.4 Pilot Study

Model-checking is a technique for verifying properties of a model of some system. The idea of using model-checking to perform WCET analysis has been investigated and shown to be adequate for analyzing parts of a single-core system [38, 60].

Timed automata[1] can be used to model real-time systems [3]. An automaton can be viewed as a state machine with locations and edges [43]. A state represents certain values of the variables in the system and which location of an automaton is active, while the edges represent the possible transitions from one state to another [43]. (Continuous) time is expressed as a set of real-valued

[1] The formal syntax and semantics of timed automata can be found in [2] and [43].

variables called clocks. UPPAAL[2] [7, 47, 89] is a tool used to model, simulate and verify networks of timed automata [7, 8, 43].

Preceding the work presented in this thesis, an initial study [30] in which UPPAAL was used to model, and derive high precision estimates on the timing bounds of, a small parallel real-time system was performed. The paper shows that timing analysis of parallel real-time systems can be performed using the model-checking techniques available in for example UPPAAL. However, the proposed method (i.e., the way the system was modeled and analyzed) did not scale very well, for example with respect to the number of threads in the analyzed program. Therefore, it was decided not to continue on the pure model-checking path (although, there might be other ways to model the system that would succeed better).

1.5 Approach

The approach used in this thesis is to statically calculate safe BCET and WCET estimations by abstractly executing the analyzed program (c.f., [23, 28]) using a safe timing model of the underlying hardware. As previously discussed, statically computing the exact set of possible semantic states for an arbitrary program is an extremely complex task. Using abstract execution (which is based on abstract interpretation techniques), at least all the possible semantic states, given some set of initial system states, are considered, but in a less complex way.

Abstract interpretation [14, 23, 62] is a method for safely approximating the program semantics and can be used to obtain a set of possible abstract states for each point in a program. An abstract state collects, and most often over-approximates, the information given by a set of concrete semantic states. This means that an analysis based on abstractly interpreting the semantics of a program can become less complex and more efficient, but might suffer from imprecision, compared to an analysis based on the concrete semantics.

The concrete semantics of an arbitrary programming language can be abstracted in many different ways. The choice of abstraction is done by defining an abstract domain. An abstract domain is essentially the set of all possible abstract states that fit the definition of the domain.

An example of an abstract domain is **Intv**, defined as $\{[z_1, z_2] \mid -\infty \leq z_1 \leq z_2 \leq \infty \wedge z_1, z_2 \in \mathbb{Z} \cup \{-\infty, \infty\}\}$; i.e., the set of all intervals that "fit inside"

[2] An introduction to UPPAAL and the formal semantics of networks of timed automata are given in [7] and [43], respectively.

$[-\infty,\infty]$. (Note that the domain **Intv** is completely defined in Section 3.6.) This domain can be used to over-approximate the concrete domain $\{z \in \mathbb{Z} \cup \{-\infty,\infty\} \mid -\infty \leq z \leq \infty\} = \mathbb{Z} \cup \{-\infty,\infty\}$; i.e., the set of all integers between (and including) $-\infty$ and ∞.

Assume that the program variable x can have the value v, such that $v \in \{1,2,5,8\}$, in a given point of the program according to the concrete semantics (i.e., x has four possible values in the given program point). In the abstract domain, the value of x could safely be represented by $[1,8]$. This is an over-approximation since turning the abstract value into a set of concrete values yields $[1,8] \rightarrow \{1,2,3,4,5,6,7,8\} \supseteq \{1,2,5,8\}$. It can be noted that $[1,8]$ is the best (tightest) approximation of the values of x, since $[1,8]$ is the smallest interval containing all the possible concrete values of x.

Abstract execution is simply a way to abstractly simulate the execution of the analyzed program. This is done by collecting several concrete states into one (or, for some situations, several) abstract states using abstract interpretation, as discussed above. The existence of a timing model (of the underlying architecture) that provides safe information on the timing properties of individual operations within the analyzed program when executed in a particular system state allows the BCET and WCET of the analyzed program to be safely estimated.

Basically, the only assumption made on the underlying architecture is that it provides (or can simulate) a shared memory address space, that can be used for communication, and shared resources, that can be used for synchronization. One example of such an architecture is a multi-core CPU. Another example is a virtualization environment that runs on top of a distributed system and provides a shared memory view. Yet another example is any real-time operating system; e.g., VxWorks [92].

1.6 Contribution

The main contributions of this thesis are the following.

1. PPL: a formally defined, rudimentary, parallel programming language for real-time systems. The semantics of PPL includes timing behavior and is defined based on the familiar notation of operational semantics (c.f., [63]).

2. An abstraction of the PPL semantics where concrete points in time are abstracted using intervals.

3. A *safe* timing analysis based on the abstract semantics of PPL. A complete correctness/soundness proof is provided.

1.7 Included Publications

This thesis includes the material presented in the following papers. Andreas Gustavsson is the main author of all the listed publications and has alone contributed with all the technical material presented in them.

Paper A

Worst-Case Execution Time Analysis of Parallel Systems
Andreas Gustavsson.
Presented at the RTiS workshop, 2011.

This Paper addresses contribution 1 and presents the first definition of the parallel programming language and a very simple (non-generalized) hardware timing model.

Paper B

Toward Static Timing Analysis of Parallel Software
Andreas Gustavsson, Jan Gustafsson and Björn Lisper.
Presented at the WCET workshop, 2012.

This Paper addresses contributions 2 and 3 and presents a work-in-progress timing analysis that can analyze all aspects of a parallel program, except synchronization. The presented analysis uses abstract execution to derive safe estimations of the BCET and WCET of the analyzed program.

Paper C

Toward Static Timing Analysis of Parallel Software - Technical Report
Andreas Gustavsson, Jan Gustafsson and Björn Lisper.
Technical report, 2012.

This Paper addresses contributions 2 and 3 and is an extended version of Paper B. The Paper includes all the mathematical details and a sketch for the correctness/soundness proof.

Paper D

Timing Analysis of Parallel Software Using Abstract Execution
Andreas Gustavsson, Jan Gustafsson and Björn Lisper.
Submitted to the VMCAI conference, 2014.

This paper addresses contributions 1, 2 and 3 and summarizes the work presented in this thesis. It presents a timing analysis that is based on the analysis defined in Papers B and C. The presented analysis derives safe estimations of the BCET and WCET for any program defined using a slightly modified version of the language presented in Paper A (i.e., PPL), given some (safe) timing model of the underlying architecture.

1.8 Thesis Outline

The rest of this thesis is organized as follows.

Chapter 2 presents some research that is closely related to the material presented in this thesis.

Chapter 3 introduces the reader to the fundamental concepts and theories needed to understand the contents of the following chapters.

Chapter 4 formally defines PPL, a parallel programming language.

Chapter 5 presents a semi-safe abstraction of the PPL semantics. Note that the abstraction is not safe for arbitrary PPL programs and that special care must be taken if using it (c.f., Chapter 6).

Chapter 6 defines a *safe* timing analysis using abstract execution based on the abstraction made in Chapter 5.

Chapter 7 presents some examples that show how the analysis presented in Chapter 6 handles communication and synchronization in PPL programs.

Chapter 8 discusses the research questions and the analysis presented in Chapter 6. The chapter also gives pointers to future work.

For the reader's convenience, the following appendices are provided.

Appendix A summarizes the notations and nomenclature used in this thesis.

Appendices B-H present listings of the assumptions, definitions, figures, tables, algorithms, lemmas and theorems defined in this thesis, respectively.

Chapter 2

Related Work

WCET-related research started with the introduction of timing schemas by Shaw in 1989 [82]. Shaw presents rules to collapse the CFG (Control Flow Graph) of a program until a final single value represents the WCET. Excellent overviews of the WCET research from the years 2000 and 2008 can be found in [73] and [91] respectively. The field of WCET analysis for parallel systems is quite new, so there is no solid foundation of previous research to stand on.

2.1 Static Timing Analysis

The field of static WCET analysis has, just until recently, mainly been focusing on single-processor systems. In the field of low-level analysis, most research efforts have been dedicated to analyzing the effects of different hardware features, including pipelines [16, 35, 52, 83, 88], caches [50, 52, 88, 90], branch predictors [13], and super-scalar CPUs [51, 80].

Within flow analysis, most research has been dedicated to *loop bound* analysis. Flow analysis can also identify *infeasible paths*, i.e., paths which are executable according to the program control-flow graph structure, but not feasible when considering the semantics of the program and possible input data values. There are a number of approaches to flow analysis, using e.g., abstract interpretation, symbolic execution, Presburger arithmetics, specialized data flow analyses, and syntactical analysis of parse trees [28, 36, 37, 55, 88].

There is also some research on data flow analysis for parallel programs [15, 21, 46], which is of relevance to WCET analysis. Constant propagation

has also been considered [48]. A survey of analyses for concurrent and parallel programs is found in [77].

Three main methods exist for the WCET calculation: The tree-based method [12, 13, 52], originating from Park's *timing schemas* [68]; the *path-based* method [35, 84]; and the *Implicit Path Enumeration Technique* (IPET) [17, 37, 50, 74, 88], where the WCET calculation problem is formulated as an Integer Linear Programming (ILP) problem, and the set of execution paths is restricted by linear constraints.

An alternative way of computing the ILP problem is by using a graph-based approach [74]. A comparison of the graph-based and IPET approaches is performed in [38]. The graph-based approach is conducted using model-checking in UPPAAL [7, 47, 89]. It is shown that IPET outperforms the model-checking-based approach, but that model-checking allows for calculating tight WCET bounds and easy integration of complex hardware models. A combined approach is proposed, where model-checking is used to analyze local regions of the code, while IPET is used to solve the global analysis. Another motivation to why model-checking could be useful in WCET analysis can be found in [60].

For analyses based on abstract execution, it is possible to calculate the BCET and WCET estimates of sequential programs during the abstract execution, without first generating flow facts [18, 28]. This thesis uses basically the same approach, but applies it to explicitly parallel programs.

Some research has been conducted within the field of static WCET analysis for multi-core and other types of multi-processor systems. A static method for analyzing multi-core processors with a shared L2 instruction cache has been presented [94]. A limitation of this analysis is that the L1 data cache is assumed to be perfect (i.e., all accesses are assumed to be hits, which is generally not the case) and thus does not affect the contents of the L2 cache. Based on this work, the same authors also address the same problem for the case that the shared L2 cache is direct-mapped [95].

There is also an approach for analyzing multi-cores with a shared L2 instruction cache (that still assumes a perfect L1 data cache) that takes effects from timing anomaly influenced pipelines into account [11].

Staschulat et al. [85] consider an integrated task- and system-level analysis to estimate memory access times for sequential tasks running in parallel with tasks executing on other processors. Their approach requires full information about all tasks running in the system, and it makes quite strong assumptions about the task model.

Mittermayr and Blieberger [61] use a graph based approach and Kronecker algebra to calculate an estimation of the WCET of a concurrent program. The

graph is referred to as CPG (Concurrent Program Graph) and plays a role similar to the CFG for sequential programs.

Potop-Butucaru and Puaut [71] target static timing analysis of parallel processors where "channels" are used to communicate between, and synchronize, the parallel tasks. Additional edges representing such communication and synchronization are then used to connect the CFGs of the individual tasks. The goal of this approach is to enable the use of the traditional three-phase analysis when analyzing parallel systems.

Ozaktas et al. [65] focus on analyzing synchronization delays experienced by tasks executing on time-predictable shared-memory multi-core architectures.

Lv et al. [57] and Wu and Zhang [93] use model-checking of timed automata to perform WCET analysis. In this approach, a timed automata-model of the system to be analyzed is created. Then, specific properties of the model are verified to find a WCET estimate for the analyzed system. The achievable tightness of the WCET estimate depends on the level of details in the timed automata-model.

Both papers mainly propose methods for reducing the size of the state space by altering the program model without affecting the true WCET of the model. This is a very important aspect when using model-checking overall. If the model is too large and complex, the state space will "explode", which means that the number of possible states is very large and analyzing the model becomes infeasible.

Lv et al. [58] have also combined abstract interpretation with model-checking to avoid the scalability problems found in, e.g., [30]. This work does not focus on explicitly parallel software, though.

2.2 Multi-Core Analyzability

Some other research addresses the problem of (low) predictability in multi-core processors. This work mostly gives multi-core design guidelines and suggestions on how to use additional or modified hardware to increase the predictability, and thus, the analyzability. In an extension to the method found in [94], memory bits for each instruction is used to determine whether the instruction should be cached or not [34]. E.g., to avoid pollution of the shared cache, "Static Single Usage" instructions (i.e., instructions in the program that are only referenced/executed once) should not be cached. This generates the possibility to determine a tighter WCET estimate.

Arbiters (hardware circuits) can be added to a shared memory multi-core processor to synchronize the memory accesses from different cores in order to increase the timing predictability of the system [66]. The result is a multi-core architecture that can be analyzed with existing single-core (and single-task) WCET analysis tools.

GAMC [67] is an SDRAM controller which upper bounds the delay a core can suffer from memory-access interferences from other cores. This is an important approach since the largest memory access latency will occur when accessing the main memory. The result is tight WCET approximations which only differ a few percent from the largest measured execution times, for a specific analyzed program suite.

Time Division Multiple Access (TDMA)-based memory bus access policies can also be introduced to make all memory accesses predictable, regarding the WCET [4, 79]. The problem with this approach is that the performance of the processor will be seriously degraded since, in the average case, a memory access from any core will be stalled for half the TDMA period (and the whole period in the worst case).

Kelter et al. [44] suggest to use the Priority Division (PD) protocol instead of the TDMA protocol. They show that PD is a very promising replacement for TDMA that provides predictability while not degrading the performance as severely as TDMA.

The MERASA project [59, 78] strives towards providing a timing analyzable multi-core CPU with a system level software (c.f., operating system). A case study [78] has been performed, in which an estimation of the WCET of a parallel 3D multi-grid solver, executing on the MERASA multi-core platform, is derived. The parMERASA project [69] is a continuation of the MERASA project.

Chapter 3

Preliminaries

In general, basing a timing analysis on the concrete semantics of a program is infeasible due to the enormous number of states that must be explored. As discussed in Section 1.5, abstract interpretation [14, 23, 62] is a method for *safely* approximating the concrete program semantics and can be used to obtain a set of possible abstract states for each point in a program. An abstract state collects, and most often over-approximates, the information given by a *set* of concrete semantic states. This means that an analysis based on abstractly interpreting the semantics of a program can become less complex and more efficient compared to an analysis based on the concrete semantics. The analysis presented in this thesis is based on abstract interpretation. Therefore, this chapter introduces the foundations used by abstract interpretation techniques.

NOTE. *A summary of the notation and nomenclature used in this thesis can be found in Appendix A.*

3.1 Partially Ordered Sets & Complete Lattices[1]

The *relation*, as described by $\mathscr{R}$: $A \times B \rightarrow \{\texttt{true}, \texttt{false}\}$ where $A \times B$ is the Cartesian product of the two sets A and B, between two elements $a \in A$ and

[1] Extensive introductions to complete lattices can be found in many textbooks, e.g., [62].

$b \in B$ is denoted by $a \mathrel{\mathscr{R}} b$. Given that for every $a \in A$, there is at most one element, $b \in B$, such that $a \mathrel{\mathscr{R}} b$, then $\mathscr{R}$ is said to be a *partial function* from A to B. Given that for every $a \in A$, there is exactly one element, $b \in B$, such that $a \mathrel{\mathscr{R}} b$, then $\mathscr{R}$ is said to be a *total function* from A to B.

A *partial ordering* is a relation $\sqsubseteq: A \times A \to \{\texttt{true}, \texttt{false}\}$ that is reflexive (i.e., $\forall a \in A : a \sqsubseteq a$), transitive (i.e., $\forall a, a', a'' \in A : ((a \sqsubseteq a' \wedge a' \sqsubseteq a'') \Rightarrow a \sqsubseteq a'')$) and anti-symmetric (i.e., $\forall a, a' \in A : ((a \sqsubseteq a' \wedge a' \sqsubseteq a) \Rightarrow a = a')$). The pair $(A, \mathscr{R})$ is a *partially ordered set* if $\mathscr{R}: A \times A \to \{\texttt{true}, \texttt{false}\}$ is a partial ordering on A.

A subset A' of A has $a \in A$ as an *upper bound* if $\forall a' \in A' : a' \sqsubseteq a$ and as a *lower bound* if $\forall a' \in A' : a \sqsubseteq a'$. The element $a \in A$ is the *least upper bound* of A' if a is an upper bound of A' and for all other upper bounds, $a' \in A$, of A', $a \sqsubseteq a'$ (c.f., Definition 3.28). The element $a \in A$ is the *greatest lower bound* of A' if a is a lower bound of A' and for all other lower bounds, $a' \in A$, of A', $a' \sqsubseteq a$ (c.f., Definition 3.27). Note that a greatest lower bound and/or a least upper bound might not exist for all subsets of a partially ordered set. When they do exist, they are unique (since $\sqsubseteq$ is anti-symmetric) and will be denoted $\bigsqcap A'$ and $\bigsqcup A'$, respectively. The shorthand $a \sqcap a'$ will be used to denote $\bigsqcap\{a, a'\}$. Likewise, $a \sqcup a'$ will be used to denote $\bigsqcup\{a, a'\}$.

A *complete lattice*, $V = \langle V, \sqsubseteq, \bigsqcup, \bigsqcap, \bot, \top \rangle$, is a partially ordered set, $(V, \sqsubseteq)$, such that all subsets have greatest lower bounds and least upper bounds. The least element of V is denoted $\bot$ (the *bottom element*) and is defined as $\bot = \bigsqcup \emptyset = \bigsqcap V$. The greatest element of V is denoted $\top$ (the *top element*) and is defined as $\top = \bigsqcup V = \bigsqcap \emptyset$.

The properties of *monotone*, *completely additive* and *completely multiplicative* functions are given in Definitions 3.1, 3.2 and 3.3, respectively. Note that when V_1 and V_2 are complete lattices, all subsets of these sets have least upper bounds and greatest lower bounds. Lemma 3.4 states some specific properties of a completely multiplicative function.

Definition 3.1 (Monotone function):
A function, $f : V_1 \to V_2$, between the partially ordered sets $V_1 = (V_1, \sqsubseteq_1)$ and $V_2 = (V_2, \sqsubseteq_2)$ is monotone if:

$$\forall v_1, v_1' \in V_1 : v_1 \sqsubseteq_1 v_1' \Rightarrow f(v_1) \sqsubseteq_2 f(v_1') \qquad \square$$

Definition 3.2 (Completely additive function):
A function, $f : V_1 \to V_2$, between the partially ordered sets $V_1 = (V_1, \sqsubseteq_1)$ and $V_2 = (V_2, \sqsubseteq_2)$ is completely additive if for all $V_1' \subseteq V_1$

$$f(\bigsqcup_1 V_1') = \bigsqcup_2 \{f(v) \mid v \in V_1'\}$$

whenever $\bigsqcup_1 V_1'$ *and* $\bigsqcup_2\{f(v) \mid v \in V_1'\}$ *exist.* □

Definition 3.3 (Completely multiplicative function):
A function, $f : V_1 \to V_2$*, between the partially ordered sets* $V_1 = (V_1, \sqsubseteq_1)$ *and* $V_2 = (V_2, \sqsubseteq_2)$ *is completely multiplicative if for all* $V_1' \subseteq V_1$

$$f(\textstyle\bigsqcap_1 V_1') = \bigsqcap_2\{f(v) \mid v \in V_1'\}$$

whenever $\bigsqcap_1 V_1'$ *and* $\bigsqcap_2\{f(v) \mid v \in V_1'\}$ *exist.* □

Lemma 3.4 (Completely multiplicative functions):
If $V = \langle V, \sqsubseteq, \bigsqcup, \bigsqcap, \bot, \top\rangle$ *and* $\tilde{V} = \langle \tilde{V}, \tilde{\sqsubseteq}, \tilde{\bigsqcup}, \tilde{\bigsqcap}, \tilde{\bot}, \tilde{\top}\rangle$ *are complete lattices and* $\tilde{V}$ *is finite, then the three conditions*

1. $\gamma : \tilde{V} \to V$ *is monotone,*
2. $\gamma(\tilde{\top}) = \top$*, and*
3. $\gamma(\tilde{v} \tilde{\sqcap} \tilde{v}') = \gamma(\tilde{v}) \sqcap \gamma(\tilde{v}')$*, whenever* $\tilde{v} \not\tilde{\sqsubseteq} \tilde{v}' \wedge \tilde{v}' \not\tilde{\sqsubseteq} \tilde{v}$*, where* $\tilde{v}, \tilde{v}' \in \tilde{V}$

are jointly equivalent to $\gamma : \tilde{V} \to V$ *being completely multiplicative.* □

PROOF (c.f., [62]). Assume that $V = \langle V, \sqsubseteq, \bigsqcup, \bigsqcap, \bot, \top\rangle$ and $\tilde{V} = \langle \tilde{V}, \tilde{\sqsubseteq}, \tilde{\bigsqcup}, \tilde{\bigsqcap}, \tilde{\bot}, \tilde{\top}\rangle$ are complete lattices and that $\tilde{V}$ is finite.

First note that if $\gamma : \tilde{V} \to V$ is completely multiplicative, then the three conditions trivially hold. Next, assuming that the three conditions are fulfilled, it will be proven that

$$\gamma(\tilde{\bigsqcap}\tilde{V}') = \bigsqcap\{\gamma(\tilde{v}) \mid \tilde{v} \in \tilde{V}'\}$$

where $\tilde{V}' \subseteq \tilde{V}$, using induction on the finite cardinality of $\tilde{V}' \subseteq \tilde{V}$.

If the cardinality of $\tilde{V}'$ is 0, then $\gamma(\tilde{\bigsqcap}\tilde{V}') = \bigsqcap\{\gamma(\tilde{v}) \mid \tilde{v} \in \tilde{V}'\}$ follows from condition 2. This proves the base case of the induction.

If the cardinality of $\tilde{V}'$ is larger than 0, then $\tilde{V}' = \tilde{V}'' \cup \{\tilde{v}''\}$ where $\tilde{v}'' \notin \tilde{V}''$; which ensures that the cardinality of $\tilde{V}''$ is strictly less than that of $\tilde{V}'$. Note that by condition 1, $\gamma(\tilde{v} \tilde{\sqcap} \tilde{v}') = \gamma(\tilde{v}) \sqcap \gamma(\tilde{v}')$ also when $\tilde{v} \tilde{\sqsubseteq} \tilde{v}' \vee \tilde{v}' \tilde{\sqsubseteq} \tilde{v}$. Hence, by assuming that $\gamma(\tilde{\bigsqcap}\tilde{V}'') = \bigsqcap\{\gamma(\tilde{v}) \mid \tilde{v} \in \tilde{V}''\}$ (this is the induction assumption),

$$\begin{aligned}
\gamma(\tilde{\bigsqcap}\tilde{V}') &\overset{calc.}{=} \gamma((\tilde{\bigsqcap}\tilde{V}'') \tilde{\sqcap} \tilde{v}'') \\
&\overset{cond.\ 1\ and\ 3}{=} \gamma(\tilde{\bigsqcap}\tilde{V}'') \sqcap \gamma(\tilde{v}'') \\
&\overset{ind.\ ass.}{=} (\bigsqcap\{\gamma(\tilde{v}) \mid \tilde{v} \in \tilde{V}''\}) \sqcap \gamma(\tilde{v}'') \\
&\overset{calc.}{=} \bigsqcap(\{\gamma(\tilde{v}) \mid \tilde{v} \in \tilde{V}''\} \cup \{\gamma(\tilde{v}'')\}) \\
&\overset{calc.}{=} \bigsqcap\{\gamma(\tilde{v}) \mid \tilde{v} \in \tilde{V}'\}
\end{aligned}$$

which proves the lemma. ■

3.2 Constructing Complete Lattices

There are several different ways to construct complete lattices. Any given set can be *lifted* into a complete lattice (Theorem 3.5).

Theorem 3.5 (Complete lattice – Lifting):
If S is a set, then $\langle \mathscr{P}(S), \subseteq, \bigcup, \bigcap, \emptyset, S\rangle$ is a complete lattice. □

PROOF. Assume that S is a set and let $S^{\mathscr{P}} \subseteq \mathscr{P}(S)$. It is then trivially the case that $\bigsqcup S^{\mathscr{P}} = \bigcup S^{\mathscr{P}}$, $\bigsqcap S^{\mathscr{P}} = \bigcap S^{\mathscr{P}}$, $\bot = \emptyset$ and $\top = S$ if $\sqsubseteq = \subseteq$ (note that $\subseteq$ is reflexive, transitive and anti-symmetric by definition). ■

The *Cartesian product* of two complete lattices is a complete lattice (Theorem 3.6).

Theorem 3.6 (Complete lattice – Cartesian product):
If $\langle V_1, \sqsubseteq_1, \bigsqcup_1, \bigsqcap_1, \bot_1, \top_1\rangle$ and $\langle V_2, \sqsubseteq_2, \bigsqcup_2, \bigsqcap_2, \bot_2, \top_2\rangle$ are complete lattices, then so is $\langle V, \sqsubseteq, \bigsqcup, \bigsqcap, \bot, \top\rangle$ where (let $V' \subseteq V$):

$$
\begin{aligned}
&V = V_1 \times V_2 = \{(v_1, v_2) \mid v_1 \in V_1 \wedge v_2 \in V_2\}\\
&(v_1, v_2) \sqsubseteq (v_1', v_2') \Longleftrightarrow v_1 \sqsubseteq_1 v_1' \wedge v_2 \sqsubseteq_2 v_2' \text{ where } v_1, v_1' \in V_1 \text{ and } v_2, v_2' \in V_2\\
&\bigsqcup V' = (\bigsqcup_1 \{v_1 \in V_1 \mid \exists v_2 \in V_2 : (v_1, v_2) \in V'\},\\
&\qquad \bigsqcup_2 \{v_2 \in V_2 \mid \exists v_1 \in V_1 : (v_1, v_2) \in V'\})\\
&\bigsqcap V' = (\bigsqcap_1 \{v_1 \in V_1 \mid \exists v_2 \in V_2 : (v_1, v_2) \in V'\},\\
&\qquad \bigsqcap_2 \{v_2 \in V_2 \mid \exists v_1 \in V_1 : (v_1, v_2) \in V'\})\\
&\bot = (\bot_1, \bot_2)\\
&\top = (\top_1, \top_2)
\end{aligned}
$$

□

PROOF. Assume that $\langle V_1, \sqsubseteq_1, \bigsqcup_1, \bigsqcap_1, \bot_1, \top_1\rangle$ and $\langle V_2, \sqsubseteq_2, \bigsqcup_2, \bigsqcap_2, \bot_2, \top_2\rangle$ are complete lattices and let $V = \{(v_1, v_2) \mid v_1 \in V_1 \wedge v_2 \in V_2\}$ and $(v_1, v_2) \sqsubseteq (v_1', v_2') \Longleftrightarrow v_1 \sqsubseteq_1 v_1' \wedge v_2 \sqsubseteq_2 v_2'$ where $v_1, v_1' \in V_1$ and $v_2, v_2' \in V_2$. (Note that it is straightforward to verify that $(V, \sqsubseteq)$ is a partially ordered set since $\sqsubseteq_1$ and $\sqsubseteq_2$ are partial orders.) Also assume that $V' \subseteq V$.

Since $\bigsqcup_1 \{v_1 \in V_1 \mid \exists v_2 \in V_2 : (v_1, v_2) \in V'\} \sqsubseteq_1 v_1^u$ for all upper bounds, v_1^u, of $\{v_1 \in V_1 \mid \exists v_2 \in V_2 : (v_1, v_2) \in V'\}$ and $\bigsqcup_2 \{v_2 \in V_2 \mid \exists v_1 \in V_1 : (v_1, v_2) \in V'\} \sqsubseteq_2 v_2^u$ for all upper bounds, v_2^u, of $\{v_2 \in V_2 \mid \exists v_1 \in V_1 : (v_1, v_2) \in V'\}$, it is easy to see that $\bigsqcup V' = (\bigsqcup_1 \{v_1 \in V_1 \mid \exists v_2 \in V_2 : (v_1, v_2) \in V'\}, \bigsqcup_2 \{v_2 \in V_2 \mid \exists v_1 \in V_1 : (v_1, v_2) \in V'\}) \sqsubseteq (v_1^u, v_2^u)$ (c.f., the definition of $\sqsubseteq$ above). $\bigsqcap V'$ is shown in a similar manner.

Since $\bot_1 = \bigsqcup_1 \emptyset$ and $\bot_2 = \bigsqcup_2 \emptyset$, it is easy to see that $\bot = (\bigsqcup_1 \emptyset, \bigsqcup_2 \emptyset) = (\bot_1, \bot_2)$. $\top$ is shown in a similar manner. ■

A space of total functions where the domain of the functions is a set and the range is a complete lattice is itself a complete lattice (Theorem 3.7).

Theorem 3.7 (Complete lattice – Total function space):
If S is a set and $\langle V_1, \sqsubseteq_1, \bigsqcup_1, \bigsqcap_1, \bot_1, \top_1 \rangle$ is a complete lattice, then $\langle V, \sqsubseteq, \bigsqcup, \bigsqcap, \bot, \top \rangle$ where (let $V' \subseteq V$)

$$
\begin{aligned}
&V = S \to V_1 = \{f : S \to V_1 \mid f \text{ is a total function}\} \\
&f \sqsubseteq f' \iff \forall s \in S : f(s) \sqsubseteq_1 f'(s) \text{ where } f, f' \in V \\
&\bigsqcup V' = \lambda s \in S. \bigsqcup_1 \{f(s) \mid f \in V'\}, \\
&\bigsqcap V' = \lambda s \in S. \bigsqcap_1 \{f(s) \mid f \in V'\}, \\
&\bot = \lambda s \in S. \bot_1 \\
&\top = \lambda s \in S. \top_1
\end{aligned}
$$

is also a complete lattice. □

PROOF. Assume that S is a set and $\langle V_1, \sqsubseteq_1, \bigsqcup_1, \bigsqcap_1, \bot_1, \top_1 \rangle$ is a complete lattice, $V = S \to V_1 = \{f : S \to V_1 \mid f \text{ is a total function}\}$ and $f \sqsubseteq f' \iff \forall s \in S : f(s) \sqsubseteq_1 f'(s)$ where $f, f' \in V$. (Note that it is straightforward to verify that $(V, \sqsubseteq)$ is a partially ordered set.) Also assume that $V' \subseteq V$. Note that the totality of $f \in V$ will be implicitly used.

It is easy to see that $\forall s \in S : \forall f' \in V' : f'(s) \sqsubseteq_1 \bigsqcup_1 \{f(s) \mid f \in V'\}$ and that $\forall s \in S : \bigsqcup_1 \{f(s) \mid f \in V'\} \sqsubseteq_1 f''(s)$ for any $f'' \in V$ such that $\forall s \in S : \forall f' \in V' : f'(s) \sqsubseteq_1 f''(s)$ since $\langle V_1, \sqsubseteq_1, \bigsqcup_1, \bigsqcap_1, \bot_1, \top_1 \rangle$ is a complete lattice. But, then it must be that $\bigsqcup V' = \lambda s \in S. \bigsqcup_1 \{f(s) \mid f \in V'\}$ (c.f., the definition of $\sqsubseteq$ above). $\bigsqcap V'$ is shown in a similar manner.

Since $\bot_1 = \bigsqcup_1 \emptyset$, it is easy to see that $\bot = \lambda s \in S. \bigsqcup_1 \emptyset = \lambda s \in S. \bot_1$. $\top$ is shown in a similar manner. ■

A space of monotone functions where both the domain and the range of the functions are complete lattices is itself a complete lattice (Theorem 3.8).

Theorem 3.8 (Complete lattice – Monotone function space):
If $\langle V_1, \sqsubseteq_1, \bigsqcup_1, \bigsqcap_1, \bot_1, \top_1 \rangle$ and $\langle V_2, \sqsubseteq_2, \bigsqcup_2, \bigsqcap_2, \bot_2, \top_2 \rangle$ are complete lattices,

then so is $\langle V, \sqsubseteq, \bigsqcup, \bigsqcap, \bot, \top \rangle$ *where (let* $V' \subseteq V$*):*

$$
\begin{aligned}
&V = V_1 \to V_2 = \{f : V_1 \to V_2 \mid f \text{ is a monotone function}\}\\
&f \sqsubseteq f' \iff \forall v_1 \in V_1 : f(v_1) \sqsubseteq_2 f'(v_1) \text{ where } f, f' \in V\\
&\bigsqcup V' = \lambda v_1 \in V_1 . \bigsqcup_2 \{f(v_1) \mid f \in V'\},\\
&\bigsqcap V' = \lambda v_1 \in V_1 . \bigsqcap_2 \{f(v_1) \mid f \in V'\},\\
&\bot = \lambda v_1 \in V_1 . \bot_2\\
&\top = \lambda v_1 \in V_1 . \top_2
\end{aligned}
$$

□

PROOF. Similar to the proof of Theorem 3.7 with the addition that the monotonicity of $f \in V$ gives that $\forall v_1, v_1' \in V_1 : v_1 \sqsubseteq_1 v_1' \Rightarrow f(v_1) \sqsubseteq_2 f(v_1')$ (c.f., Definition 3.1). ■

3.3 Galois Connections & Galois Insertions

The concrete semantics of a programming language can be abstracted in many different ways. The choice of abstraction is done by defining an *abstract domain*. A domain is, in general, a complete lattice, and an abstract domain is essentially the set of all possible abstract states that fit the definition of the domain. It is often shown that the abstract domain is a safe over-approximation of the *concrete domain* by deriving a *Galois connection* between the two domains [62]. A Galois connection between two domains (i.e., complete lattices), V and D, is described by an *abstraction function*, α, and a *concretization function*, γ, which must fulfill the criterion in Definition 3.9.

Definition 3.9 (Galois connection):
$\langle \alpha : V \to D, \gamma : D \to V \rangle$ *is a Galois connection iff* α *and* γ *are monotone functions that fulfill*

$$
\begin{cases}
\alpha \circ \gamma \sqsubseteq_D \lambda d.d\\
\gamma \circ \alpha \sqsupseteq_V \lambda v.v
\end{cases}
$$

for all $v \in V$ *and* $d \in D$*, where* V *is the concrete domain and* D *is the abstract domain.* □

An often useful special case of a Galois connection is called a *Galois insertion*; c.f., Definition 3.10.

Definition 3.10 (Galois insertion):
$\langle \alpha : V \to D, \gamma : D \to V \rangle$ is a Galois insertion iff α and γ are monotone functions that fulfill

$$\begin{cases} \alpha \circ \gamma = & \lambda d.d \\ \gamma \circ \alpha \sqsupseteq_V & \lambda v.v \end{cases}$$

for all $v \in V$ and $d \in D$, where V is the concrete domain and D is the abstract domain. □

A function in the concrete domain, $f : V \to V$, can be safely approximated by a function in the abstract domain, $\tilde{f} : D \to D$, iff $\forall d \in D : f(\gamma(d)) \sqsubseteq \gamma(\tilde{f}(d))$. The best approximation is achieved by inducing f along α [62]; c.f., Definition 3.11.

Definition 3.11 (Induced function):
Assuming that $\langle \alpha : V \to D, \gamma : D \to V \rangle$ is a Galois connection, the best approximation, $\tilde{f}$, of $f : V \to V$ in $D \to D$ is given by:

$$\tilde{f} = \alpha \circ f \circ \gamma$$ □

Sometimes, it is more convenient to work with adjunctions (c.f., Definition 3.12) instead of Galois connections.

Definition 3.12 (Adjunction):
$\langle \alpha : V \to D, \gamma : D \to V \rangle$ is said to be an adjunction between the complete lattices $V = \langle V, \sqsubseteq_V, \bigsqcup_V, \bigsqcap_V, \bot_V, \top_V \rangle$ and $D = \langle D, \sqsubseteq_D, \bigsqcup_D, \bigsqcap_D, \bot_D, \top_D \rangle$ iff α and γ are total functions that satisfy

$$\alpha(v) \sqsubseteq_D d \iff v \sqsubseteq_V \gamma(d)$$

for all $v \in V$ and $d \in D$. □

In fact, adjunctions are Galois connections (Theorem 3.13).

Theorem 3.13 (Adjunctions and Galois connections):
$\langle \alpha : V \to D, \gamma : D \to V \rangle$ is an adjunction iff it is a Galois connection. □

PROOF (c.f., [62]). First assume that $\langle \alpha : V \to D, \gamma : D \to V \rangle$ is an adjunction. It will be proven that it also is a Galois connection by showing that $\gamma \circ \alpha \sqsupseteq_V \lambda v.v$ and $\alpha \circ \gamma \sqsubseteq_D \lambda d.d$. For any $v \in V$, trivially $\alpha(v) \sqsubseteq_D \alpha(v)$. Using that $\alpha(v) \sqsubseteq_D d \Rightarrow v \sqsubseteq_V \gamma(d)$, it can be established that $v \sqsubseteq_V \gamma(\alpha(v))$. Similarly, for any $d \in D$, trivially $\gamma(d) \sqsubseteq_V \gamma(d)$. Using that $v \sqsubseteq_V \gamma(d) \Rightarrow \alpha(v) \sqsubseteq_D d$, it can

be established that $\alpha(\gamma(d)) \sqsubseteq_D d$. Thus, $\langle \alpha : V \to D, \gamma : D \to V \rangle$ is a Galois connection.

Next assume that $\langle \alpha : V \to D, \gamma : D \to V \rangle$ is a Galois connection. It will be proven that it also is an adjunction by showing that $\alpha(v) \sqsubseteq_D d \Rightarrow v \sqsubseteq_V \gamma(d)$ and $v \sqsubseteq_V \gamma(d) \Rightarrow \alpha(v) \sqsubseteq_D d$. So, first assume that $\alpha(v) \sqsubseteq_D d$. Then, since γ is monotone, $\gamma(\alpha(v)) \sqsubseteq_V \gamma(d)$. Using that $\gamma \circ \alpha \sqsupseteq_V \lambda v.v$, it can be established that $v \sqsubseteq_V \gamma(\alpha(v)) \sqsubseteq_V \gamma(d)$ as required. For the second part of the proof, assume that $v \sqsubseteq_V \gamma(d)$. Then, since α is monotone, $\alpha(v) \sqsubseteq_D \alpha(\gamma(d))$. Using that $\alpha \circ \gamma \sqsubseteq_D \lambda d.d$, it can be established that $\alpha(v) \sqsubseteq_D \alpha(\gamma(d)) \sqsubseteq_D d$ as required.■

The abstraction and concretization functions are strictly related as described by Lemma 3.14.

Lemma 3.14 (Relation between α and γ):
If $V = \langle V, \sqsubseteq, \bigsqcup, \bigsqcap, \bot, \top \rangle$ and $\tilde{V} = \langle \tilde{V}, \tilde{\sqsubseteq}, \tilde{\bigsqcup}, \tilde{\bigsqcap}, \tilde{\bot}, \tilde{\top} \rangle$ are complete lattices, and $\langle \alpha : V \to \tilde{V}, \gamma : \tilde{V} \to V \rangle$ is a Galois connection between these lattices, then ($v \in V$ and $\tilde{v} \in \tilde{V}$):

1. *α uniquely determines γ by $\gamma(\tilde{v}) = \bigsqcup\{v \mid \alpha(v) \tilde{\sqsubseteq} \tilde{v}\}$ and γ uniquely determines α by $\alpha(v) = \tilde{\bigsqcap}\{\tilde{v} \mid v \sqsubseteq \gamma(\tilde{v})\}$.*

2. *α is completely additive and γ is completely multiplicative.*

In particular, $\alpha(\bot) = \tilde{\bot}$ and $\gamma(\tilde{\top}) = \top$. □

PROOF (c.f., [62]). Assume that $V = \langle V, \sqsubseteq, \bigsqcup, \bigsqcap, \bot, \top \rangle$ and $\tilde{V} = \langle \tilde{V}, \tilde{\sqsubseteq}, \tilde{\bigsqcup}, \tilde{\bigsqcap}, \tilde{\bot}, \tilde{\top} \rangle$ are complete lattices, $\langle \alpha : V \to \tilde{V}, \gamma : \tilde{V} \to V \rangle$ is a Galois connection between these lattices, $v \in V$ and $\tilde{v} \in \tilde{V}$.

To show 1, it will first be shown that γ is determined by α. Since $\langle \alpha : V \to \tilde{V}, \gamma : \tilde{V} \to V \rangle$ is an adjunction (Theorem 3.13), it must be that $\gamma(\tilde{v}) = \bigsqcup\{v \mid v \sqsubseteq \gamma(\tilde{v})\} = \bigsqcup\{v \mid \alpha(v) \sqsubseteq \tilde{v}\}$. Assume that both $\langle \alpha, \gamma_1 \rangle$ and $\langle \alpha, \gamma_2 \rangle$ are Galois connections, then $\gamma_1(\tilde{v}) = \bigsqcup\{v \mid v \sqsubseteq \gamma_1(\tilde{v})\} = \bigsqcup\{v \mid \alpha(v) \sqsubseteq \tilde{v}\} = \bigsqcup\{v \mid v \sqsubseteq \gamma_2(\tilde{v})\} = \gamma_2(\tilde{v})$, and thus, $\gamma_1 = \gamma_2$. This shows that α uniquely determines γ. Similarly, it must be that $\alpha(v) = \tilde{\bigsqcap}\{\tilde{v} \mid \alpha(v) \tilde{\sqsubseteq} \tilde{v}\} = \tilde{\bigsqcap}\{\tilde{v} \mid v \tilde{\sqsubseteq} \gamma(\tilde{v})\}$. This shows that γ uniquely determines α.

To show 2, consider $V' \subseteq V$, then

$$\begin{aligned}
\alpha(\textstyle\bigsqcup V') \tilde{\sqsubseteq} \tilde{v} &\overset{Th.\ 3.13}{\Longleftrightarrow} \textstyle\bigsqcup V' \sqsubseteq \gamma(\tilde{v}) \\
&\overset{calc.}{\Longleftrightarrow} \forall v \in V' : v \sqsubseteq \gamma(\tilde{v}) \\
&\overset{Th.\ 3.13}{\Longleftrightarrow} \forall v \in V' : \alpha(v) \sqsubseteq \tilde{v} \\
&\overset{calc.}{\Longleftrightarrow} \tilde{\textstyle\bigsqcup}\{\alpha(v) \mid v \in V'\} \sqsubseteq \tilde{v}
\end{aligned}$$

and it follows that $\alpha(\bigsqcup V') = \tilde{\bigsqcup}\{\alpha(v) \mid v \in V'\}$.

The proof that $\gamma(\tilde{\bigsqcap}\tilde{V}') = \bigsqcap\{\gamma(\tilde{v}) \mid \tilde{v} \in \tilde{V}'\}$ is analogous. ∎

Thus, by Lemma 3.15, it suffices to specify either a completely additive abstraction function or a completely multiplicative concretization function in order to obtain a Galois connection.

Lemma 3.15 (Galois connection – Existence):
If $V = \langle V, \sqsubseteq, \bigsqcup, \bigsqcap, \bot, \top\rangle$ and $\tilde{V} = \langle \tilde{V}, \tilde{\sqsubseteq}, \tilde{\bigsqcup}, \tilde{\bigsqcap}, \tilde{\bot}, \tilde{\top}\rangle$ are complete lattices, and

1. *$\alpha : V \to \tilde{V}$ is completely additive, then there exists a $\gamma : \tilde{V} \to V$ such that $\langle \alpha, \gamma\rangle$ is a Galois connection.*
2. *$\gamma : \tilde{V} \to V$ is completely multiplicative, then there exists an $\alpha : V \to \tilde{V}$ such that $\langle \alpha, \gamma\rangle$ is a Galois connection.* □

PROOF (c.f., [62]). Assume that $V = \langle V, \sqsubseteq, \bigsqcup, \bigsqcap, \bot, \top\rangle$ and $\tilde{V} = \langle \tilde{V}, \tilde{\sqsubseteq}, \tilde{\bigsqcup}, \tilde{\bigsqcap}, \tilde{\bot}, \tilde{\top}\rangle$ are complete lattices, $v \in V$ and $\tilde{v} \in \tilde{V}$.

To show 1, assume that α is completely additive and define γ by:

$$\gamma(\tilde{v}) = \bigsqcup\{v' \mid \alpha(v') \tilde{\sqsubseteq} \tilde{v}\}$$

Then it must be that $\alpha(v) \tilde{\sqsubseteq} \tilde{v} \Rightarrow v \in \{v' \mid \alpha(v') \tilde{\sqsubseteq} \tilde{v}\} \Rightarrow v \sqsubseteq \gamma(\tilde{v})$, where the last implication follows from the definition of γ. For the other direction, first observe that $v \sqsubseteq \gamma(\tilde{v}) \Rightarrow \alpha(v) \tilde{\sqsubseteq} \alpha(\gamma(\tilde{v}))$ since α is completely additive and thus monotone. Then,

$$\begin{aligned}\alpha(\gamma(\tilde{v})) &= \alpha(\bigsqcup\{v' \mid \alpha(v') \tilde{\sqsubseteq} \tilde{v}\}) \\ &= \tilde{\bigsqcup}\{\alpha(v') \mid \alpha(v') \tilde{\sqsubseteq} \tilde{v}\} \\ &\tilde{\sqsubseteq} \tilde{v}\end{aligned}$$

and so $v \sqsubseteq \gamma(\tilde{v}) \Rightarrow \alpha(v) \tilde{\sqsubseteq} \tilde{v}$. Thus, $\langle \alpha, \gamma\rangle$ is a Galois connection (Theorem 3.13).

The proof of 2 is similar. ∎

3.4 Constructing Galois Connections

A Galois connection can be constructed in several ways. The following theorems (except Theorem 3.21) specify some of them.

The Cartesian product can be used to combine two existing Galois connections (Theorem 3.16).

Theorem 3.16 (Galois connection – Independent attribute method):
If $\langle \alpha_1 : V_1 \to D_1, \gamma_1 : D_1 \to V_1 \rangle$ *and* $\langle \alpha_2 : V_2 \to D_2, \gamma_2 : D_2 \to V_2 \rangle$ *are Galois connections, then so is* $\langle \alpha : (V_1 \times V_2) \to (D_1 \times D_2), \gamma : (D_1 \times D_2) \to (V_1 \times V_2) \rangle$, *where*

$$\begin{cases} \alpha((v_1, v_2)) = (\alpha_1(v_1), \alpha_2(v_2)) \\ \gamma((d_1, d_2)) = (\gamma_1(d_1), \gamma_2(d_2)) \end{cases}$$

and $(v_1, v_2) \in V_1 \times V_2$ *and* $(d_1, d_2) \in D_1 \times D_2$. □

PROOF (c.f., [62]). Assume that $\langle \alpha_1 : V_1 \to D_1, \gamma_1 : D_1 \to V_1 \rangle$ and $\langle \alpha_2 : V_2 \to D_2, \gamma_2 : D_2 \to V_2 \rangle$ are Galois connections, $(v_1, v_2) \in V_1 \times V_2$ and $(d_1, d_2) \in D_1 \times D_2$. Note that $V_1 \times V_2$ and $D_1 \times D_2$ are complete lattices (Theorem 3.6).

First calculate the following.

$$\begin{aligned} \alpha((v_1, v_2)) \sqsubseteq_D (d_1, d_2) &\overset{\text{Def. } \alpha}{\iff} (\alpha_1(v_1), \alpha_2(v_2)) \sqsubseteq_D (d_1, d_2) \\ &\overset{\text{calc.}}{\iff} \alpha_1(v_1) \sqsubseteq_{D_1} d_1 \wedge \alpha_2(v_2) \sqsubseteq_{D_2} d_2 \\ &\overset{\text{Th. 3.13}}{\iff} v_1 \sqsubseteq_{V_1} \gamma_1(d_1) \wedge v_2 \sqsubseteq_{V_2} \gamma_2(d_2) \\ &\overset{\text{calc.}}{\iff} (v_1, v_2) \sqsubseteq_V (\gamma_1(d_1), \gamma_2(d_2)) \\ &\overset{\text{Def. } \gamma}{\iff} (v_1, v_2) \sqsubseteq_V \gamma((d_1, d_2)) \end{aligned}$$

Then, using Theorem 3.13, the result follows. ■

The Cartesian product can also be used on lifted sets (Theorem 3.17).

Theorem 3.17 (Galois connection – Lifted independent attribute method):
If $\langle \alpha_1 : \mathscr{P}(V_1) \to D_1, \gamma_1 : D_1 \to \mathscr{P}(V_1) \rangle$ *and* $\langle \alpha_2 : \mathscr{P}(V_2) \to D_2, \gamma_2 : D_2 \to \mathscr{P}(V_2) \rangle$ *are Galois connections, then so is* $\langle \alpha : \mathscr{P}(V_1 \times V_2) \to (D_1 \times D_2), \gamma : (D_1 \times D_2) \to \mathscr{P}(V_1 \times V_2) \rangle$, *where*

$$\begin{cases} \alpha(V) = (\alpha_1(\{v_1 \in V_1 \mid \exists v_2 \in V_2 : (v_1, v_2) \in V\}), \\ \qquad\qquad \alpha_2(\{v_2 \in V_2 \mid \exists v_1 \in V_1 : (v_1, v_2) \in V\})) \\ \gamma((d_1, d_2)) = \gamma_1(d_1) \times \gamma_2(d_2) \end{cases}$$

and $V \subseteq V_1 \times V_2$ *and* $(d_1, d_2) \in D_1 \times D_2$. □

PROOF. Assume that $\langle \alpha_1 : \mathscr{P}(V_1) \to D_1, \gamma_1 : D_1 \to \mathscr{P}(V_1) \rangle$ and $\langle \alpha_2 : \mathscr{P}(V_2) \to D_2, \gamma_2 : D_2 \to \mathscr{P}(V_2) \rangle$ are Galois connections, $V \subseteq V_1 \times V_2$ and $(d_1, d_2) \in D_1 \times D_2$. Note that $\mathscr{P}(V_1 \times V_2)$ and $D_1 \times D_2$ are complete lattices (Theorems 3.5 and 3.6).

First, calculate

$$\begin{array}{lll}
\alpha(V) \sqsubseteq (d_1,d_2) & \overset{Def.\ \alpha}{\Longleftrightarrow} & (\alpha_1(V_1'),\alpha_2(V_2')) \sqsubseteq (d_1,d_2) \\
& \overset{calc.}{\Longleftrightarrow} & \alpha_1(V_1') \sqsubseteq_1 d_1 \wedge \alpha_2(V_2') \sqsubseteq_2 d_2 \\
& \overset{Th.\ 3.13}{\Longleftrightarrow} & V_1' \subseteq \gamma_1(d_1) \wedge V_2' \subseteq \gamma_2(d_2) \\
& \overset{calc.}{\Longleftrightarrow} & V_1' \times V_2' \subseteq \gamma_1(d_1) \times \gamma_2(d_2) \\
& \overset{Def.\ \gamma}{\Longleftrightarrow} & V_1' \times V_2' \subseteq \gamma((d_1,d_2)) \\
& \overset{V \subseteq V_1' \times V_2'}{\Longleftrightarrow} & V \subseteq \gamma((d_1,d_2))
\end{array}$$

where $V_1' = \{v_1 \in V_1 \mid \exists v_2 \in V_2 : (v_1,v_2) \in V\}$ and $V_2' = \{v_2 \in V_2 \mid \exists v_1 \in V_1 : (v_1,v_2) \in V\}$. Then, using Theorem 3.13, the result follows. ■

Both the concrete and abstract domains of an existing Galois connection can be lifted to derive a new Galois connection (Theorem 3.20). Note that Lemmas 3.18 and 3.19 give that the specified abstraction and concretization functions are monotone.

Lemma 3.18 (Monotonicity of $\alpha_{\mathscr{P}}$):
The function $\alpha_{\mathscr{P}} : \mathscr{P}(V) \to \mathscr{P}(D)$, defined as

$$\alpha_{\mathscr{P}}(V') = \{\alpha(v) \mid v \in V'\}$$

where $V' \subseteq V$, α is monotone and $\alpha : V \to D$, is monotone. □

PROOF. This proof amounts to showing that $\forall V', V'' \in \mathscr{P}(V) : (V' \subseteq V'' \Rightarrow \alpha_{\mathscr{P}}(V') \subseteq \alpha_{\mathscr{P}}(V''))$.

Assume that $V', V'' \in \mathscr{P}(V)$ and that $V' \subseteq V''$. Then, by definition:

$$\begin{array}{lll}
\alpha_{\mathscr{P}}(V'') & \overset{Def.\ \alpha_{\mathscr{P}}}{=} & \{\alpha(v) \mid v \in V''\} \\
& \overset{calc.}{=} & \{\alpha(v) \mid v \in V' \cup (V'' \setminus V')\} \\
& \overset{calc.}{=} & \{\alpha(v) \mid v \in V'\} \cup \{\alpha(v) \mid v \in V'' \setminus V'\} \\
& \overset{calc.}{\supseteq} & \{\alpha(v) \mid v \in V'\} \\
& \overset{Def.\ \alpha_{\mathscr{P}}}{=} & \alpha_{\mathscr{P}}(V')
\end{array}$$

where the rewriting of $\alpha(V'')$ and the set splitting are possible since $V' \subseteq V''$ and α is monotone.

Thus, it has been shown that $\alpha_{\mathscr{P}}$ is monotone. ■

Lemma 3.19 (Monotonicity of $\gamma_{\mathscr{P}}$):
The function $\gamma_{\mathscr{P}} : \mathscr{P}(D) \to \mathscr{P}(V)$, defined as

$$\gamma_{\mathscr{P}}(D') = \{v \in V \mid \alpha(v) \in D'\}$$

where $D' \subseteq D$, α is monotone and $\gamma : D \to V$, is monotone. □

PROOF. This proof amounts to showing that $\forall D', D'' \in \mathscr{P}(D) : (D' \subseteq D'' \Rightarrow \gamma_{\mathscr{P}}(D') \subseteq \gamma_{\mathscr{P}}(D''))$.

Assume that $D', D'' \in \mathscr{P}(D)$ and that $D' \subseteq D''$. Then, by definition:

$$\begin{aligned}
\gamma_{\mathscr{P}}(D'') &\overset{Def.\ \gamma_{\mathscr{P}}}{=} \{v \in V \mid \alpha(v) \in D''\} \\
&\overset{calc.}{=} \{v \in V \mid \alpha(v) \in D' \cup (D'' \setminus D')\} \\
&\overset{calc.}{=} \{v \in V \mid \alpha(v) \in D'\} \cup \{v \in V \mid \alpha(v) \in D'' \setminus D'\} \\
&\overset{calc.}{\supseteq} \{v \in V \mid \alpha(v) \in D'\} \\
&\overset{Def.\ \gamma_{\mathscr{P}}}{=} \gamma_{\mathscr{P}}(D')
\end{aligned}$$

where the rewriting of D'' and the set splitting are possible since $D' \subseteq D''$ and α is monotone.

Thus, $\gamma_{\mathscr{P}}(D') \subseteq \gamma_{\mathscr{P}}(D'')$, and hence it has been shown that $\gamma_{\mathscr{P}}$ is monotone. ■

Theorem 3.20 (Galois connection – Double lifting):
If $\langle \alpha : V \to D, \gamma : D \to V \rangle$ is a Galois connection, then so is $\langle \alpha_{\mathscr{P}} : \mathscr{P}(V) \to \mathscr{P}(D), \gamma_{\mathscr{P}} : \mathscr{P}(D) \to \mathscr{P}(V) \rangle$, where

$$\begin{cases} \alpha_{\mathscr{P}}(V') = \{\alpha(v) \mid v \in V'\} \\ \gamma_{\mathscr{P}}(D') = \{v \in V \mid \alpha(v) \in D'\} \end{cases}$$

and $V' \subseteq V$ and $D' \subseteq D$. □

PROOF. Assume that $\langle \alpha : V \to D, \gamma : D \to V \rangle$ is a Galois connection. Note that $\mathscr{P}(V)$ and $\mathscr{P}(D)$ are complete lattices (Theorem 3.5).

Since $\alpha_{\mathscr{P}}$ and $\gamma_{\mathscr{P}}$ are monotone (Lemmas 3.18 and 3.19, respectively), this proof amounts to showing that (c.f., Definition 3.9)

1. $\gamma_{\mathscr{P}}(\alpha_{\mathscr{P}}(V')) \supseteq V'$
2. $\alpha_{\mathscr{P}}(\gamma_{\mathscr{P}}(D')) \subseteq D'$

where $V' \subseteq V$ and $D' \subseteq D$. Note that both cases trivially hold if $V' = \emptyset$ or $D' = \emptyset$, which corresponds to the bottom elements in the two lattices. Therefore, assume that $V' \neq \emptyset$ and $D' \neq \emptyset$.

For case 1, assume that $V' \subseteq V$. Then, by definition:

$$\gamma_{\mathscr{P}}(\alpha_{\mathscr{P}}(V')) = \{v \in V \mid \alpha(v) \in \{\alpha(v') \mid v' \in V'\}\}$$

Assume that $v'' \in V'$, then it must be that $\alpha(v'') \in \{\alpha(v') \mid v' \in V'\}$. But, then $v'' \in \gamma_{\mathscr{P}}(\alpha_{\mathscr{P}}(V'))$ and thus $\gamma_{\mathscr{P}}(\alpha_{\mathscr{P}}(V')) \supseteq V'$.

For case 2, assume that $D' \subseteq D$. Then, by definition:

$$\alpha_{\mathscr{P}}(\gamma_{\mathscr{P}}(D')) = \{\alpha(v) \mid v \in \{v' \in V \mid \alpha(v') \in D'\}\}$$

Assume that $d \in \alpha_{\mathscr{P}}(\gamma_{\mathscr{P}}(D'))$. Then it must be that $\exists v \in \{v' \in V \mid \alpha(v') \in D'\} : d = \alpha(v)$. Hence, for that v, it must be that $\alpha(v) \in D'$, and therefore, $d \in D'$. Thus, $\alpha_{\mathscr{P}}(\gamma_{\mathscr{P}}(D')) \subseteq D'$. ∎

It might be tempting to use the definition of $\alpha_{\mathscr{P}}$ and $\gamma_{\mathscr{P}}$ as given in Theorem 3.21, but as the theorem shows, this does *not* result in a Galois connection.

Theorem 3.21 (*Not* a Galois connection – Double lifting):
If $\langle \alpha : V \to D, \gamma : D \to V \rangle$ *is a Galois connection, then* $\langle \alpha'_{\mathscr{P}} : \mathscr{P}(V) \to \mathscr{P}(D), \gamma'_{\mathscr{P}} : \mathscr{P}(D) \to \mathscr{P}(V) \rangle$ *is* not *a Galois connection, where*

$$\begin{cases} \alpha'_{\mathscr{P}}(V') = \{\alpha(v) \mid v \in V'\} \\ \gamma'_{\mathscr{P}}(D') = \{\gamma(d) \mid d \in D'\} \end{cases}$$

and $V' \subseteq V$ *and* $D' \subseteq D$. □

PROOF. Assume that $\langle \alpha : V \to D, \gamma : D \to V \rangle$ is a Galois connection. From the definition of $\alpha'_{\mathscr{P}}$ and $\gamma'_{\mathscr{P}}$, it clearly follows that they are monotone since α and γ are (c.f., Lemma 3.18).

By way of contradiction, assume that $\langle \alpha'_{\mathscr{P}}, \gamma'_{\mathscr{P}} \rangle$ is a Galois connection. Then, by Definition 3.9, $\gamma'_{\mathscr{P}}(\alpha'_{\mathscr{P}}(V')) \supseteq V'$. A closer look at $\gamma'_{\mathscr{P}}(\alpha'_{\mathscr{P}}(V'))$ reveals that:

$$\gamma'_{\mathscr{P}}(\alpha'_{\mathscr{P}}(V')) = \{\gamma(d) \mid d \in \{\alpha(v) \mid v \in V'\}\}$$

Assume that $v' \in V'$, then $v' \in \gamma'_{\mathscr{P}}(\alpha'_{\mathscr{P}}(V'))$ since $\gamma'_{\mathscr{P}}(\alpha'_{\mathscr{P}}(V')) \supseteq V'$. This means that $\exists d' \in \{\alpha(v) \mid v \in V'\} : d' = \alpha(v')$ and hence, for this d', $\exists v'' \in \{\gamma(d) \mid d \in \{\alpha(v) \mid v \in V'\}\} : v' = v'' = \gamma(d') = \gamma(\alpha(v'))$.

But, since $\langle \alpha, \gamma \rangle$ is a Galois connection, $\gamma(\alpha(v')) \sqsupseteq v'$. This means that it could be the case that $\gamma(\alpha(v')) \sqsupset v'$, and thus $v' \neq v''$, which means that $\gamma'_{\mathscr{P}}(\alpha'_{\mathscr{P}}(V')) \not\supseteq V'$ is possible. Thus, $\langle \alpha'_{\mathscr{P}}, \gamma'_{\mathscr{P}} \rangle$ is not a Galois connection. ∎

The domains of a Galois connection can be extended to spaces of (total or monotone) functions (Theorem 3.22).

Theorem 3.22 (Galois connection – Function space):
If $\langle \alpha : V \to D, \gamma : D \to V \rangle$ *is a Galois connection, then so is* $\langle \alpha' : (S \to V) \to (S \to D), \gamma' : (S \to D) \to (S \to V) \rangle$ *for some set,* S*, where:*

$$\begin{cases} \alpha'(f) = \alpha \circ f \\ \gamma'(g) = \gamma \circ g \end{cases}$$

□

PROOF (c.f., [62]). Assume that $\langle \alpha : V \to D, \gamma : D \to V \rangle$ is a Galois connection and that S is a set. Note that $S \to V$ and $S \to D$ are complete lattices (Theorems 3.7 and 3.8).

First note that α' and γ' are monotone since α and γ are. Furthermore, since $\langle \alpha, \gamma \rangle$ is a Galois connection,

$$\gamma'(\alpha'(f)) = \gamma \circ \alpha \circ f \sqsupseteq f$$

and

$$\alpha'(\gamma'(g)) = \alpha \circ \gamma \circ g \sqsubseteq g$$

and, thus, the theorem holds. ■

A lifted concrete domain of a Galois connection can be extended to a lifted space of (total or monotone) functions when also extending the abstract domain (Theorem 3.24). Note that Lemma 3.23 gives that the concretization function is monotone.

Lemma 3.23 (Monotonicity of γ_s):
The function $\gamma_s : (S \to D) \to \mathscr{P}(S \to V)$*, defined as*

$$\gamma_s(d) = \begin{cases} S \to V & \textbf{if } d = \tilde{\top} \\ \emptyset & \textbf{if } d = \tilde{\bot} \\ \{\lambda s \in S.v \mid v \in \gamma(d\ s)\} & \textbf{otherwise} \end{cases}$$

for some set S *and complete lattices* V *and* D*, is monotone, given that* $\gamma : D \to \mathscr{P}(V)$ *is a monotone function and* $d \in S \to D$*.* □

PROOF. This proof amounts to showing that $\forall d', d'' \in S \to D : (d' \sqsubseteq d'' \Rightarrow \gamma_s(d') \subseteq \gamma_s(d''))$, which is trivially the case if $d' = \tilde{\bot}$ or $d'' = \tilde{\top}$.

Assume that $\gamma : D \to \mathscr{P}(V)$ is a monotone function, $d', d'' \in S \to D$ and that $d' \sqsubseteq d'' \wedge d' \neq \tilde{\bot} \wedge d'' \neq \tilde{\top}$. Then, by definition:

$$\begin{cases} \gamma_s(d') = \{\lambda s \in S.v \mid v \in \gamma(d'\ s)\} \\ \gamma_s(d'') = \{\lambda s \in S.v \mid v \in \gamma(d''\ s)\} \end{cases}$$

Since γ is monotone, it must be that $\forall s \in S : \gamma(d'\ s) \subseteq \gamma(d''\ s)$. This means that

$$\begin{aligned} \gamma_s(d'') &= \{\lambda s \in S.v \mid v \in \gamma(d'\ s) \cup (\gamma(d''\ s) \setminus \gamma(d'\ s))\} \\ &\supseteq \{\lambda s \in S.v \mid v \in \gamma(d'\ s)\} \cup \{\lambda s \in S.v \mid v \in (\gamma(d''\ s) \setminus \gamma(d'\ s))\} \\ &= \gamma_s(d') \cup \{\lambda s \in S.v \mid v \in (\gamma(d''\ s) \setminus \gamma(d'\ s))\} \end{aligned}$$

and thus, trivially, $\gamma_s(d') \subseteq \gamma_s(d'')$. ■

Theorem 3.24 (Galois connection – Lifted function space):
If $\langle \alpha : \mathscr{P}(V) \to D, \gamma : D \to \mathscr{P}(V) \rangle$ is a Galois connection, then so is $\langle \alpha_s : \mathscr{P}(S \to V) \to (S \to D), \gamma_s : (S \to D) \to \mathscr{P}(S \to V) \rangle$, for some set S, where

$$\begin{cases} \alpha_s(V') = \begin{cases} \tilde{\top} & \textbf{if } V' = S \to V \\ \tilde{\bot} & \textbf{if } V' = \emptyset \\ \lambda s \in S.\alpha(\{v'\ s \mid v' \in V'\}) & \textbf{otherwise} \end{cases} \\ \gamma_s(d) = \begin{cases} S \to V & \textbf{if } d = \tilde{\top} \\ \emptyset & \textbf{if } d = \tilde{\bot} \\ \{\lambda s \in S.v \mid v \in \gamma(d\ s)\} & \textbf{otherwise} \end{cases} \end{cases}$$

and $V' \subseteq S \to V$ and $d \in S \to D$. □

PROOF. Assume that $\langle \alpha : \mathscr{P}(V) \to D, \gamma : D \to \mathscr{P}(V) \rangle$ is a Galois connection, S is a set, $V' \subseteq S \to V$ and $d \in S \to D$. Note that $\mathscr{P}(S \to V)$ and $S \to D$ are complete lattices (Theorems 3.5, 3.7 and 3.8).

First note that:

$$\begin{aligned} &\gamma_s(\alpha_s(S \to V)) = \gamma_s(\tilde{\top}) = S \to V \supseteq S \to V \\ &\gamma_s(\alpha_s(\emptyset)) = \gamma_s(\tilde{\bot}) = \emptyset \supseteq \emptyset \\ &\alpha_s(\gamma_s(\tilde{\top})) = \alpha_s(S \to V) = \tilde{\top} \sqsubseteq \tilde{\top} \\ &\alpha_s(\gamma_s(\tilde{\bot})) = \alpha_s(\emptyset) = \tilde{\bot} \sqsubseteq \tilde{\bot} \end{aligned}$$

Then note that γ_s is monotone (Lemma 3.23) and calculate the following.

$$\begin{aligned}
\alpha_s(V') \sqsubseteq d &\overset{\text{Def. } \alpha_s}{\iff} \lambda s \in S.\alpha(\{v'\ s \mid v' \in V'\}) \sqsubseteq d \\
&\overset{\gamma_s \text{ mon.}}{\iff} \gamma_s(\lambda s \in S.\alpha(\{v'\ s \mid v' \in V'\})) \subseteq \gamma_s(d) \\
&\overset{\text{Def. } \gamma_s}{\iff} \{\lambda s \in S.v \mid v \in \gamma((\lambda s' \in S.\alpha(\{v'\ s' \mid v' \in V'\}))\ s)\} \subseteq \gamma_s(d) \\
&\overset{\text{calc.}}{\iff} \{\lambda s \in S.v \mid v \in \gamma(\alpha(\{v'\ s \mid v' \in V'\}))\} \subseteq \gamma_s(d) \\
&\overset{\lambda v.v \sqsubseteq \gamma\circ\alpha}{\iff} \{\lambda s \in S.v \mid v \in \{v'\ s \mid v' \in V'\}\} \subseteq \\
&\qquad\qquad \{\lambda s \in S.v \mid v \in \gamma(\alpha(\{v'\ s \mid v' \in V'\}))\} \subseteq \gamma_s(d) \\
&\overset{\text{calc.}}{\iff} \{\lambda s \in S.v \mid v \in \{v'\ s \mid v' \in V'\}\} \subseteq \gamma_s(d) \\
&\overset{\text{calc.}}{\iff} \{\lambda s \in S.(v'\ s) \mid v' \in V'\} \subseteq \gamma_s(d) \\
&\overset{\text{calc.}}{\iff} \{v' \mid v' \in V'\} \subseteq \{\lambda s \in S.(v'\ s) \mid v' \in V'\} \subseteq \gamma_s(d) \\
&\overset{\text{calc.}}{\iff} \{v' \mid v' \in V'\} \subseteq \gamma_s(d) \\
&\overset{\text{calc.}}{\iff} V' \subseteq \gamma_s(d)
\end{aligned}$$

Then, using Theorem 3.13, the result follows. ∎

The domains of a Galois connection can be indexed with the elements from some set (Theorem 3.25).

Theorem 3.25 (Galois connection – Indexing):
If $\langle \alpha : V \to D, \gamma : D \to V \rangle$ is a Galois connection, then so is $\langle \alpha' : (S \times V) \to (S \times D), \gamma' : (S \times D) \to (S \times V) \rangle$, for some set $S \ni s$ (with the partial order $=$), where

$$\begin{cases} \alpha'((s,v)) = (s, \alpha(v)) \\ \gamma'((s',d)) = (s', \gamma(d)) \end{cases}$$

and $(s,v) \in S \times V$ and $(s',d) \in S \times D$. The top elements, $\top'_V$ and $\top'_D$, correspond to the elements (s,v) and (s,d) for some $s \in S$, respectively, where $\alpha(v) = \top_D$ and $\gamma(d) = \top_V$. The bottom elements are defined in a corresponding manner.

α' and γ' for $\langle \alpha' : (V \times S) \to (D \times S), \gamma' : (D \times S) \to (V \times S) \rangle$ are defined similarly. □

PROOF. Assume that $\langle \alpha : V \to D, \gamma : D \to V \rangle$ is a Galois connection, S is a set, $(s,v) \in S \times V$ and $(s',d) \in S \times D$.

First note that:

$$\begin{aligned}
\gamma'(\alpha'(\top'_V)) &= \gamma'(\tilde{\top}'_D) = \top'_V \sqsupseteq'_V \top'_V \\
\gamma'(\alpha'(\bot'_V)) &= \gamma'(\tilde{\bot}'_D) = \bot'_V \sqsupseteq'_V \bot'_V \\
\alpha'(\gamma'(\tilde{\top}'_D)) &= \alpha'(\top'_V) = \tilde{\top}'_D \sqsubseteq'_D \tilde{\top}'_D \\
\alpha'(\gamma'(\tilde{\bot}'_D)) &= \alpha'(\bot'_V) = \tilde{\bot}'_D \sqsubseteq'_D \tilde{\bot}'_D
\end{aligned}$$

Then, calculate the following.

$$\begin{aligned}
\alpha'((s,v)) \sqsubseteq_{SD} (s',d) &\overset{\text{Def. } \alpha'}{\Longleftrightarrow} (s,\alpha(v)) \sqsubseteq_{SD} (s',d) \\
&\overset{\text{calc.}}{\Longleftrightarrow} s = s' \wedge \alpha(v) \sqsubseteq_D d \\
&\overset{\text{Th. } 3.13}{\Longleftrightarrow} s = s' \wedge v \sqsubseteq_V \gamma(d) \\
&\overset{\text{calc.}}{\Longleftrightarrow} (s,v) \sqsubseteq_{SV} (s',\gamma(d)) \\
&\overset{\text{Def. } \gamma'}{\Longleftrightarrow} (s,v) \sqsubseteq_{SV} \gamma'((s',d))
\end{aligned}$$

Now, using Theorem 3.13, the result follows.

The proof for $\langle \alpha' : (V \times S) \to (D \times S), \gamma' : (D \times S) \to (V \times S) \rangle$ being a Galois connection is conducted analogously. ■

3.5 Constructing Galois Insertions

A Galois insertion $\langle \alpha, \gamma \rangle$ between two domains, D and $\tilde{D}$, can be constructed by following steps 1-5 below [23].

1. A domain, D, with a partial order, $\sqsubseteq$, a least (bottom) element, $\bot$, a greatest (top) element, $\top$, a greatest lower bound, $\bigsqcap$, and a least upper bound, $\bigsqcup$, so that $\langle D, \sqsubseteq, \bigsqcup, \bigsqcap, \bot, \top \rangle$ is a complete lattice must be given.
2. Define a domain $\tilde{D}$ and a monotone concretization function $\gamma : \tilde{D} \to D$.
3. Define the partial order $\tilde{\sqsubseteq}$ for $\tilde{D}$.
4. The greatest lower bound $\tilde{\bigsqcap}$ and the least upper bound $\tilde{\bigsqcup}$ must exist for all subsets of $\tilde{D}$. Then, by definition, $\langle \tilde{D}, \tilde{\sqsubseteq}, \tilde{\bigsqcup}, \tilde{\bigsqcap}, \tilde{\bot}, \tilde{\top} \rangle$ is a complete lattice.
5. Define the abstraction function $\alpha : D \to \tilde{D}$, which must be monotone.

Assuming that the domains D and $\tilde{D}$ and the monotone concretization function, γ, are defined, the partial ordering $\tilde{\sqsubseteq}$ can easily be defined as given by Definition 3.26 [23].

Definition 3.26 (Partial order):
$\tilde{\sqsubseteq}$ is a partial order for the domain $\tilde{D}$ iff $\forall \tilde{d_1}, \tilde{d_2} \in \tilde{D} : (\tilde{d_1} \tilde{\sqsubseteq} \tilde{d_2} \iff \gamma(\tilde{d_1}) \sqsubseteq \gamma(\tilde{d_2}))$. □

Based on this definition of the partial order, the greatest lower bound and least upper bound can be defined as given by Definitions 3.27 and 3.28, respectively [23].

Definition 3.27 (Greatest lower bound):
The element $\tilde{d} \in \tilde{D}$ is a lower bound of $\tilde{D'} \subseteq \tilde{D}$ iff $\forall \tilde{d'} \in \tilde{D'} : \tilde{d} \tilde{\sqsubseteq} \tilde{d'}$. The element $\tilde{d} \in \tilde{D}$ is the greatest lower bound of $\tilde{D'} \subseteq \tilde{D}$ ($\tilde{d} = \tilde{\bigsqcap} \tilde{D'}$) iff $\tilde{d}$ is a lower bound of $\tilde{D'}$ and for all other lower bounds $\tilde{d'}$ of $\tilde{D'}$, $\tilde{d'} \tilde{\sqsubseteq} \tilde{d}$. □

Definition 3.28 (Least upper bound):
The element $\tilde{d} \in \tilde{D}$ is an upper bound of $\tilde{D'} \subseteq \tilde{D}$ iff $\forall \tilde{d'} \in \tilde{D'} : \tilde{d'} \tilde{\sqsubseteq} \tilde{d}$. The element $\tilde{d} \in \tilde{D}$ is the least upper bound of $\tilde{D'} \subseteq \tilde{D}$ ($\tilde{d} = \tilde{\bigsqcup} \tilde{D'}$) iff $\tilde{d}$ is an upper bound of $\tilde{D'}$ and for all other upper bounds $\tilde{d'}$ of $\tilde{D'}$, $\tilde{d} \tilde{\sqsubseteq} \tilde{d'}$. □

The abstraction function α can be defined based on the definition of the greatest lower bound operator as given by Definition 3.29 [23].

Definition 3.29 (Abstraction function, α):
Given two domains D and $\tilde{D}$ and a monotone concretization function $\gamma : \tilde{D} \to D$, the abstraction function $\alpha : D \to \tilde{D}$ is defined by:

$$\alpha(d) = \tilde{\bigsqcap} \{\tilde{d} \mid d \sqsubseteq \gamma(\tilde{d})\}$$

where $d \in D$ and $\tilde{d} \in \tilde{D}$. □

Alternatively, assuming that two domains and a monotone abstraction function have been defined, the concretization function γ can be defined based on the least upper bound operator as given by Definition 3.30 [23].

Definition 3.30 (Alternative definition – Concretization function, γ):
Given two domains D and $\tilde{D}$ and a monotone abstraction function $\alpha : D \to \tilde{D}$, the concretization function $\gamma : \tilde{D} \to D$ is defined by:

$$\gamma(\tilde{d}) = \bigsqcup \{d \mid \alpha(d) \tilde{\sqsubseteq} \tilde{d}\}$$

where $d \in D$ and $\tilde{d} \in \tilde{D}$. □

3.6 The Interval Domain

One example of an abstract domain for values is the interval domain [19, 23, 62]. The definition of an interval is given in Definition 3.31.

Definition 3.31 (Interval):
An interval is defined as $[n_1, n_2]$*, where* $n_1, n_2 \in \mathbf{Val} = \mathbb{Z} \cup \{-\infty, \infty\}$ *are the lower and upper bounds of the interval, respectively, and* $n_1 \leq n_2$*. Formally, the set of all intervals is defined as* $\mathbf{Intv} = \{\bot_{int}, \top_{int}\} \cup \{[n_1, n_2] \mid n_1 \leq n_2 \wedge n_1, n_2 \in \mathbf{Val}\}$*, where* $\bot_{int}$ *denotes an invalid interval and* $\top_{int}$ *is greater than any other element of* **Intv**. □

A Galois insertion will now be created between $\mathscr{P}(\mathbf{Val})$ and **Intv**, using the steps of Section 3.5. The concretization function $\gamma_{int} : \mathbf{Intv} \to \mathscr{P}(\mathbf{Val})$ is given by Definition 3.32.

Definition 3.32 (concretization of intervals):

$$\gamma_{int}(i) = \begin{cases} \mathbb{Z} \cup \{-\infty, \infty\} & \textbf{if } i = \top_{int} \\ \emptyset & \textbf{if } i = \bot_{int} \\ \{n \in \mathbf{Val} \mid n_1 \leq n \leq n_2\} & \textbf{otherwise } (\textit{i.e., } i = [n_1, n_2]) \end{cases}$$ □

The partial order relation for intervals, $\sqsubseteq_{int}$, is given by Definition 3.33 (using Definition 3.26).

Definition 3.33 (Partial order for intervals):

$$\begin{cases} i \sqsubseteq_{int} \top_{int} \\ \bot_{int} \sqsubseteq_{int} i \\ [n_1, n_2] \sqsubseteq_{int} [n_1', n_2'] \Longleftrightarrow n_1' \leq n_1 \wedge n_2 \leq n_2' \end{cases}$$ □

The greatest lower bound operator for intervals $\bigsqcap_{int}$ is defined as given by Definition 3.34 (using Definition 3.27).

Definition 3.34 (Greatest lower bound for intervals):

$$\begin{cases} i \sqcap_{int} \top_{int} = \top_{int} \sqcap_{int} i = i \\ i \sqcap_{int} \bot_{int} = \bot_{int} \sqcap_{int} i = \bot_{int} \\ [n_1, n_2] \sqcap_{int} [n_1', n_2'] = \\ \quad \begin{cases} [\max(\{n_1, n_1'\}), \min(\{n_2, n_2'\})] & \textbf{if } \max(\{n_1, n_1'\}) \leq \min(\{n_2, n_2'\}) \\ \bot_{int} & \textbf{otherwise} \end{cases} \end{cases}$$

□

The least upper bound operator for intervals $\bigsqcup_{int}$ is defined as given by Definition 3.35 (using Definition 3.28).

Definition 3.35 (Least upper bound for intervals):

$$\begin{cases} i \sqcup_{int} \top_{int} = \top_{int} \sqcup_{int} i = \top_{int} \\ i \sqcup_{int} \bot_{int} = \bot_{int} \sqcup_{int} i = i \\ [n_1, n_2] \sqcup_{int} [n_1', n_2'] = [\min(\{n_1, n_1'\}), \max(\{n_2, n_2'\})] \end{cases}$$ □

The abstraction function $\alpha_{int} : \mathscr{P}(\mathbf{Val}) \to \mathbf{Intv}$ is defined as given by Definition 3.36 (using Definition 3.29).

Definition 3.36 (Abstraction to interval):

$$\alpha_{int}(V) = \begin{cases} \top_{int} & \textbf{if } V = \mathbb{Z} \cup \{-\infty, \infty\} \\ \bot_{int} & \textbf{if } V = \emptyset \\ [\min(V), \max(V)] & \textbf{otherwise} \end{cases}$$ □

To show that $\langle \alpha_{int}, \gamma_{int} \rangle$ is a Galois insertion, it would suffice to show that γ_{int} is monotone, since the steps of Section 3.5 have been used. However, for clarity, the entire proof is given in the proof of Theorem 3.39. Note that Lemmas 3.37 and 3.38 give that γ_{int} and α_{int}, respectively, are monotone.

Lemma 3.37 (Monotonicity of γ_{int}):
The function $\gamma_{int} : \mathbf{Intv} \to \mathscr{P}(\mathbf{Val})$ is monotone. □

PROOF. It should be shown that $\forall i, i' \in \mathbf{Intv} : (i \sqsubseteq_{int} i' \Rightarrow \gamma_{int}(i) \subseteq \gamma_{int}(i'))$. Note that the proof is trivial for the case that $i = \bot_{int}$ or $i' = \top_{int}$.

Assume that $i = [n_1, n_2] \in \mathbf{Intv}$ and $i' = [n_1', n_2'] \in \mathbf{Intv}$, such that $i \sqsubseteq_{int} i'$. Further assume that $n \in \gamma_{int}(i)$. Then it must be the case that $n_1 \leq n \leq n_2$ (Definition 3.32). Since $i \sqsubseteq_{int} i'$, it must be the case that $n_1' \leq n_1 \leq n \leq n_2 \leq n_2'$ (Definition 3.33). But, then it must be that $n \in \gamma_{int}(i')$ (Definition 3.32), and thus, $\gamma_{int}(i) \subseteq \gamma_{int}(i')$. ■

Lemma 3.38 (Monotonicity of α_{int}):
The function $\alpha_{int} : \mathscr{P}(\mathbf{Val}) \to \mathbf{Intv}$ is monotone. □

PROOF. It should be shown that $\forall V, V' \in \mathscr{P}(\mathbf{Val}) : (V \subseteq V' \Rightarrow \alpha_{int}(V) \sqsubseteq_{int} \alpha_{int}(V'))$. Note that the proof is trivial for the case that $V = \emptyset$ or $V' = \mathbb{Z} \cup \{-\infty, \infty\}$.

Assume that $V, V' \in \mathscr{P}(\mathbf{Val})$, such that $V \subseteq V'$. Further assume that $\alpha_{int}(V) = [n_1, n_2]$ and $\alpha_{int}(V') = [n_1', n_2']$. Since $V \subseteq V'$, it must be that $\forall v \in V : \{v\} \subseteq V'$, and hence, $\{n_1, n_2\} \subseteq V'$. But then, it must be that $\min(V') = n_1' \leq n_1 = \min(V)$ and $\max(V) = n_2 \leq n_2' = \max(V')$, and thus, $[n_1, n_2] \sqsubseteq_{int} [n_1', n_2']$ (Definition 3.33), which means that $\alpha_{int}(V) \sqsubseteq_{int} \alpha_{int}(V')$.■

Theorem 3.39 (Galois insertion – Intervals):
$\langle \alpha_{int} : \mathscr{P}(\mathbf{Val}) \to \mathbf{Intv}, \gamma_{int} : \mathbf{Intv} \to \mathscr{P}(\mathbf{Val}) \rangle$ *is a Galois insertion.* □

PROOF. The proof amounts to showing that the constraints in Definition 3.10 are fulfilled by $\langle \alpha_{int}, \gamma_{int} \rangle$. Note that $\mathscr{P}(\mathbf{Val})$ and **Intv** are complete lattices [62].

According to Lemmas 3.37 and 3.38, γ_{int} and α_{int} are monotone. To show that $\alpha_{int}(\gamma_{int}(i)) = i$, assume that $i \in \mathbf{Intv}$.

- If $i = \top_{int}$, then $\gamma_{int}(i) = \mathbb{Z} \cup \{-\infty, \infty\}$. Thus, $\alpha_{int}(\gamma_{int}(i)) = \alpha_{int}(\mathbb{Z} \cup \{-\infty, \infty\}) = \top_{int} = i$.
- If $i = \bot_{int}$, then $\gamma_{int}(i) = \emptyset$. Thus, $\alpha_{int}(\gamma_{int}(i)) = \alpha_{int}(\emptyset) = \bot_{int} = i$.
- Otherwise (i.e., if $i = [n_1, n_2]$) then $\gamma_{int}(i) = \{n \in \mathbf{Val} \mid n_1 \leq n \leq n_2\}$. Thus, $\alpha_{int}(\gamma_{int}(i)) = \alpha_{int}(\{n \in \mathbf{Val} \mid n_1 \leq n \leq n_2\}) = [n_1, n_2] = i$.

To show that $\gamma_{int}(\alpha_{int}(V)) \supseteq V$, assume that $V \in \mathscr{P}(\mathbf{Val})$.

- If $V = \mathbb{Z} \cup \{-\infty, \infty\}$, then $\alpha_{int}(V) = \top_{int}$. Thus, $\gamma_{int}(\alpha_{int}(V)) = \gamma_{int}(\top_{int}) = \mathbb{Z} \cup \{-\infty, \infty\} \supseteq \mathbb{Z} \cup \{-\infty, \infty\} = V$.
- If $V = \emptyset$, then $\alpha_{int}(V) = \bot_{int}$. Thus, $\gamma_{int}(\alpha_{int}(V)) = \gamma_{int}(\bot_{int}) = \emptyset \supseteq \emptyset = V$.
- Otherwise, $\alpha_{int}(V) = [\min(V), \max(V)]$. Thus, $\gamma_{int}(\alpha_{int}(V)) = \gamma_{int}([\min(V), \max(V)]) = \{n \in \mathbf{Val} \mid \min(V) \leq n \leq \max(V)\} \supseteq V$. ■

Chapter 4

PPL: A Parallel Programming Language

In this chapter, PPL, a parallel programming language will be defined. The parallel entity of execution is referred to as a thread.

PPL provides both thread-private and globally shared memory, referred to as registers, $r \in \mathbf{Reg}$, and variables, $x \in \mathbf{Var}$, respectively, and arithmetical operations etc. within a thread can be performed using the values of the thread's registers. PPL also provides shared resources, referred to as locks, $lck \in \mathbf{Lck}$, that can be acquired in a mutually exclusive manner by the threads. The operations (statements) provided by the instruction set may have variable execution times. (C.f., multi-core CPUs, which have both local and global memory, a shared memory bus and atomic, i.e., mutually exclusive, operations.)

NOTE. *A summary of the notation and nomenclature used in this thesis can be found in Appendix A.*

The syntax of PPL, which is a set of operations using the discussed architectural features, is defined in Table 4.1. $\Pi \in \mathbf{Prg}$ denotes a program, which simply is a set of threads, i.e., $\Pi = \mathbf{Thrd} \in \mathscr{P}(\mathbf{ThrdID} \times \mathbf{Stm}) = \mathbf{Prg}$, where each thread, $\mathrm{T} \in \mathbf{Thrd}$, is a pair of a unique identifier, $d \in \mathbf{ThrdID}$, and a statement, $s \in \mathbf{Stm}$. This makes every thread unique and distinguishable from other

$$
\begin{aligned}
\Pi &::= \{\mathrm{T}_1, \ldots, \mathrm{T}_m\} \\
\mathrm{T} &::= (d, s) \\
s &::= [\texttt{halt}]^l \mid [\texttt{skip}]^l \mid [r := a]^l \mid [\texttt{if } b \texttt{ goto } l']^l \mid [\texttt{load } r \texttt{ from } x]^l \mid \\
&\quad\ [\texttt{store } r \texttt{ to } x]^l \mid [\texttt{lock } lck]^l \mid [\texttt{unlock } lck]^l \mid s_1 \texttt{ ; } s_2 \\
a &::= n \mid r \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \mid a_1 \,/\, a_2 \\
b &::= \texttt{true} \mid \texttt{false} \mid \texttt{!}\, b \mid b_1 \,\&\&\, b_2 \mid a_1 == a_2 \mid a_1 <= a_2
\end{aligned}
$$

Table 4.1: The Syntax of PPL.

threads, even if several threads consist of the same statement. To increase the readability of the semantics, it will be assumed that the axiom-statements (all statements except the sequentially composed statement, $s_1 \texttt{ ; } s_2$) of each thread are uniquely labeled with consecutive labels, $l \in \mathbf{Lbl}$, and stored in an array-like fashion in ascending order of their labels. $a \in \mathbf{Aexp}$ and $b \in \mathbf{Bexp}$ denote an arithmetic and a boolean expression, respectively, and $n \in \mathbf{Val}$ is an integer value.

Locks can be acquired in a mutually exclusive manner using `lock` and released using `unlock`. Values can be transferred between variables and registers using `load` and `store`. Conditional branching is performed using `if`. A register is assigned a value using :=. A no-operation is performed using `skip`. And, `halt` stops the execution of the issuing thread. `halt` must be the last statement of each thread in the program, but it could also occur anywhere "within" a thread. The semantics of PPL is formally defined in Section 4.2.

4.1 States & Configurations

A number of sub-states will be used when expressing how a set of given statements affects the state of the entire system when the statements are executed in parallel; i.e., when expressing the semantics of PPL. For each thread, T, of a program, there is an instance of the following states.

pc_{T} : $\mathbf{Lbl}_{\mathrm{T}}$ – a program counter that keeps track of which statement within the thread T that is active.

NOTE. *The tuple* $\langle pc_{\mathrm{T}_1}, \ldots, pc_{\mathrm{T}_m} \rangle$, *assuming that* $\mathbf{Thrd} = \{\mathrm{T}_1, \ldots, \mathrm{T}_m\}$, *defines a unique program point.*

$\mathbb{r}_{\mathrm{T}}$: $\mathbf{Reg}_{\mathrm{T}} \to \mathbf{Val}$ – a mapping from T's registers to their values.

t^{a}_{T} : **Time** – an absolute point in discrete time when the previous statement in T was executed.

For the program as such, there is an instance of the following states.

$\mathbb{x}$: $\mathbf{Var} \to \mathbf{Thrd} \to \mathscr{P}(\mathbf{Val} \times \mathbf{Time})$ – a mapping from variables to a set of writes; i.e., a pair of a value and an absolute point in time.

$\mathbb{l}$: $\mathbf{Lck} \to (\mathbf{Lck_{stt}} \times \mathbf{Thrd}_{\perp} \times \mathbf{Time} \times \mathbf{Thrd}_{\perp} \times \mathbf{Time})$ – a mapping from locks to their values; i.e., a state ($\mathbf{Lck_{stt}} = \{\mathit{unlocked}, \mathit{locked}\}$), a current owner ($\mathbf{Thrd}_{\perp} = \mathbf{Thrd} \cup \{\perp_{\mathit{thrd}}\}$), an absolute point in time when the lock must be taken by the current owner, a previous owner, and an absolute point in time when the lock was last released. For the case that no thread owns the lock, the owner is $\perp_{\mathit{thrd}}$.

NOTE. *The only information about locks that is needed in the concrete case is the current owner of each lock (c.f., Tables 4.2 and 4.3). The rest of the information is only necessary when expressing the abstract semantics (c.f., Chapter 5). However, the soundness of the abstract semantics is easier proven if this information is included in the concrete case as well.*

The types of the states $\mathbb{x}$ and $\mathbb{l}$ might look a bit peculiar at first glance; the need for their definitions will become apparent when defining an abstract interpretation of the PPL semantics in Section 5.8.

The above listed states, together with the threads of the program, will be referred to as a program state or configuration, $c \in \mathbf{Conf}$. **Conf** and c are defined as follows.

$$\begin{aligned}\mathbf{Conf} ::= {} & \mathscr{P}_{\mathrm{T} \in \mathbf{Thrd}}(\{\mathrm{T}\} \times \mathbf{Lbl}_{\mathrm{T}} \times (\mathbf{Reg}_{\mathrm{T}} \to \mathbf{Val}) \times \mathbf{Time}) \times \\ & (\mathbf{Var} \to \mathbf{Thrd} \to \mathscr{P}(\mathbf{Val} \times \mathbf{Time})) \times \\ & (\mathbf{Lck} \to (\mathbf{Lck_{stt}} \times \mathbf{Thrd}_{\perp} \times \mathbf{Time} \times \mathbf{Thrd}_{\perp} \times \mathbf{Time}))\end{aligned}$$

$$c ::= \langle [\mathrm{T}, pc_\mathrm{T}, \mathbb{r}_\mathrm{T}, t^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle$$

Since it is not possible to beforehand determine the number of threads specified by a program, $\mathscr{P}_{\mathrm{T}\in\mathbf{Thrd}}(\ldots)$ is defined to expand to $|\mathbf{Thrd}|$ instances (i.e., one instance for each thread, T, in a given program) of type $\{\mathrm{T}\} \times \mathbf{Lbl}_\mathrm{T} \times (\mathbf{Reg}_\mathrm{T} \to \mathbf{Val}) \times \mathbf{Time}$. Likewise, $[\ldots]_{\mathrm{T}\in\mathbf{Thrd}}$ is defined to expand in the corresponding manner. This way, $c \in \mathbf{Conf}$ can be regarded as a tuple with a known size when the number of threads in a program is known.

Sub-components of a configuration will also be of interest when considering single threads (see Table 4.2). Therefore, the following "smaller" configurations, $c^{ax}_{in} \in \mathbf{Conf}^{\mathbf{ax}}_{\mathbf{in}}$ and $c^{ax}_{out} \in \mathbf{Conf}^{\mathbf{ax}}_{\mathbf{out}}$, are defined for $\mathrm{T} \in \mathbf{Thrd}$.

$$\begin{aligned}\mathbf{Conf}^{\mathbf{ax}}_{\mathbf{in}} = {} & \{\mathrm{T}\} \times \mathbf{Lbl}_\mathrm{T} \times (\mathbf{Reg}_\mathrm{T} \to \mathbf{Val}) \times \\ & (\mathbf{Var} \to \mathbf{Thrd} \to \mathscr{P}(\mathbf{Val} \times \mathbf{Time})) \times \\ & (\mathbf{Lck} \to (\mathbf{Lck}_{\mathbf{stt}} \times \mathbf{Thrd}_\perp \times \mathbf{Time} \times \mathbf{Thrd}_\perp \times \mathbf{Time})) \\ & \times \mathbf{Time}\end{aligned}$$

$$c^{ax}_{in} ::= \langle \mathrm{T}, pc, \mathbb{r}, \mathbb{x}, \mathbb{l}, t\rangle$$

$$\begin{aligned}\mathbf{Conf}^{\mathbf{ax}}_{\mathbf{out}} = {} & \mathbf{Lbl}_\mathrm{T} \times (\mathbf{Reg}_\mathrm{T} \to \mathbf{Val}) \times \\ & (\mathbf{Var} \to \mathbf{Thrd} \to \mathscr{P}(\mathbf{Val} \times \mathbf{Time})) \times \\ & (\mathbf{Lck} \to (\mathbf{Lck}_{\mathbf{stt}} \times \mathbf{Thrd}_\perp \times \mathbf{Time} \times \mathbf{Thrd}_\perp \times \mathbf{Time}))\end{aligned}$$

$$c^{ax}_{out} ::= \langle pc, \mathbb{r}, \mathbb{x}, \mathbb{l}\rangle$$

4.2 Semantics

The semantic rules for individual statements within a thread, the language axioms, are described by the relation $\xrightarrow[ax]{} : \mathbf{Conf}^{\mathbf{ax}}_{\mathbf{in}} \times \mathbf{Conf}^{\mathbf{ax}}_{\mathbf{out}} \to \{\mathtt{true}, \mathtt{false}\}$, which is formally defined in Table 4.2. The semantic rule for a set of threads (i.e., the program) is described by the relation $\xrightarrow[prg]{} : \mathbf{Conf} \times \mathbf{Conf} \to \{\mathtt{true}, \mathtt{false}\}$, which is defined based on $\xrightarrow[ax]{}$ as given in Table 4.3. Note that the functions STM and STT, OWN, DL, POWN and REL are defined in Tables 4.4 and 4.5, respectively. Note that STM is a total function.

$\xrightarrow[ax]{}$ and $\xrightarrow[prg]{}$ are relations, not functions, because one single input configuration can have several outputs; e.g., if two or more threads execute `lock` *lck*, where $\mathrm{STT}(\mathbb{l}\ lck) = \mathit{unlocked}$, then *lck* is assigned to one of these threads by

$\xrightarrow[prg]{}$ in a non-deterministic fashion. $\xrightarrow[ax]{}$, $\xrightarrow[prg]{}$ and the semantic behavior of each statement in PPL are further described below.

To execute a program, or rather, to derive some possible execution trace for a given initial configuration, $c \in \mathbf{Conf}$, a succeeding configuration is given by any $c' \in \mathbf{Conf}$, such that $c \xrightarrow[prg]{} c'$. Then, a succeeding configuration, $c'' \in \mathbf{Conf}$, to c' is given by $c' \xrightarrow[prg]{} c''$, and so on.

STM(T, pc)	$\langle pc', \mathtt{r}', \mathtt{x}', \mathtt{l}' \rangle$	If
$[\mathtt{halt}]^{pc}$	$\langle pc, \mathtt{r}, \mathtt{x}, \mathtt{l} \rangle$	
$[\mathtt{skip}]^{pc}$	$\langle pc+1, \mathtt{r}, \mathtt{x}, \mathtt{l} \rangle$	
$[r := a]^{pc}$	$\langle pc+1, \mathtt{r}[r \mapsto \mathscr{A}[\![a]\!]\mathtt{r}], \mathtt{x}, \mathtt{l} \rangle$	
$[\mathtt{if}\ b\ \mathtt{goto}\ l]^{pc}$	$\langle pc+1, \mathtt{r}, \mathtt{x}, \mathtt{l} \rangle$	$\neg\mathscr{B}[\![b]\!]\mathtt{r}$
$[\mathtt{if}\ b\ \mathtt{goto}\ l]^{pc}$	$\langle l, \mathtt{r}, \mathtt{x}, \mathtt{l} \rangle$	$\mathscr{B}[\![b]\!]\mathtt{r}$
$[\mathtt{store}\ r\ \mathtt{to}\ x]^{pc}$	$\langle pc+1, \mathtt{r}, \mathtt{x}[x \mapsto (\mathtt{x}\ x)[\mathrm{T} \mapsto \{(\mathtt{r}\ r, t)\}]], \mathtt{l} \rangle$	
$[\mathtt{load}\ r\ \mathtt{from}\ x]^{pc}$	$\langle pc+1, \mathscr{R}(r, \mathtt{r}, x, \mathtt{x}), \mathtt{x}, \mathtt{l} \rangle$	
$[\mathtt{lock}\ lck]^{pc}$	$\langle pc+1, \mathtt{r}, \mathtt{x}, \mathtt{l}[lck \mapsto (locked, \mathrm{T}, \mathrm{DL}(\mathtt{l}\ lck), \mathrm{POWN}(\mathtt{l}\ lck), \mathrm{REL}(\mathtt{l}\ lck))] \rangle$	$\mathrm{OWN}(\mathtt{l}\ lck) = \mathrm{T}$
$[\mathtt{lock}\ lck]^{pc}$	$\langle pc, \mathtt{r}, \mathtt{x}, \mathtt{l} \rangle$	$\mathrm{OWN}(\mathtt{l}\ lck) \neq \mathrm{T}$
$[\mathtt{unlock}\ lck]^{pc}$	$\langle pc+1, \mathtt{r}, \mathtt{x}, \mathtt{l}[lck \mapsto (unlocked, \bot_{thrd}, \mathrm{DL}(\mathtt{l}\ lck), \mathrm{T}, t)] \rangle$	$\mathrm{OWN}(\mathtt{l}\ lck) = \mathrm{T}$
$[\mathtt{unlock}\ lck]^{pc}$	$\langle pc+1, \mathtt{r}, \mathtt{x}, \mathtt{l} \rangle$	$\mathrm{OWN}(\mathtt{l}\ lck) \neq \mathrm{T}$

where $\mathscr{R}(r, \mathtt{r}, x, \mathtt{x}) = \mathtt{r}[r \mapsto v]$ for some v, such that $\begin{cases} \exists t' \in \mathbf{Time} : (v, t') \in \bigcup_{\mathrm{T}' \in \mathbf{Thrd}} ((\mathtt{x}\ x)\ \mathrm{T}') & \textbf{if } \bigcup_{\mathrm{T}' \in \mathbf{Thrd}} ((\mathtt{x}\ x)\ \mathrm{T}') \neq \emptyset \\ v \in \gamma_{int}([-\infty, \infty]) & \textbf{otherwise} \end{cases}$

Table 4.2: $\langle \mathrm{T}, pc, \mathtt{r}, \mathtt{x}, \mathtt{l}, t \rangle \xrightarrow[ax]{} \langle pc', \mathtt{r}', \mathtt{x}', \mathtt{l}' \rangle$, the semantics of concrete axiom transitions.

$$\mathbf{Thrd}_{exe} \neq \emptyset \wedge \forall \mathrm{T} \in \mathbf{Thrd}_{exe} : \langle \mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, \mathbb{x}, \mathbb{l}'', t^{a\prime}_{\mathrm{T}} \rangle \underset{ax}{\longrightarrow} \langle pc'_{\mathrm{T}}, \mathbb{r}'_{\mathrm{T}}, \mathbb{x}'_{\mathrm{T}}, \mathbb{l}'_{\mathrm{T}} \rangle$$

$$c @ \langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^{a}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}}, \mathbb{x}, \mathbb{l} \rangle \underset{prg}{\longrightarrow} c' @ \langle [\mathrm{T}, (\mathrm{T} \in \mathbf{Thrd}_{exe} \ ? \ pc'_{\mathrm{T}} : pc_{\mathrm{T}}), (\mathrm{T} \in \mathbf{Thrd}_{exe} \ ? \ \mathbb{r}'_{\mathrm{T}} : \mathbb{r}_{\mathrm{T}}), t^{a\prime}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}}, \mathbb{x}', \mathbb{l}' \rangle$$

where

$$t = \min(\{t^{a}_{\mathrm{T}} + \textsc{time}(c, \mathrm{T}) \mid \mathrm{T} \in \mathbf{Thrd} \wedge \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) \neq [\texttt{halt}]^{pc_{\mathrm{T}}}\})$$

$$\mathbf{Thrd}_{exe} = \{\mathrm{T} \in \mathbf{Thrd} \mid t = t^{a}_{\mathrm{T}} + \textsc{time}(c, \mathrm{T}) \wedge \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) \neq [\texttt{halt}]^{pc_{\mathrm{T}}}\}$$

$$t^{a\prime}_{\mathrm{T}} = \begin{cases} t^{a}_{\mathrm{T}} + \textsc{time}(c, \mathrm{T}) & \textbf{if } \mathrm{T} \in \mathbf{Thrd}_{exe} \\ t^{a}_{\mathrm{T}} & \textbf{otherwise} \end{cases}$$

$$\mathbb{x}' \ x = \begin{cases} \mathbb{x} \ x & \textbf{if } \mathbf{Thrd}_{x} = \emptyset \\ \lambda \mathrm{T} \in \mathbf{Thrd}.(\mathrm{T} = \mathrm{T}' \ ? \ (\mathbb{x}'_{\mathrm{T}'} \ x) \ \mathrm{T}' : \emptyset) & \textbf{otherwise} \\ \quad \textbf{where } \mathrm{T}' \text{ is one of the threads in } \mathbf{Thrd}_{x} = \{\mathrm{T} \in \mathbf{Thrd}_{exe} \mid \exists r \in \mathbf{Reg}_{\mathrm{T}} : \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{store } r \texttt{ to } x]^{pc_{\mathrm{T}}}\} \end{cases}$$

$$\mathbb{l}'' \ lck = \begin{cases} (\mathit{unlocked}, \mathrm{T}', t, \textsc{pown}(\mathbb{l} \ lck), \textsc{rel}(\mathbb{l} \ lck)) & \textbf{for some } \mathrm{T}' \in \{\mathrm{T} \in \mathbf{Thrd}_{exe} \mid \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{lock } lck]^{pc_{\mathrm{T}}}\}, \\ & \textbf{if } \{\mathrm{T} \in \mathbf{Thrd}_{exe} \mid \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{lock } lck]^{pc_{\mathrm{T}}}\} \neq \emptyset \wedge \\ & \quad \textsc{own}(\mathbb{l} \ lck) = \bot_{thrd} \\ \mathbb{l} \ lck & \textbf{otherwise} \end{cases}$$

$$\mathbb{l}' \ lck = \begin{cases} \mathbb{l}'_{\textsc{own}(\mathbb{l}'' \ lck)} \ lck & \textbf{if } \textsc{own}(\mathbb{l}'' \ lck) \in \mathbf{Thrd}_{exe} \\ \mathbb{l} \ lck & \textbf{otherwise} \end{cases}$$

Table 4.3: $c \underset{prg}{\longrightarrow} c'$, the semantics of concrete program transitions.

$$\textsc{stm} : (\mathbf{Thrd} \times \mathbf{Lbl}) \to \mathbf{Stm} = ((\mathbf{ThrdID} \times \mathbf{Stm}) \times \mathbf{Lbl}) \to \mathbf{Stm}$$

$$\textsc{stm}((d,s),pc) = \begin{cases} s & \textbf{if } s = [\texttt{skip}]^{pc} \vee \\ & \quad s = [r := a]^{pc} \vee \\ & \quad s = [\texttt{if } b \texttt{ goto } l']^{pc} \vee \\ & \quad s = [\texttt{load } r \texttt{ from } x]^{pc} \vee \\ & \quad s = [\texttt{store } r \texttt{ to } x]^{pc} \vee \\ & \quad s = [\texttt{lock } lck]^{pc} \vee \\ & \quad s = [\texttt{unlock } lck]^{pc} \vee \\ & \quad s = [\texttt{halt}]^{pc} \\ \textsc{stm}((d,s'),pc) & \textbf{if } s = s'\texttt{;}\, s'' \wedge pc \in \textsc{labels}(s') \\ \textsc{stm}((d,s''),pc) & \textbf{if } s = s'\texttt{;}\, s'' \wedge pc \in \textsc{labels}(s'') \end{cases}$$

$$\textsc{labels} : \mathbf{Stm} \to \mathscr{P}(\mathbf{Lbl})$$

$$\textsc{labels}(s) = \begin{cases} \{l\} & \textbf{if } s = [\texttt{skip}]^{l} \vee \\ & \quad s = [r := a]^{l} \vee \\ & \quad s = [\texttt{if } b \texttt{ goto } l']^{l} \vee \\ & \quad s = [\texttt{load } r \texttt{ from } x]^{l} \vee \\ & \quad s = [\texttt{store } r \texttt{ to } x]^{l} \vee \\ & \quad s = [\texttt{lock } lck]^{l} \vee \\ & \quad s = [\texttt{unlock } lck]^{l} \vee \\ & \quad s = [\texttt{halt}]^{l} \\ \textsc{labels}(s') \cup \textsc{labels}(s'') & \textbf{if } s = s'\texttt{;}\, s'' \end{cases}$$

Table 4.4: Definition of STM and LABELS.

$$\textsc{stt} : (\mathbf{Lck_{stt}} \times \mathbf{Thrd}_\bot \times \mathbf{Time} \times \mathbf{Thrd}_\bot \times \mathbf{Time}) \to \mathbf{Lck_{stt}}$$
$$\textsc{stt}((u, \mathrm{T}, t, \mathrm{T}', t')) = u$$

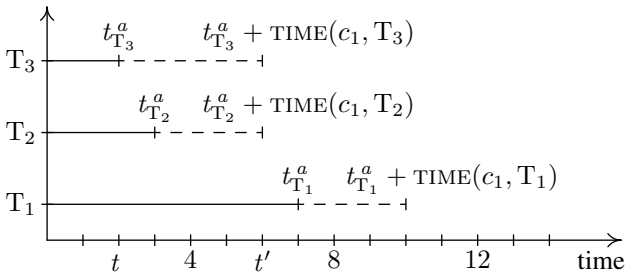
$$\textsc{own} : (\mathbf{Lck_{stt}} \times \mathbf{Thrd}_\bot \times \mathbf{Time} \times \mathbf{Thrd}_\bot \times \mathbf{Time}) \to \mathbf{Thrd}_\bot$$
$$\textsc{own}((u, \mathrm{T}, t, \mathrm{T}', t')) = \mathrm{T}$$

$$\textsc{dl} : (\mathbf{Lck_{stt}} \times \mathbf{Thrd}_\bot \times \mathbf{Time} \times \mathbf{Thrd}_\bot \times \mathbf{Time}) \to \mathbf{Time}$$
$$\textsc{dl}((u, \mathrm{T}, t, \mathrm{T}', t')) = t$$

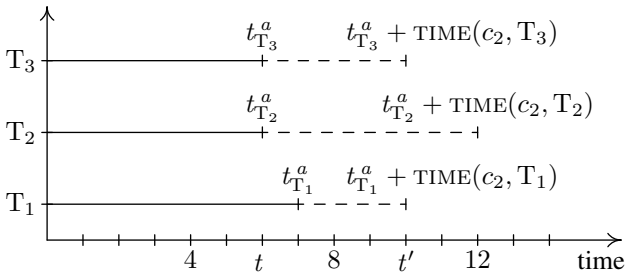
$$\textsc{pown} : (\mathbf{Lck_{stt}} \times \mathbf{Thrd}_\bot \times \mathbf{Time} \times \mathbf{Thrd}_\bot \times \mathbf{Time}) \to \mathbf{Thrd}_\bot$$
$$\textsc{pown}((u, \mathrm{T}, t, \mathrm{T}', t')) = \mathrm{T}'$$

$$\textsc{rel} : (\mathbf{Lck_{stt}} \times \mathbf{Thrd}_\bot \times \mathbf{Time} \times \mathbf{Thrd}_\bot \times \mathbf{Time}) \to \mathbf{Time}$$
$$\textsc{rel}((u, \mathrm{T}, t, \mathrm{T}', t')) = t'$$

Table 4.5: Definition of STT, OWN, DL, POWN and REL.



(a) c_1: $\mathbf{Thrd}_{exe} = \{\mathrm{T}_2, \mathrm{T}_3\}$



(b) c_2: $\mathbf{Thrd}_{exe} = \{\mathrm{T}_1, \mathrm{T}_3\}$

Figure 4.6: Illustration of how $\mathbf{Thrd}_{exe}$ is determined ($c_1 \xrightarrow[prg]{} c_2$).

TIME and $\mathbf{Thrd}_{exe}$

TIME is assumed to be provided by a timing model of the underlying architecture. It should return a relative execution time for the active statement of thread T, i.e., $\mathrm{STM}(\mathrm{T}, pc_{\mathrm{T}})$, based on the current system state. Given Assumption 4.1, time is guaranteed to move forward when using $\xrightarrow[prg]{}$ for a given configuration (Lemma 4.2). Assumption 4.3 gives that a thread that is waiting to acquire some lock cannot spin an infinite number of times in zero amount of time.

The set of threads to execute, $\mathbf{Thrd}_{exe}$, in a given configuration, $c@\langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}}, \mathbb{X}, \mathbb{l} \rangle \in \mathbf{Conf}$, is determined based on t^a_{T} and $\mathrm{TIME}(c, \mathrm{T})$, for each thread, $\mathrm{T} \in \mathbf{Thrd}$. It simply consists of the threads that will execute their active statements at the nearest point in time, denoted by t in Table 4.3. An illustration of how $\mathbf{Thrd}_{exe}$ is determined is given in Figure 4.6. For c_1 in Figure 4.6a, $t = t^a_{\mathrm{T}_2} + \mathrm{TIME}(c_1, \mathrm{T}_2) = t^a_{\mathrm{T}_3} + \mathrm{TIME}(c_1, \mathrm{T}_3) = 6$ and

$t^a_{\mathrm{T}_1} + \textsc{time}(c_1, \mathrm{T}_1) = 10$. Thus, $\mathbf{Thrd}_{exe} = \{\mathrm{T}_2, \mathrm{T}_3\}$ and $t^{a\prime}_{\mathrm{T}_2} = t^{a\prime}_{\mathrm{T}_3} = 6$, while $t^{a\prime}_{\mathrm{T}_1} = t^a_{\mathrm{T}_1} = 7$. For c_2 in Figure 4.6b, $\mathbf{Thrd}_{exe}$ is determined in a similar manner (note that $c_1 \xrightarrow[prg]{} c_2$).

Assumption 4.1 (TIME is non-negative):
It is assumed that $\forall c \in \mathbf{Conf} : \forall \mathrm{T} \in \mathbf{Thrd} : 0 \leq \textsc{time}(c, \mathrm{T})$. □

Lemma 4.2 (Time only moves forward):
Given that the two configurations $c@\langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle \in \mathbf{Conf}$ *and* $c'@\langle [\mathrm{T}, pc'_{\mathrm{T}}, \mathbb{r}'_{\mathrm{T}}, t^{a\prime}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}', \mathbb{l}'\rangle \in \mathbf{Conf}$ *are such that* $c \xrightarrow[prg]{} c'$, $\forall \mathrm{T} \in \mathbf{Thrd}$: $t^a_{\mathrm{T}} \leq t^{a\prime}_{\mathrm{T}}$. □

PROOF. Assume that $c@\langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle \in \mathbf{Conf}$ and $c'@\langle [\mathrm{T}, pc'_{\mathrm{T}}, \mathbb{r}'_{\mathrm{T}}, t^{a\prime}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}', \mathbb{l}'\rangle \in \mathbf{Conf}$ are such that $c \xrightarrow[prg]{} c'$. From Table 4.3, it is apparent that there are two possibilities for the value of $t^{a\prime}_{\mathrm{T}}$.

If $t^a_{\mathrm{T}} + \textsc{time}(c, \mathrm{T}) = \min(\{t^a_{\mathrm{T}'} + \textsc{time}(c, \mathrm{T}') \mid \mathrm{T}' \in \mathbf{Thrd}\}) \wedge \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) \neq [\texttt{halt}]^{pc_{\mathrm{T}}}$, then $t^{a\prime}_{\mathrm{T}} = t^a_{\mathrm{T}} + \textsc{time}(c, \mathrm{T})$. Thus, $t^{a\prime}_{\mathrm{T}} \geq t^a_{\mathrm{T}}$ (Assumption 4.1).

If $t^a_{\mathrm{T}} + \textsc{time}(c, \mathrm{T}) \neq \min(\{t^a_{\mathrm{T}'} + \textsc{time}(c, \mathrm{T}') \mid \mathrm{T}' \in \mathbf{Thrd}\}) \vee \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{halt}]^{pc_{\mathrm{T}}}$, then $t^{a\prime}_{\mathrm{T}} = t^a_{\mathrm{T}}$.

Thus, it must be that $\forall \mathrm{T} \in \mathbf{Thrd} : t^a_{\mathrm{T}} \leq t^{a\prime}_{\mathrm{T}}$. ■

Assumption 4.3 (TIME is non-zero when spin-locking):
It is assumed that $\forall c@\langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle \in \mathbf{Conf} : \forall \mathrm{T} \in \mathbf{Thrd} : ((\exists lck \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc_{\mathrm{T}}} \wedge \textsc{own}(\mathbb{l}\ lck) \notin \{\bot_{thrd}, \mathrm{T}\})) \Rightarrow 0 < \textsc{time}(c, \mathrm{T}))$. □

`halt` and `skip`

As previously discussed, `halt` stops the execution of a thread and `skip` is a no-operation. This is implemented by letting the semantic rule for `halt` return the input state without modifying it, which means that the issuing thread will still be executing the same `halt`-statement in the next iterative step; thus the thread halts. Note that threads issuing a `halt`-statement are not included in $\mathbf{Thrd}_{exe}$, however. The rule for the `skip`-statement only increments the thread's program counter, *pc*, and thus advances the thread to execute its subsequent statement in the next iterative step.

:= and $\mathscr{A}$

The statement $r := a$ returns a register state in which the register r has the value of the arithmetic expression a. The value of a is, in the general case,

$$\mathscr{A}[\![n]\!]\mathbb{r} = n \qquad \mathscr{A}[\![a_1 - a_2]\!]\mathbb{r} = \mathscr{A}[\![a_1]\!]\mathbb{r} - \mathscr{A}[\![a_2]\!]\mathbb{r}$$
$$\mathscr{A}[\![r]\!]\mathbb{r} = \mathbb{r}\ r \qquad \mathscr{A}[\![a_1 \star a_2]\!]\mathbb{r} = \mathscr{A}[\![a_1]\!]\mathbb{r} \cdot \mathscr{A}[\![a_2]\!]\mathbb{r}$$
$$\mathscr{A}[\![a_1 + a_2]\!]\mathbb{r} = \mathscr{A}[\![a_1]\!]\mathbb{r} + \mathscr{A}[\![a_2]\!]\mathbb{r} \qquad \mathscr{A}[\![a_1 \,/\, a_2]\!]\mathbb{r} = \left\lfloor \frac{\mathscr{A}[\![a_1]\!]\mathbb{r}}{\mathscr{A}[\![a_2]\!]\mathbb{r}} \right\rfloor$$

Table 4.7: Semantics of concrete evaluation of arithmetic expressions.

$$\mathscr{B}[\![\texttt{true}]\!]\mathbb{r} \Longleftrightarrow \texttt{true} \qquad \mathscr{B}[\![b_1 \,\&\&\, b_2]\!]\mathbb{r} \Longleftrightarrow \mathscr{B}[\![b_1]\!]\mathbb{r} \wedge \mathscr{B}[\![b_2]\!]\mathbb{r}$$
$$\mathscr{B}[\![\texttt{false}]\!]\mathbb{r} \Longleftrightarrow \texttt{false} \qquad \mathscr{B}[\![a_1 == a_2]\!]\mathbb{r} \Longleftrightarrow \mathscr{A}[\![a_1]\!]\mathbb{r} = \mathscr{A}[\![a_2]\!]\mathbb{r}$$
$$\mathscr{B}[\![\,!\,b]\!]\mathbb{r} \Longleftrightarrow \neg\mathscr{B}[\![b]\!]\mathbb{r} \qquad \mathscr{B}[\![a_1 <= a_2]\!]\mathbb{r} \Longleftrightarrow \mathscr{A}[\![a_1]\!]\mathbb{r} \leq \mathscr{A}[\![a_2]\!]\mathbb{r}$$

Table 4.8: Semantics of concrete evaluation of boolean expressions.

dependent on the register values in the input register state and is determined using the function $\mathscr{A} : \textbf{Aexp} \to (\textbf{Reg} \to \textbf{Val}) \to \textbf{Val}$. $\mathscr{A}$ evaluates arithmetic expressions based on a given register state as defined in Table 4.7.

`if` and $\mathscr{B}$

The statement `if` b `goto` l performs conditional branching. If the boolean expression b evaluates to `true`, the issuing thread's *pc* is set to l. If b evaluates to `false`, then `if` acts like the `skip`-statement. The value of b is, in the general case, dependent on the register values in the input register state and is determined using the function $\mathscr{B} : \textbf{Bexp} \to (\textbf{Reg} \to \textbf{Val}) \to \textbf{Bool}$. $\mathscr{B}$ evaluates boolean expressions based on a given register state as defined in Table 4.8.

`store` and `load`

To achieve a high precision in the analysis (see Chapters 5 and 6), the abstraction of the state for variables will need to save write history; i.e., what abstract

writes (each write being a pair of value and time) have been performed by each thread on each variable (see Chapter 5). Therefore, to derive a Galois connection between the concrete and abstract domains for variable states, the concrete state, $\mathbb{x}$, has to be defined accordingly. This is why the definition of $\mathbb{x}$ might look a bit peculiar at first glance. In the concrete semantics, only one single write is saved for each variable, though, since this is all the information that is needed in the concrete case. If several threads write to a variable (using the `store`-statement) at the same time, there is a race on that variable and the resulting state will contain one of the writes; i.e., one of the threads will win the race. The winning thread is non-deterministically chosen from one of the threads writing the variable at the given point in time.

`load` is defined to put the value of the saved write (or rather, the value of one of the saved writes in the general case) in the given register.

`lock` and `unlock`

As the observant reader might have noticed already, the only information that should be needed in order to successfully express the semantic behavior of locks is what thread is currently assigned (i.e., is currently the owner of) a lock. This is truly the case. However, the extra information given in the concrete state for locks, $\mathbb{l}$, will ease the deriving of an approximation of the concrete semantics (see Chapter 5) and achieve a high precision in the analysis (see Chapter 6). This is why the definition of $\mathbb{l}$ might look a bit peculiar at first glance. Here, the state of locks, i.e., *locked* or *unlocked*, is only used to increase the readability of the rules in Tables 4.2 and 4.3. Note that a consequence of this is that, in a given configuration, $c@\langle[\mathrm{T}, pc_\mathrm{T}, \mathbb{r}_\mathrm{T}, t^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle$, $\textsc{stt}(\mathbb{l}\ lck) = locked$ whenever $\textsc{own}(\mathbb{l}\ lck) \neq \bot_{thrd}$ (c.f., Lemma 4.5). Also note that $(\textsc{stt}(\mathbb{l}\ lck) = locked \wedge \textsc{own}(\mathbb{l}\ lck) = \bot_{thrd}) \vee \exists \mathrm{T} \in \mathbf{Thrd} : t^a_\mathrm{T} + \textsc{time}(c, \mathrm{T}) < \textsc{rel}(\mathbb{l}\ lck)$ implies that the given configuration is actually not valid (c.f., Definition 4.4).

The `lock`-statement has the same behavior as the `halt`-statement as long as the issuing thread is not assigned the given lock; i.e., the issuing thread will wait for its turn to acquire the lock. If the issuing thread is assigned the given lock (within $\xrightarrow[prg]{}$, using $\mathbb{l}''$), `lock` is defined to basically take the lock (i.e., set its state to *locked*) and advance the thread's *pc*. Only one single thread can be assigned a given lock at any point in time.

The `unlock`-statement has the same behavior as the `skip`-statement if the given lock is not assigned to the issuing thread. If the issuing thread is assigned the given lock, `unlock` is defined to release the lock so that it can be re-assigned in the next iterative step to some thread, if any, issuing `lock` on it.

Note that a thread can repeatedly acquire a lock that is assigned to, and taken by, it, without first releasing it.

Definition 4.4 (Valid concrete configuration):
A concrete configuration, $c@\langle[\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle \in \mathbf{Conf}$*, is valid with respect to the lock state,* $\mathbb{l}$*, iff*

$$\begin{aligned}\forall lck \in \mathbf{Lck} : ((&\mathrm{STT}(\mathbb{l}\ lck) = locked \Leftrightarrow \mathrm{OWN}(\mathbb{l}\ lck) \neq \bot_{thrd}) \wedge \\ &(\mathrm{STT}(\mathbb{l}\ lck) = unlocked \Leftrightarrow \mathrm{OWN}(\mathbb{l}\ lck) = \bot_{thrd}) \wedge \\ &\forall \mathrm{T} \in \mathbf{Thrd} : \mathrm{REL}(\mathbb{l}\ lck) \leq t^a_{\mathrm{T}} + \mathrm{TIME}(c, \mathrm{T}))\end{aligned}$$ □

Lemma 4.5 ($\xrightarrow[prg]{}$ preserves lock state validity):
Given that the configuration $c@\langle[\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle \in \mathbf{Conf}$ *is valid (c.f., Definition 4.4), then so is* $c'@\langle[\mathrm{T}, pc'_{\mathrm{T}}, \mathbb{r}'_{\mathrm{T}}, t^{a\prime}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}', \mathbb{l}'\rangle \in \mathbf{Conf}$*, whenever* $c \xrightarrow[prg]{} c'$. □

PROOF. From Table 4.2, it is apparent that the possible axiom output lock states, called $\mathbb{l}'_{\mathrm{T}}$ in Table 4.3, given an input lock state, called $\mathbb{l}''$ in Table 4.3, are

1. $\mathbb{l}''[lck \mapsto (locked, \mathrm{T}, \mathrm{DL}(\mathbb{l}''\ lck), \mathrm{POWN}(\mathbb{l}''\ lck), \mathrm{REL}(\mathbb{l}''\ lck)]$, whenever $\mathrm{STM}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc_{\mathrm{T}}} \wedge \mathrm{OWN}(\mathbb{l}''\ lck) = \mathrm{T}$,
2. $\mathbb{l}''[lck \mapsto (unlocked, \bot_{thrd}, \mathrm{DL}(\mathbb{l}''\ lck), \mathrm{T}, t^{a\prime}_{\mathrm{T}})]$, whenever $\mathrm{STM}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{unlock}\ lck]^{pc_{\mathrm{T}}} \wedge \mathrm{OWN}(\mathbb{l}''\ lck) = \mathrm{T}$, and
3. $\mathbb{l}''$, otherwise.

Assume that the configurations $c@\langle[\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle \in \mathbf{Conf}$ and $c'@\langle[\mathrm{T}, pc'_{\mathrm{T}}, \mathbb{r}'_{\mathrm{T}}, t^{a\prime}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}', \mathbb{l}'\rangle \in \mathbf{Conf}$ are such that c is valid and $c \xrightarrow[prg]{} c'$.

From Table 4.3, it is apparent that $\xrightarrow[ax]{}$ is only applied to axiom input configurations in which $\mathbb{l}''$ is such that

1. $\mathbb{l}''\ lck = \mathbb{l}\ lck$, or
2. $\mathbb{l}''\ lck = (unlocked, \mathrm{T}, t^{a\prime}_{\mathrm{T}}, \mathrm{POWN}(\mathbb{l}\ lck), \mathrm{REL}(\mathbb{l}\ lck))$.

For the first case, it is easy to see that all the three possible output lock states result in a valid configuration since c is valid.

The second case only occurs when $\exists \mathrm{T}' \in \mathbf{Thrd}_{exe} : (\mathrm{STM}(\mathrm{T}', pc_{\mathrm{T}'}) = [\texttt{lock}\ lck]^{pc_{\mathrm{T}'}} \wedge \mathrm{OWN}(\mathbb{l}\ lck) = \bot_{thrd})$. Note that the assigned owner,

$\mathrm{T} \in \mathbf{Thrd}_{exe}$, is one of the threads executing `lock` *lck*. For thread T, the output lock state is $\mathbb{l}''[lck \mapsto (locked, \mathrm{T}, \textsc{dl}(\mathbb{l}'' \; lck), \textsc{pown}(\mathbb{l}'' \; lck), \textsc{rel}(\mathbb{l}'' \; lck)]$, since $\textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc_{\mathrm{T}}} \wedge \textsc{own}(\mathbb{l}'' \; lck) = \mathrm{T}$. Hence, $\mathbb{l}' \; lck = \mathbb{l}[lck \mapsto (locked, \mathrm{T}, t^{a\prime}_{\mathrm{T}}, \textsc{pown}(\mathbb{l} \; lck), \textsc{rel}(\mathbb{l} \; lck)]$.

Since time moves forward for each thread (Lemma 4.2), it is easy to see that c' is valid. ■

Lemma 4.6 gives some important properties of the "intermediate" lock state, $\mathbb{l}''$, defined in Table 4.3, which is used as a means of assigning a lock to a specific thread. These properties will be used when proving the correctness of the abstract semantics in Tables 5.5 and 5.6.

Lemma 4.6 (Properties of $\mathbb{l}''$):
*If for some valid configuration, $c@\langle[\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^{a}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle \in$ **Conf**, and lock, $lck \in$ **Lck**, $\textsc{own}(\mathbb{l} \; lck) = \bot_{thrd} \wedge \exists \mathrm{T}' \in \{\mathrm{T} \in \mathbf{Thrd}_{exe} \mid \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc_{\mathrm{T}}}\} : \textsc{own}(\mathbb{l}'' \; lck) = \mathrm{T}'$, where $\mathbb{l}''$ and $\mathbf{Thrd}_{exe}$ are as defined in Table 4.3, then*

1. $\textsc{stt}(\mathbb{l}'' \; lck) = unlocked$,
2. $\textsc{dl}(\mathbb{l}'' \; lck) \not< t^{a\prime}_{\mathrm{T}'}$, and
3. $t^{a\prime}_{\mathrm{T}'} \not< \textsc{rel}(\mathbb{l}'' \; lck)$. □

PROOF. For this proof, each of the properties above will be shown based on the definition of $\xrightarrow[ax]{}$ and $\xrightarrow[prg]{}$, defined in Tables 4.2 and 4.3, respectively.

Assume that for the valid configuration $c@\langle[\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^{a}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle \in$ **Conf** (c.f., Definition 4.4) and some lock, $lck \in$ **Lck**, $\textsc{own}(\mathbb{l} \; lck) = \bot_{thrd} \wedge \exists \mathrm{T}' \in \{\mathrm{T} \in \mathbf{Thrd}_{exe} \mid \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc_{\mathrm{T}}}\} : \textsc{own}(\mathbb{l}'' \; lck) = \mathrm{T}'$.

1 follows directly from the definition of $\mathbb{l}''$ in Table 4.3.

Table 4.3 also gives that $\textsc{dl}(\mathbb{l}'' \; lck) = t$ and that $t^{a\prime}_{\mathrm{T}'} = t$, since $\mathrm{T}' \in \mathbf{Thrd}_{exe} \wedge \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc_{\mathrm{T}}}$. Thus, $\textsc{dl}(\mathbb{l}'' \; lck) = t^{a\prime}_{\mathrm{T}'}$, and hence, 2 has been shown.

For 3, Assumption 4.1 gives that time moves forward when using $\xrightarrow[prg]{}$ (Lemma 4.2). Thus, it must be that $t^{a\prime}_{\mathrm{T}'} = t \geq \textsc{rel}(\mathbb{l} \; lck) = \textsc{rel}(\mathbb{l}'' \; lck)$ (c.f., Table 4.3), which concludes the proof. ■

4.3 Collecting Semantics

This section defines the collecting semantics, $\mathscr{C}(C)$, of a program; i.e., the set of all possible semantic configurations given an initial set of configurations, C, (c.f., Definition 4.7).

Definition 4.7 (Collecting semantics):
The collecting semantics, $\mathscr{C}(C)$, of an initial set of configurations, C, is defined as:

$$\mathscr{C}(C) = \bigcup_{i \geq 0} C^i \qquad \textbf{where} \begin{cases} C^0 & = C \\ C^{i+1} & = \{c' \in \textbf{Conf} \mid \exists c \in C^i : c \xrightarrow[prg]{} c'\} \end{cases}$$

□

As can be seen, the collecting semantics will include all possible configurations that a given initial configuration can ever reach. Note that the collecting semantics might be of infinite size in the case of a non-terminating program; i.e., the accumulated time, t^a_{T}, for some thread, $\mathrm{T} \in \textbf{Thrd}$, could increase indefinitely.

Chapter 5

Abstractly Interpreting PPL

In this chapter, the semantics of PPL, defined in Chapter 4, will be abstracted. First it must be decided what parts of the system state to interpret in an abstract way. Abstract states will be crowned with '˜'.

To allow for the hardware timing model to be approximated as well, **Time** will be abstracted using the interval domain, i.e., $\widetilde{\mathbf{Time}} = \mathbf{Intv}$. This approach is also taken by Chattopadhyay et al. [11] to approximate the execution time of pipeline stages in order to deal with timing anomalies in multi-core platforms. **Val** will also be abstracted using intervals, i.e., $\widetilde{\mathbf{Val}} = \mathbf{Intv}$, to allow for an efficient handling of data flow (note that many other domains could be used as well). Since **Thrd**, **Lbl**, **Var**, **Reg**, **Lck**, **Aexp** and **Bexp** are defined by the software, and the elements of them are used as identifiers, it does not make much sense to abstract them for the defined analysis (see Chapter 6). And, since $\mathbf{Lck_{stt}}$ is comparable to **Bool**, an abstraction of it would most probably not be very beneficial. The states affected by the abstractions of **Time** and **Val** are $\mathbb{r}$, $\mathbb{x}$, t^a, $\mathbb{l}$ and c. The abstraction of these will be referred to as $\tilde{\mathbb{r}}$, $\tilde{\mathbb{x}}$, $\tilde{t}^a$, $\tilde{\mathbb{l}}$ and $\tilde{c}$, respectively.

NOTE. *A summary of the notation and nomenclature used in this thesis can be found in Appendix A.*

5.1 Arithmetical Operators for Intervals

Since values and time are abstracted using the interval domain, the operators of PPL must be extended to act on intervals. This is done in Table 5.1; note that ∞/∞, $0/0$, $0*\infty$ and $\infty-\infty$ need not be defined – they all result in $[-\infty,\infty]$.

NOTE. *In the following, $\tilde{+}_t$ and $\tilde{+}_{val}$ both refer to $+_{int}$, and similarly for the rest of the operators.*

5.2 Abstract Register States

Using Theorems 3.24 and 3.39, it is easy to see that there is indeed a Galois connection, $\langle \alpha_{reg}, \gamma_{reg} \rangle$, between the concrete domain $\mathscr{P}(\mathbf{Reg} \to \mathbf{Val})$ and the abstract domain $(\mathbf{Reg} \to \tilde{\mathbf{Val}}) \cup \{\tilde{\bot}_{reg}, \tilde{\top}_{reg}\}$ (Theorem 5.6). The concretization function, γ_{reg}, partial order, $\tilde{\sqsubseteq}_{reg}$, greatest lower bound, $\tilde{\bigsqcap}_{reg}$, least upper bound, $\tilde{\bigsqcup}_{reg}$, and abstraction function, α_{reg}, are given by Definitions 5.1, 5.2, 5.3, 5.4 and 5.5, respectively. $\tilde{\mathtt{r}}$ is the bottom element, $\tilde{\bot}_{reg}$, if $\exists r \in \mathbf{Reg} : \tilde{\mathtt{r}}\ r = \tilde{\bot}_{val}$; i.e., if $\tilde{\mathtt{r}}$ maps some register to $\tilde{\bot}_{val}$. The top element, $\tilde{\top}_{reg}$, corresponds to an abstract mapping for which all registers map to $\tilde{\top}_{val}$.

Definition 5.1 (Concretization of an abstract register state):

$$\gamma_{reg}(\tilde{\mathtt{r}}) = \begin{cases} \mathbf{Reg} \to \mathbf{Val} & \textbf{if } \tilde{\mathtt{r}} = \tilde{\top}_{reg} \\ \emptyset & \textbf{if } \tilde{\mathtt{r}} = \tilde{\bot}_{reg} \\ \{\lambda r \in \mathbf{Reg}.v \mid v \in \gamma_{val}(\tilde{\mathtt{r}}\ r)\} & \textbf{otherwise} \end{cases}$$ □

Definition 5.2 (Partial order for abstract register states):

$$\begin{cases} \tilde{\mathtt{r}} \ \tilde{\sqsubseteq}_{reg}\ \tilde{\top}_{reg} \\ \tilde{\bot}_{reg}\ \tilde{\sqsubseteq}_{reg}\ \tilde{\mathtt{r}} \\ \tilde{\mathtt{r}} \ \tilde{\sqsubseteq}_{reg}\ \tilde{\mathtt{r}}' \iff \forall r \in \mathbf{Reg} : \tilde{\mathtt{r}}\ r \ \tilde{\sqsubseteq}_{val}\ \tilde{\mathtt{r}}'\ r \end{cases}$$ □

$$\top_{int}\ op_{int}\ i = i\ op_{int}\ \top_{int} = \top_{int} \qquad \textbf{where}\ op_{int} \in \{+_{int}, -_{int}, *_{int}, /_{int}\}$$
$$\bot_{int}\ op_{int}\ i = i\ op_{int}\ \bot_{int} = \bot_{int}$$

$$[n_1,n_2] +_{int} [n_1',n_2'] = \begin{cases} [n_1+n_1', n_2+n_2'] & \textbf{if } -\infty < n_1,n_1',n_2,n_2' < \infty \\ [n_1+n_1', \infty] & \textbf{if } (n_2 = \infty \vee n_2' = \infty) \wedge \\ & \quad \neg((n_1 = -\infty \wedge n_1' = \infty) \vee \\ & \quad\quad (n_1 = \infty \wedge n_1' = -\infty)) \\ [-\infty, n_2+n_2'] & \textbf{if } (n_1 = -\infty \vee n_1' = -\infty) \wedge \\ & \quad \neg((n_2 = -\infty \wedge n_2' = \infty) \vee \\ & \quad\quad (n_2 = \infty \wedge n_2' = -\infty)) \\ [-\infty, \infty] & \textbf{otherwise} \end{cases}$$

$$[n_1,n_2] -_{int} [n_1',n_2'] = \begin{cases} [n_1-n_2', n_2-n_1'] & \textbf{if } -\infty < n_1,n_1',n_2,n_2' < \infty \\ [n_1-n_2', \infty] & \textbf{if } (n_2 = \infty \vee n_1' = -\infty) \wedge \\ & \quad \neg((n_1 = \infty \wedge n_2' = \infty) \vee \\ & \quad\quad (n_1 = -\infty \wedge n_2' = -\infty)) \\ [-\infty, n_2-n_1'] & \textbf{if } (n_1 = -\infty \vee n_2' = \infty) \wedge \\ & \quad \neg((n_2 = \infty \wedge n_1' = \infty) \vee \\ & \quad\quad (n_2 = -\infty \wedge n_1' = -\infty)) \\ [-\infty, \infty] & \textbf{otherwise} \end{cases}$$

$$[n_1,n_2] *_{int} [n_1',n_2'] = \begin{cases} [\min(V), \max(V)] & \textbf{if } (n_2 < 0 \wedge n_1' > 0) \vee \\ & \quad (n_2 < 0 \wedge n_2' < 0) \vee \\ & \quad (n_1 > 0 \wedge n_1' > 0) \vee \\ & \quad (n_1 > 0 \wedge n_2' < 0) \vee \\ & \quad (-\infty < n_1,n_1',n_2,n_2' < \infty) \\ & \textbf{where } V = \{n_1*n_1', n_1*n_2', n_2*n_1', n_2*n_2'\} \\ [-\infty, \infty] & \textbf{otherwise} \end{cases}$$

$$[n_1,n_2] /_{int} [n_1',n_2'] = \begin{cases} [\lfloor \min(V) \rfloor, \lceil \max(V) \rceil] & \textbf{if } (-\infty < n_1' \wedge n_2' < 0) \vee \\ & \quad (0 < n_1' \wedge n_2' < \infty) \\ & \textbf{where } V = \{n_1/n_1', n_1/n_2', n_2/n_1', n_2/n_2'\} \\ [-\infty, -n_1] & \textbf{if } n_1' \leq 0 \leq n_2' \wedge n_2 < 0 \\ [-n_2, \infty] & \textbf{if } n_1' \leq 0 \leq n_2' \wedge 0 < n_1 \\ [-\infty, \infty] & \textbf{otherwise} \end{cases}$$

Table 5.1: PPL operators defined for interval arguments.

Definition 5.3 (Greatest lower bound of abstract register states):

$$\begin{cases} \tilde{\top}_{reg} \tilde{\sqcap}_{reg} \tilde{\mathbb{r}} = \tilde{\mathbb{r}} \tilde{\sqcap}_{reg} \tilde{\top}_{reg} = \tilde{\mathbb{r}} \\ \tilde{\bot}_{reg} \tilde{\sqcap}_{reg} \tilde{\mathbb{r}} = \tilde{\mathbb{r}} \tilde{\sqcap}_{reg} \tilde{\bot}_{reg} = \tilde{\bot}_{reg} \\ (\tilde{\mathbb{r}} \tilde{\sqcap}_{reg} \tilde{\mathbb{r}}')\ r = (\tilde{\mathbb{r}}\ r) \tilde{\sqcap}_{val} (\tilde{\mathbb{r}}'\ r) \end{cases}$$ □

Definition 5.4 (Least upper bound of abstract register states):

$$\begin{cases} \tilde{\top}_{reg} \tilde{\sqcup}_{reg} \tilde{\mathbb{r}} = \tilde{\mathbb{r}} \tilde{\sqcup}_{reg} \tilde{\top}_{reg} = \tilde{\top}_{reg} \\ \tilde{\bot}_{reg} \tilde{\sqcup}_{reg} \tilde{\mathbb{r}} = \tilde{\mathbb{r}} \tilde{\sqcup}_{reg} \tilde{\bot}_{reg} = \tilde{\mathbb{r}} \\ (\tilde{\mathbb{r}} \tilde{\sqcup}_{reg} \tilde{\mathbb{r}}')\ r = (\tilde{\mathbb{r}}\ r) \tilde{\sqcup}_{val} (\tilde{\mathbb{r}}'\ r) \end{cases}$$ □

Definition 5.5 (Abstraction of a set of register states):

$$\alpha_{reg}(\mathbb{R}) = \begin{cases} \tilde{\top}_{reg} & \textbf{if } \mathbb{R} = \textbf{Reg} \to \textbf{Val} \\ \tilde{\bot}_{reg} & \textbf{if } \mathbb{R} = \emptyset \\ \lambda r \in \textbf{Reg}.\alpha_{val}(\{\mathbb{r}\ r \mid \mathbb{r} \in \mathbb{R}\}) & \textbf{otherwise} \end{cases}$$ □

Theorem 5.6 (Galois connection – Register states):
$\langle \alpha_{reg}, \gamma_{reg} \rangle$, where γ_{reg} and α_{reg} are defined as in Definitions 5.1 and 5.5, respectively, is a Galois connection. □

PROOF. Since $\alpha_{val} = \alpha_{int}$ and $\gamma_{val} = \gamma_{int}$, $\langle \alpha_{val}, \gamma_{val} \rangle$ is a Galois insertion between $\mathscr{P}(\textbf{Val})$ and $\tilde{\textbf{Val}}$ (Theorem 3.39).

By Theorem 3.24, $\langle \alpha_{reg} : \mathscr{P}(\textbf{Reg} \to \textbf{Val}) \to ((\textbf{Reg} \to \tilde{\textbf{Val}}) \cup \{\tilde{\bot}_{reg}, \tilde{\top}_{reg}\}), \gamma_{reg} : ((\textbf{Reg} \to \tilde{\textbf{Val}}) \cup \{\tilde{\bot}_{reg}, \tilde{\top}_{reg}\}) \to \mathscr{P}(\textbf{Reg} \to \textbf{Val}) \rangle$, where γ_{reg} and α_{reg} are as presented in Definitions 5.1 and 5.5, respectively, is a Galois connection. ■

5.3 Abstract Evaluation of Arithmetical Expressions

The function evaluating arithmetic expressions, $\mathscr{A}$, must be abstracted since values and register states are abstracted. The abstraction will be $\tilde{\mathscr{A}} : \textbf{Aexp} \to (\textbf{Reg} \to \tilde{\textbf{Val}}) \to \tilde{\textbf{Val}}$, which is equivalent to $\tilde{\mathscr{A}} : \textbf{Aexp} \to (\textbf{Reg} \to \textbf{Intv}) \to$

$$\tilde{\mathscr{A}}[\![n]\!]\tilde{\mathtt{r}} = \alpha_{val}(\{n\}) \qquad \tilde{\mathscr{A}}[\![a_1 + a_2]\!]\tilde{\mathtt{r}} = \tilde{\mathscr{A}}[\![a_1]\!]\tilde{\mathtt{r}} \mathbin{\tilde{+}_{val}} \tilde{\mathscr{A}}[\![a_2]\!]\tilde{\mathtt{r}}$$

$$\tilde{\mathscr{A}}[\![r]\!]\tilde{\mathtt{r}} = \tilde{\mathtt{r}}\ r \qquad \tilde{\mathscr{A}}[\![a_1 - a_2]\!]\tilde{\mathtt{r}} = \tilde{\mathscr{A}}[\![a_1]\!]\tilde{\mathtt{r}} \mathbin{\tilde{-}_{val}} \tilde{\mathscr{A}}[\![a_2]\!]\tilde{\mathtt{r}}$$

$$\tilde{\mathscr{A}}[\![a_1 \star a_2]\!]\tilde{\mathtt{r}} = \tilde{\mathscr{A}}[\![a_1]\!]\tilde{\mathtt{r}} \mathbin{\tilde{\star}_{val}} \tilde{\mathscr{A}}[\![a_2]\!]\tilde{\mathtt{r}}$$

$$\tilde{\mathscr{A}}[\![a_1 \,/\, a_2]\!]\tilde{\mathtt{r}} = \tilde{\mathscr{A}}[\![a_1]\!]\tilde{\mathtt{r}} \mathbin{\tilde{/}_{val}} \tilde{\mathscr{A}}[\![a_2]\!]\tilde{\mathtt{r}}$$

Table 5.2: The abstract function evaluating arithmetic expressions.

Intv, and can be derived using Definition 3.11 to induce $\mathscr{A}$. To do this, $\mathscr{A}$ must first be lifted to sets of concrete register mappings:

$$\mathscr{A}_{\mathscr{P}}[\![a]\!]\mathbb{R} = \{\mathscr{A}[\![a]\!]\mathtt{r} \mid \mathtt{r} \in \mathbb{R}\}$$

The abstract evaluation function can then be derived as:

$$\tilde{\mathscr{A}}[\![a]\!] = \alpha_{val} \circ \mathscr{A}_{\mathscr{P}}[\![a]\!] \circ \gamma_{reg}$$

The details of this function can be found in Table 5.2.

5.4 Boolean Restriction

The function $\tilde{\mathscr{B}}_{\mathscr{R}}$, defined in Definition 5.7, will be used in the abstract axiom transition rules (see Table 5.5 on page 84). This function is safely induced from $\mathscr{B}$, using Definition 3.11, so that the concretization of $\tilde{\mathscr{B}}_{\mathscr{R}}[\![b]\!]\tilde{\mathtt{r}}$, where $b \in$ **Bexp**, always contains (at least) the concrete register states, derived from $\tilde{\mathtt{r}}$, for which b evaluates to `true`.

Definition 5.7 (Boolean restriction):

$$\tilde{\mathscr{B}}_{\mathscr{R}}[\![b]\!]\tilde{\mathtt{r}} = \alpha_{reg}(\{\mathtt{r} \in \gamma_{reg}(\tilde{\mathtt{r}}) \mid \mathscr{B}[\![b]\!]\mathtt{r}\}) \qquad \square$$

5.5 Abstract Variable States

Using Theorems 3.17, 3.20, 3.24 and 3.39, a Galois connection, $\langle \alpha_{var}, \gamma_{var} \rangle$, between the concrete domain $\mathscr{P}(\mathbf{Var} \to \mathbf{Thrd} \to \mathscr{P}(\mathbf{Val} \times \mathbf{Time}))$ and the abstract domain $(\mathbf{Var} \to \mathbf{Thrd} \to \mathscr{P}(\tilde{\mathbf{Val}} \times \tilde{\mathbf{Time}})) \cup \{\tilde{\bot}_{var}, \tilde{\top}_{var}\}$ can be established. The concretization function, γ_{var}, abstraction function, α_{var}, partial order, $\tilde{\sqsubseteq}_{var}$, greatest lower bound, $\tilde{\bigsqcap}_{var}$, and least upper bound, $\tilde{\bigsqcup}_{var}$, are given by Definitions 5.8, 5.9, 5.14, 5.15 and 5.16, respectively.

$\tilde{\mathbb{x}}$ is the bottom element, $\tilde{\bot}_{var}$, if $\exists x \in \mathbf{Var} : \exists \mathrm{T} \in \mathbf{Thrd} : ((\tilde{\mathbb{x}}\ x)\ \mathrm{T}) = \emptyset$; i.e., some variable and thread combination maps to the empty set (there is no write-history available for that combination). Note that such an abstract variable state has no concrete counterparts ($\gamma_{var}(\tilde{\bot}_{var}) = \emptyset$). Therefore, an abstract variable state, $\tilde{\mathbb{x}}$, that actually contains no history for thread T on variable x, should have $((\tilde{\mathbb{x}}\ x)\ \mathrm{T}) = \{(\tilde{\bot}_{val}, \tilde{\bot}_t)\}$ to make $\tilde{\mathbb{x}} \neq \tilde{\bot}_{var}$. Note that $\gamma_{var}(\tilde{\mathbb{x}})$, where $((\tilde{\mathbb{x}}\ x)\ \mathrm{T}) = \{(\tilde{\bot}_{val}, \tilde{\bot}_t)\}$ for some variable, x, and thread, T, is a set of concrete states, $\mathbb{X}$, for which all $\mathbb{x} \in \mathbb{X}$ are such that $((\mathbb{x}\ x)\ \mathrm{T}) = \emptyset$.

The top element, $\tilde{\top}_{var}$, corresponds to a state where all variable and thread combinations are mapped to $\tilde{\mathbf{Val}} \times \tilde{\mathbf{Time}}$.

Definition 5.8 (Concretization of an abstract variable state):

$$
\begin{cases}
\gamma_{var}(\tilde{\top}_{var}) = \mathbf{Var} \to \mathbf{Thrd} \to (\mathbf{Val} \times \mathbf{Time}) \\
\gamma_{var}(\tilde{\bot}_{var}) = \emptyset \\
\gamma_{var}(\tilde{\mathbb{x}}) = \{\lambda x \in \mathbf{Var}.f \mid f \in \{\lambda \mathrm{T} \in \mathbf{Thrd}.W \mid W \in \\
\qquad \{W' \mid (\alpha_{val}(\{v \in \mathbf{Val} \mid \exists t \in \mathbf{Time} : (v,t) \in W'\}), \\
\qquad\qquad \alpha_t(\{t \in \mathbf{Time} \mid \exists v \in \mathbf{Val} : (v,t) \in W'\})) \in \\
\qquad\qquad\qquad ((\tilde{\mathbb{x}}\ x)\ \mathrm{T})\}\}\}
\end{cases}
$$

□

Definition 5.9 (Abstraction of a set of variable states):

$$
\begin{cases}
\alpha_{var}(\mathbf{Var} \to \mathbf{Thrd} \to (\mathbf{Val} \times \mathbf{Time})) = \tilde{\top}_{var} \\
\alpha_{var}(\emptyset) = \tilde{\bot}_{var} \\
\alpha_{var}(\mathbb{X}) = \lambda x \in \mathbf{Var}.\lambda \mathrm{T} \in \mathbf{Thrd}.\{(\alpha_{val}(\{v \mid \exists t \in \mathbf{Time} : (v,t) \in W\}), \\
\qquad\qquad\qquad \alpha_t(\{t \mid \exists v \in \mathbf{Val} : (v,t) \in W\})) \mid \\
\qquad\qquad\qquad\qquad W \in \{(\mathbb{x}\ x)\ \mathrm{T} \mid \mathbb{x} \in \mathbb{X}\}\}
\end{cases}
$$

□

Theorem 5.10 (Galois connection – Variable states):
$\langle \alpha_{var}, \gamma_{var} \rangle$, where γ_{var} and α_{var} are defined as in Definitions 5.8 and 5.9, respectively, defines a Galois connection.

PROOF. Since $\langle \alpha_{int}, \gamma_{int} \rangle$ is a Galois insertion (Theorem 3.39) and thus a Galois connection, so are $\langle \alpha_{val}, \gamma_{val} \rangle$ and $\langle \alpha_t, \gamma_t \rangle$ (since $\alpha_{val} = \alpha_t = \alpha_{int}$ and $\gamma_{val} = \gamma_t = \gamma_{int}$). Using Theorems 3.17, 3.20 and 3.24 to derive α_{var} and γ_{var}, the result follows (note that the cases $\gamma_{var}(\tilde{\top}_{var})$, $\gamma_{var}(\tilde{\bot}_{var})$, $\alpha_{var}(\mathbf{Var} \to \mathbf{Thrd} \to (\mathbf{Val} \times \mathbf{Time}))$ and $\alpha_{var}(\emptyset)$ follow trivially):

$$
\begin{aligned}
\gamma_{var}(\tilde{\mathbb{x}}) &\overset{Th.\ 3.24}{=} \{\lambda x \in \mathbf{Var}.f \mid f \in \gamma'(\tilde{\mathbb{x}}\ x)\} \\
&\overset{Th.\ 3.24}{=} \{\lambda x \in \mathbf{Var}.f \mid f \in \{\lambda \mathrm{T} \in \mathbf{Thrd}.W \mid W \in \gamma''((\tilde{\mathbb{x}}\ x)\ \mathrm{T})\}\} \\
&\overset{Th.\ 3.20}{=} \{\lambda x \in \mathbf{Var}.f \mid f \in \{\lambda \mathrm{T} \in \mathbf{Thrd}.W \mid W \in \\
&\qquad\qquad \{W' \mid \alpha'''(W') \in ((\tilde{\mathbb{x}}\ x)\ \mathrm{T})\}\}\} \\
&\overset{Th.\ 3.17}{=} \{\lambda x \in \mathbf{Var}.f \mid f \in \{\lambda \mathrm{T} \in \mathbf{Thrd}.W \mid W \in \\
&\qquad\qquad \{W' \mid (\alpha_{val}(\{v \in \mathbf{Val} \mid \exists t \in \mathbf{Time} : (v,t) \in W'\}), \\
&\qquad\qquad\qquad \alpha_t(\{t \in \mathbf{Time} \mid \exists v \in \mathbf{Val} : (v,t) \in W'\})) \in \\
&\qquad\qquad\qquad\qquad ((\tilde{\mathbb{x}}\ x)\ \mathrm{T})\}\}\}
\end{aligned}
$$

$$
\begin{aligned}
\alpha_{var}(\mathbb{X}) &\overset{Th.\ 3.24}{=} \lambda x \in \mathbf{Var}.\alpha'(\{\mathbb{x}\ x \mid \mathbb{x} \in \mathbb{X}\}) \\
&\overset{Th.\ 3.24}{=} \lambda x \in \mathbf{Var}.\lambda \mathrm{T} \in \mathbf{Thrd}.\alpha''(\{f\ \mathrm{T} \mid f \in \{\mathbb{x}\ x \mid \mathbb{x} \in \mathbb{X}\}\}) \\
&\overset{calc.}{=} \lambda x \in \mathbf{Var}.\lambda \mathrm{T} \in \mathbf{Thrd}.\alpha''(\{(\mathbb{x}\ x)\ \mathrm{T} \mid \mathbb{x} \in \mathbb{X}\}) \\
&\overset{Th.\ 3.20}{=} \lambda x \in \mathbf{Var}.\lambda \mathrm{T} \in \mathbf{Thrd}.\{\alpha'''(W) \mid W \in \{(\mathbb{x}\ x)\ \mathrm{T} \mid \mathbb{x} \in \mathbb{X}\}\} \\
&\overset{Th.\ 3.17}{=} \lambda x \in \mathbf{Var}.\lambda \mathrm{T} \in \mathbf{Thrd}.\{(\alpha_{val}(\{v \mid \exists t \in \mathbf{Time} : (v,t) \in W\}), \\
&\qquad\qquad\qquad \alpha_t(\{t \mid \exists v \in \mathbf{Val} : (v,t) \in W\})) \mid \\
&\qquad\qquad\qquad\qquad W \in \{(\mathbb{x}\ x)\ \mathrm{T} \mid \mathbb{x} \in \mathbb{X}\}\}
\end{aligned}
$$

■

The state $\tilde{\mathbb{x}} \in (\mathbf{Var} \to \mathbf{Thrd} \to \mathscr{P}(\tilde{\mathbf{Val}} \times \tilde{\mathbf{Time}})) \cup \{\tilde{\bot}_{var}, \tilde{\top}_{var}\}$ can save any number (i.e., history) of abstract writes, $\tilde{w} \in \tilde{\mathbf{Val}} \times \tilde{\mathbf{Time}}$, for each thread that occur on some variable. This is done to counteract the precision loss due to approximating points in time with intervals. The information available in such history makes it possible to use sequence information (within each thread) and timing information (between threads) to get a reasonably tight value when reading a variable.

For convenience in expressing, and increased readability of, the upcoming algorithms, some relations for abstract writes, $\tilde{w} ::= (\tilde{v}, \tilde{t})$, will be defined. The

partial order, $\tilde{\sqsubseteq}_w$, and least upper bound operator, $\tilde{\bigsqcup}_w$, for writes follow naturally (c.f., Definitions 3.26 and 3.28) from the partial orders and least upper bound operators for values, $\tilde{\sqsubseteq}_{val}$ and $\tilde{\bigsqcup}_{val}$, and time, $\tilde{\sqsubseteq}_t$ and $\tilde{\bigsqcup}_t$. $\tilde{\sqsubseteq}_w$ and $\tilde{\bigsqcup}_w$ are given by Definitions 5.11 and 5.12, respectively.

NOTE. *The notations "$\tilde{w}$" and "$(\tilde{v},\tilde{t})$" for abstract writes will from here on be used interchangeably.*

Definition 5.11 (Partial order of writes, $\tilde{\sqsubseteq}_w$):

$$\begin{cases} \tilde{w} \,\tilde{\sqsubseteq}_w\, \tilde{\top}_w \\ \tilde{\bot}_w \,\tilde{\sqsubseteq}_w\, \tilde{w} \\ (\tilde{v}_1,\tilde{t}_1) \,\tilde{\sqsubseteq}_w\, (\tilde{v}_2,\tilde{t}_2) \Longleftrightarrow \tilde{v}_1 \,\tilde{\sqsubseteq}_{val}\, \tilde{v}_2 \wedge \tilde{t}_1 \,\tilde{\sqsubseteq}_t\, \tilde{t}_2 \end{cases} \qquad \square$$

Definition 5.12 (Least upper bound of writes, $\tilde{\bigsqcup}_w$):

$$\begin{cases} \tilde{w} \,\tilde{\sqcup}_w\, \tilde{\top}_w = \tilde{\top}_w \,\tilde{\sqcup}_w\, \tilde{w} = \tilde{\top}_w \\ \tilde{\bot}_w \,\tilde{\sqcup}_w\, \tilde{w} = \tilde{w} \,\tilde{\sqcup}_w\, \tilde{\bot}_w = \tilde{w} \\ (\tilde{v}_1,\tilde{t}_1) \,\tilde{\sqcup}_w (\tilde{v}_2,\tilde{t}_2) = (\tilde{v}_1 \,\tilde{\sqcup}_{val}\, \tilde{v}_2, \tilde{t}_1 \,\tilde{\sqcup}_t\, \tilde{t}_2) \end{cases} \qquad \square$$

The precedence relation, $\tilde{<}_t$, on abstract times given by Definition 5.13 will be useful to determine whether two writes are performed at disjoint times, or the order two arbitrary events.

Definition 5.13 (Time precedence, $\tilde{<}_t$):

$$\begin{cases} \tilde{t} \,\tilde{<}_t\, \tilde{\top}_t & \textbf{if } \tilde{t} \neq \tilde{\top}_t \\ \tilde{\bot}_t \,\tilde{<}_t\, \tilde{t} & \textbf{if } \tilde{t} \neq \tilde{\bot}_t \\ \tilde{t}_1 \,\tilde{<}_t\, \tilde{t}_2 \Longleftrightarrow \max(\gamma_t(\tilde{t}_1)) < \min(\gamma_t(\tilde{t}_2)) & \textbf{if } \tilde{t}_1, \tilde{t}_2 \notin \{\tilde{\bot}_t, \tilde{\top}_t\} \end{cases} \qquad \square$$

The definitions of $\tilde{\sqsubseteq}_{var}$, $\tilde{\bigsqcap}_{var}$ and $\tilde{\bigsqcup}_{var}$ follow naturally from the definition of the domain (c.f., Definitions 3.26, 3.27 and 3.28) and are presented in Definitions 5.14, 5.15 and 5.16, respectively. However, these relations and operators cannot be used directly within the analysis to, e.g., join (merge) the histories of writes in several variable states. This is due to the fact that the history in the states might have different sequence information (i.e., traces), that would

be lost if merging the two states. Reading a safe and tight value for a variable requires the sequence information to be available. Therefore, the operations to be used within the analysis should instead be defined based on Definition 5.18 to ensure that all threads see safe values (see Definition 5.19) at all times. Note that Definition 5.17 defines the uniquely most recent write in a set of writes. This definition defines the most recent write both among several threads (globally) and for single threads (locally).

Definition 5.14 (Partial order for abstract variable states):

$$\begin{cases} \tilde{\mathfrak{x}} \,\tilde{\sqsubseteq}_{var}\, \tilde{\top}_{var} \\ \tilde{\bot}_{var} \,\tilde{\sqsubseteq}_{var}\, \tilde{\mathfrak{x}} \\ \tilde{\mathfrak{x}} \,\tilde{\sqsubseteq}_{var}\, \tilde{\mathfrak{x}}' \iff \forall x \in \mathbf{Var} : \forall \mathrm{T} \in \mathbf{Thrd} : (\tilde{\mathfrak{x}}\ x)\ \mathrm{T} \subseteq (\tilde{\mathfrak{x}}'\ x)\ \mathrm{T} \end{cases}$$ □

Definition 5.15 (Greatest lower bound of abstract variable states):

$$\begin{cases} \tilde{\top}_{var} \,\tilde{\sqcap}_{var}\, \tilde{\mathfrak{x}} = \tilde{\mathfrak{x}} \,\tilde{\sqcap}_{var}\, \tilde{\top}_{var} = \tilde{\mathfrak{x}} \\ \tilde{\bot}_{var} \,\tilde{\sqcap}_{var}\, \tilde{\mathfrak{x}} = \tilde{\mathfrak{x}} \,\tilde{\sqcap}_{var}\, \tilde{\bot}_{var} = \tilde{\bot}_{var} \\ (((\tilde{\mathfrak{x}} \,\tilde{\sqcap}_{var}\, \tilde{\mathfrak{x}}')\ x)\ \mathrm{T}) = ((\tilde{\mathfrak{x}}\ x)\ \mathrm{T}) \cap ((\tilde{\mathfrak{x}}'\ x)\ \mathrm{T}) \end{cases}$$ □

Definition 5.16 (Least upper bound of abstract variable states):

$$\begin{cases} \tilde{\top}_{var} \,\tilde{\sqcup}_{var}\, \tilde{\mathfrak{x}} = \tilde{\mathfrak{x}} \,\tilde{\sqcup}_{var}\, \tilde{\top}_{var} = \tilde{\top}_{var} \\ \tilde{\bot}_{var} \,\tilde{\sqcup}_{var}\, \tilde{\mathfrak{x}} = \tilde{\mathfrak{x}} \,\tilde{\sqcup}_{var}\, \tilde{\bot}_{var} = \tilde{\mathfrak{x}} \\ (((\tilde{\mathfrak{x}} \,\tilde{\sqcup}_{var}\, \tilde{\mathfrak{x}}')\ x)\ \mathrm{T}) = ((\tilde{\mathfrak{x}}\ x)\ \mathrm{T}) \cup ((\tilde{\mathfrak{x}}'\ x)\ \mathrm{T}) \end{cases}$$ □

Definition 5.17 (Time of most recent write):
The most recent write(s), $(\tilde{v},\tilde{t})$*, in a set of writes is defined such that* $\min(\gamma_t(\tilde{t})) \geq \min(\gamma_t(\tilde{t}'))$*, for all other writes,* $(\tilde{v}',\tilde{t}')$*. If several writes,* $(\tilde{v}',\tilde{t}')$*, are such that* $\min(\gamma_t(\tilde{t})) = \min(\gamma_t(\tilde{t}'))$*, the time of the most recent write,* $\tilde{t}$*, is uniquely determined from the write(s) with* $\max(\gamma_t(\tilde{t})) = \max(\{\max(\gamma_t(\tilde{t}')) \mid \tilde{t}'$ *ranges over the time-stamps of the writes such that* $\min(\gamma_t(\tilde{t})) = \min(\gamma_t(\tilde{t}'))\}$*.* □

NOTE. *The definition of the "time of the most recent write" and many of the upcoming algorithms could be very much simplified if the notion of lists were known. The positions of the writes in a list could simply correspond to the sequential order in which they occurred within a thread. Therefore, implementations of the upcoming algorithms could look very different from how they are defined here.*

Definition 5.18 (Safe write history):
An abstract variable state, $\tilde{\mathbb{x}}$, is safe at time $\tilde{t}$ if $\gamma_{var}(\tilde{\mathbb{x}})$ represents at least all the possible concrete variable states that can be valid at time $t \in \gamma_t(\tilde{t})$ for the given thread trace(s).

*Thus, to be safe at time $\tilde{t}$, $\tilde{\mathbb{x}}$ must, for each variable, $x \in$ **Var** and each thread, $\mathrm{T} \in$ **Thrd**, be such that $((\tilde{\mathbb{x}}\ x)\ \mathrm{T})$ contains at least*

1. *all writes, $(\tilde{v},\tilde{t}')$, by T on x, such that $\tilde{t}' \not\precsim_t \tilde{t} \wedge \tilde{t} \not\precsim_t \tilde{t}'$, and*
2. *the latest (most recent) write(s), $(\tilde{v},\tilde{t}')$, by T on x, such that $\tilde{t}' \precsim_t \tilde{t}$, if $\tilde{t}' \sqcap_t \tilde{t}^{mrw} \neq \tilde{\bot}_t$, where $\tilde{t}^{mrw}$ is the time of the globally most recent write,*

or,

3. *$(\tilde{\bot}_{val}, \tilde{\bot}_t)$, otherwise (i.e., if there are no writes that fit 1 or 2 above), or if no writes have occurred by T on x.*

*From how the concrete and abstract domains (c.f., Section 4.1 and this section) and transition rules (c.f., Section 4.2) are defined, it is apparent that $\tilde{\mathbb{x}}$ is a safe approximation of $\mathbb{x}$ (i.e., $\tilde{\mathbb{x}}$ contains safe write history) iff $\exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}) : \forall x \in$ **Var** $: \forall \mathrm{T} \in$ **Thrd** $: ((\mathbb{x}\ x)\ \mathrm{T}) \subseteq ((\mathbb{x}'\ x)\ \mathrm{T})$.* □

Definition 5.19 (Safe value of x as seen by thread T):
Assuming that $\tilde{\mathbb{x}}$ contains safe write history for all threads on variable x, according to Definition 5.18, then a safe value of x, as seen by thread T, at time $\tilde{t}$ is the least upper bound, $\tilde{\bigsqcup}_{val}$, of the values of at least the following writes on x.

1. *All writes, $\tilde{w}_{\mathrm{T}'} = (\tilde{v}_{\mathrm{T}'}, \tilde{t}_{\mathrm{T}'})$, for threads $\mathrm{T}' \in$ **Thrd** $\setminus \{\mathrm{T}\}$ on x such that $\tilde{t}_{\mathrm{T}'} \not\precsim_t \tilde{t} \wedge \tilde{t} \not\precsim_t \tilde{t}_{\mathrm{T}'}$.*
2. *The most recent write in $\{(\tilde{v}'_{\mathrm{T}'}, \tilde{t}'_{\mathrm{T}'}) \in (\tilde{\mathbb{x}}\ x)\ \mathrm{T}' \mid \tilde{t}'_{\mathrm{T}'} \precsim_t \tilde{t} \wedge \tilde{t}'_{\mathrm{T}'} \sqcap_t \tilde{t}^{mrw} \neq \tilde{\bot}_t\}$ for each thread $\mathrm{T}' \in$ **Thrd** $\setminus \{\mathrm{T}\}$, and the most recent*

write, $(\tilde{v}'_{\mathrm{T}}, \tilde{t}'_{\mathrm{T}}) \in (\tilde{\mathbb{x}}\ x)\ \mathrm{T}$, *such that* $\min(\gamma_t(\tilde{t}'_{\mathrm{T}})) \leq \min(\gamma_t(\tilde{t}))$, *if* $\tilde{t}'_{\mathrm{T}} \mathbin{\tilde{\cap}_t} \tilde{t}^{mrw} \neq \tilde{\bot}_t$, *where* $\tilde{t}^{mrw}$ *is the time of the (globally) most recent write in* $\{(\tilde{v}'_{\mathrm{T}}, \tilde{t}'_{\mathrm{T}})\} \cup \bigcup_{\mathrm{T}' \in \mathbf{Thrd} \setminus \{\mathrm{T}\}} \{(\tilde{v}'_{\mathrm{T}'}, \tilde{t}'_{\mathrm{T}'}) \in (\tilde{\mathbb{x}}\ x)\ \mathrm{T}' \mid \tilde{t}'_{\mathrm{T}'} \mathbin{\tilde{<}_t} \tilde{t}\}$. □

Algorithm 5.1 Partial Order of Abstract Variable States

```
 1: function PARTIALORDERVAR(x̃, x̃′)
 2:   for all x ∈ Var do
 3:     for all T ∈ Thrd do
 4:       W̃ ← ((x̃ x) T)
 5:       W̃′ ← ((x̃′ x) T)
 6:       while W̃ ≠ ∅ ∧ W̃′ ≠ ∅ do
 7:         w̃ ← EARLIESTWRITETHREAD(W̃)
 8:         w̃′ ← EARLIESTWRITETHREAD(W̃′)
 9:         W̃ ← W̃ \ {w̃}
10:         W̃′ ← W̃′ \ {w̃′}
11:         if w̃ ≠ w̃′ then
12:           if W̃′ = ∅ then
13:             for all w̃″ ∈ W̃ ∪ {w̃} do
14:               if w̃″ ⋢̃_w w̃′ then
15:                 return false
16:               end if
17:             end for
18:             W̃ ← ∅
19:           else
20:             return false
21:           end if
22:         end if
23:       end while
24:       if W̃ ≠ ∅ then
25:         return false
26:       end if
27:     end for
28:   end for
29:   return true
30: end function
```

The partial order for abstract variable states to be used within the analysis, $\tilde{\sqsubseteq}'_{var}$, is given by Definition 5.20 based on PARTIALORDERVAR, defined in Algorithm 5.1, taking the safety of write history (Definition 5.18) into account. Note that EARLIESTWRITETHREAD, as defined in Algorithm 5.2, returns a

Algorithm 5.2 Earliest Write for a Thread

```
1:  function EARLIESTWRITETHREAD(W̃)
2:      if W̃ = ∅ then
3:          return ⊥̃_w
4:      end if
5:      t̃_min ← α_t({∞})
6:      for all (ṽ, t̃) ∈ W̃ do
7:          if min(γ_t(t̃)) < min(γ_t(t̃_min)) then
8:              t̃_min ← t̃
9:          else if min(γ_t(t̃)) = min(γ_t(t̃_min)) then
10:             t̃_min ← t̃ ⊓̃_t t̃_min
11:         end if
12:     end for
13:     W̃′ ← {(ṽ, t̃) | (ṽ, t̃) ∈ W̃ ∧ t̃ = t̃_min}
14:     ṽ_min ← α_val({∞})
15:     for all (ṽ, t̃) ∈ W̃′ do
16:         if min(γ_val(ṽ)) < min(γ_val(ṽ_min)) then
17:             ṽ_min ← ṽ
18:         else if min(γ_val(ṽ)) = min(γ_val(ṽ_min)) then
19:             ṽ_min ← ṽ ⊓̃_val ṽ_min
20:         end if
21:     end for
22:     return (ṽ_min, t̃_min)
23: end function
```

deterministically defined write. The idea is that the history (trace) for each thread and variable should be the same in both states for the relation to be true. However, the histories are allowed to differ somewhat. The greater state could also contain newer writes than those in the history of the lesser state. It could also be the case that the newest write in the greater state is an upper bound to all of the most recent writes in the lesser state that are not part of both histories.

Definition 5.20 (Safe partial order of abstract variable states):

$$\begin{cases} \tilde{\mathbb{x}} \,\tilde{\sqsubseteq}'_{var}\, \tilde{\top}_{var} \\ \tilde{\bot}_{var} \,\tilde{\sqsubseteq}'_{var}\, \tilde{\mathbb{x}} \\ \tilde{\mathbb{x}} \,\tilde{\sqsubseteq}'_{var}\, \tilde{\mathbb{x}}' \iff \text{PARTIALORDERVAR}(\tilde{\mathbb{x}}, \tilde{\mathbb{x}}') \end{cases}$$ □

Based on this partial order relation, the lower bound and upper bound operators to be used within the analysis, $\tilde{\sqcap}'_{var}$ and $\tilde{\sqcup}'_{var}$, are given by Definitions 5.21 and 5.22, respectively. Note that MEETVAR is defined in Algorithm 5.3 and JOINVAR is defined in Algorithm 5.4.

Definition 5.21 (Safe lower bound of abstract variable states):

$$\begin{cases} \tilde{\top}_{var} \,\tilde{\sqcap}'_{var}\, \tilde{\mathbb{x}} = \tilde{\mathbb{x}} \,\tilde{\sqcap}'_{var}\, \tilde{\top}_{var} = \tilde{\mathbb{x}} \\ \tilde{\bot}_{var} \,\tilde{\sqcap}'_{var}\, \tilde{\mathbb{x}} = \tilde{\mathbb{x}} \,\tilde{\sqcap}'_{var}\, \tilde{\bot}_{var} = \tilde{\bot}_{var} \\ \tilde{\mathbb{x}} \,\tilde{\sqcap}'_{var}\, \tilde{\mathbb{x}}' = \text{MEETVAR}(\tilde{\mathbb{x}}, \tilde{\mathbb{x}}') \end{cases}$$ □

Definition 5.22 (Safe upper bound of abstract variable states):

$$\begin{cases} \tilde{\top}_{var} \,\tilde{\sqcup}'_{var}\, \tilde{\mathbb{x}} = \tilde{\mathbb{x}} \,\tilde{\sqcup}'_{var}\, \tilde{\top}_{var} = \tilde{\top}_{var} \\ \tilde{\bot}_{var} \,\tilde{\sqcup}'_{var}\, \tilde{\mathbb{x}} = \tilde{\mathbb{x}} \,\tilde{\sqcup}'_{var}\, \tilde{\bot}_{var} = \tilde{\mathbb{x}} \\ \tilde{\mathbb{x}} \,\tilde{\sqcup}'_{var}\, \tilde{\mathbb{x}}' = \text{JOINVAR}(\tilde{\mathbb{x}}, \tilde{\mathbb{x}}') \end{cases}$$ □

NOTE. *Neither $\tilde{\sqsubseteq}'_{var}$, $\tilde{\sqcap}'_{var}$ nor $\tilde{\sqcup}'_{var}$ is currently used by the analysis (c.f., Chapter 6) but are just presented for completeness of the abstraction since the operators cannot be directly based on the lattice. However, if, e.g., merging of configurations [25] is introduced to lower the complexity of the analysis, at least $\tilde{\sqcup}'_{var}$ will be needed.*

Algorithm 5.3 Meeting Two Abstract Variable States

```
1:  function MEETVAR(x̃, x̃′)
2:      x̃″ ← ⊥̃_var
3:      for all x ∈ Var do
4:          for all T ∈ Thrd do
5:              W̃ ← (x̃ x) T
6:              W̃′ ← (x̃′ x) T
7:              C ← ∅
8:              while W̃ ≠ ∅ ∧ W̃′ ≠ ∅ do
9:                  (ṽ, t̃) ← EARLIESTWRITETHREAD(W̃)
10:                 (ṽ′, t̃′) ← EARLIESTWRITETHREAD(W̃′)
11:                 W̃ ← W̃ \ (ṽ, t̃)
12:                 W̃′ ← W̃′ \ (ṽ′, t̃′)
13:                 if (ṽ, t̃) = (ṽ′, t̃′) then
14:                     C ← C ∪ {(ṽ, t̃)}
15:                 else if ṽ ⊓̃_val ṽ′ ≠ ⊥̃_val ∧ t̃ ⊓̃_t t̃′ ≠ ⊥̃_t ∧ W̃ = ∅ ∧ W̃′ = ∅ then
16:                     C ← C ∪ {(ṽ ⊓̃_val ṽ′, t̃ ⊓̃_t t̃′)}
17:                 else
18:                     W̃ ← ∅
19:                     W̃′ ← ∅
20:                 end if
21:             end while
22:             if C = ∅ then
23:                 ((x̃″ x) T) ← {(⊥̃_var, ⊥̃_t)}
24:             else
25:                 ((x̃″ x) T) ← C
26:             end if
27:         end for
28:     end for
29:     return x̃″
30: end function
```

Algorithm 5.4 Joining Two Abstract Variable States

```
 1: function JOINVAR(x̃, x̃′)
 2:     x̃″ ← ⊥̃_var
 3:     for all x ∈ Var do
 4:         for all T ∈ Thrd do
 5:             W̃ ← (x̃ x) T
 6:             W̃′ ← (x̃′ x) T
 7:             C ← ∅
 8:             M ← (⊥̃_val, ⊥̃_t)
 9:             while W̃ ≠ ∅ ∨ W̃′ ≠ ∅ do
10:                 w̃ ← EARLIESTWRITETHREAD(W̃)
11:                 w̃′ ← EARLIESTWRITETHREAD(W̃′)
12:                 if w̃ = w̃′ then
13:                     C ← C ∪ {w̃}
14:                     W̃ ← W̃ \ {w̃}
15:                     W̃′ ← W̃′ \ {w̃′}
16:                 else if W̃ = ∅ then
17:                     C ← C ∪ W̃′
18:                     W̃′ ← ∅
19:                 else if W̃′ = ∅ then
20:                     C ← C ∪ W̃
21:                     W̃ ← ∅
22:                 else
23:                     M ← (⊔̃_w W̃) ⊔̃_w (⊔̃_w W̃′)
24:                     W̃ ← ∅
25:                     W̃′ ← ∅
26:                 end if
27:             end while
28:             ((x̃″ x) T) ← C
29:             if M ≠ (⊥̃_val, ⊥̃_t) then
30:                 ((x̃″ x) T) ← ((x̃″ x) T) ∪ {M}
31:             end if
32:         end for
33:     end for
34:     return x̃″
35: end function
```

Algorithm 5.5 Write to Variable

1: **function** WRITE($\mathrm{T}, \tilde{\mathbb{x}}, x, \tilde{w}$)
2: $(\tilde{\mathbb{x}}'\ x')\ \mathrm{T}' \leftarrow \begin{cases} ((\tilde{\mathbb{x}}\ x)\ \mathrm{T}) \cup \{\tilde{w}\} & \textbf{if } x' = x \wedge \mathrm{T}' = \mathrm{T} \\ (\tilde{\mathbb{x}}\ x')\ \mathrm{T}' & \textbf{otherwise} \end{cases}$
3: **return** $\tilde{\mathbb{x}}'$
4: **end function**

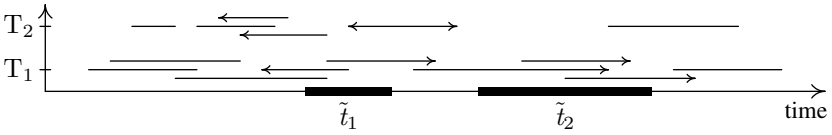


Figure 5.3: The time-stamps of the writes considered by READ$(\tilde{\mathbb{x}}, x, \mathrm{T}_1, \tilde{t}_1)$ and READ$(\tilde{\mathbb{x}}, x, \mathrm{T}_2, \tilde{t}_2)$.

WRITE$(\mathrm{T}, \tilde{\mathbb{x}}, x, \tilde{w})$, as defined in Algorithm 5.5, safely (Lemma 5.23) adds the write, $\tilde{w}$, to the set of write-history for thread T, i.e., to $((\tilde{\mathbb{x}}\ x)\ \mathrm{T})$.

Lemma 5.23 (Soundness of WRITE):
Assuming that $\tilde{\mathbb{x}}$ *contains safe write history for variable* x *and thread* T *(c.f., Definition 5.18) before the write by thread* T *is performed at time* $\tilde{t}$*, then so will* WRITE$(\mathrm{T}, \tilde{\mathbb{x}}, x, (\tilde{v}, \tilde{t}))$. □

PROOF. Since WRITE$(\mathrm{T}, \tilde{\mathbb{x}}, x, (\tilde{v}, \tilde{t}))$ simply adds the write $(\tilde{v}, \tilde{t})$ to the history of thread T's writes on variable x in the state $\tilde{\mathbb{x}}$, and $\tilde{\mathbb{x}}$ is assumed to contain safe write history for T on x, WRITE$(\mathrm{T}, \tilde{\mathbb{x}}, x, (\tilde{v}, \tilde{t}))$ trivially fulfills the safety condition in Definition 5.18 with regards to T and x. ■

Using the sequence and timing information provided by Definition 5.19, READ$(\tilde{\mathbb{x}}, x, \mathrm{T}, \tilde{t})$, as defined in Algorithm 5.6, only takes the writes that might be valid at $\tilde{t}$ (the point in time when T issues the READ) into consideration for its returned value $\tilde{v} \in \mathbf{V\tilde{a}l}$, which is safe (Lemma 5.26). These writes, $\tilde{w} = (\tilde{v}', \tilde{t}')$, come from two categories. The first category covers the writes on x for threads $\mathrm{T}' \in \mathbf{Thrd} \setminus \{\mathrm{T}\}$ whose "time-stamps" overlap in time with $\tilde{t}$, i.e., $\tilde{t} \tilde{\sqcap}_t \tilde{t}' \neq \tilde{\perp}_t$. The second category covers the most recent write on x for all threads (including T) such that its time-stamp overlaps with the overall most recent write of any write, not belonging to the first category. Note that any write for thread T with a time-stamp that begins after the beginning of $\tilde{t}$ is discarded. So is any write for $\mathrm{T}' \in \mathbf{Thrd} \setminus \{\mathrm{T}\}$ such that its time-stamp completely succeeds $\tilde{t}$. This is because such writes can simply not have occurred at the time of issuing the READ

Algorithm 5.6 Read from Variable

function READ($\tilde{\mathrm{x}}, x, \mathrm{T}, \tilde{t}$)
  $\tilde{\mathrm{x}}' \leftarrow \tilde{\bot}_{var}$
  for all $\mathrm{T}' \in$ **Thrd** $\setminus \{\mathrm{T}\}$ **do**
    $((\tilde{\mathrm{x}}'\ x)\ \mathrm{T}') \leftarrow \{(\tilde{v}', \tilde{t}') \in ((\tilde{\mathrm{x}}\ x)\ \mathrm{T}') \mid \tilde{t} \not\sqsubset_t \tilde{t}'\}$
  end for
  $((\tilde{\mathrm{x}}'\ x)\ \mathrm{T}) \leftarrow \{(\tilde{v}', \tilde{t}') \in ((\tilde{\mathrm{x}}\ x)\ \mathrm{T}) \mid \min(\gamma_t(\tilde{t})) \geq \min(\gamma_t(\tilde{t}'))\}$
  $\tilde{W} \leftarrow \emptyset$
  for all $\mathrm{T}' \in$ **Thrd** $\setminus \{\mathrm{T}\}$ **do**
    $\tilde{W}_{\mathrm{T}'} \leftarrow \{(\tilde{v}', \tilde{t}') \in ((\tilde{\mathrm{x}}'\ x)\ \mathrm{T}') \mid \tilde{t}' \not\sqsubset_t \tilde{t}\}$
    $((\tilde{\mathrm{x}}'\ x)\ \mathrm{T}') \leftarrow ((\tilde{\mathrm{x}}'\ x)\ \mathrm{T}') \setminus \tilde{W}_{\mathrm{T}'}$
    $\tilde{W} \leftarrow \tilde{W} \cup \tilde{W}_{\mathrm{T}'}$
  end for
  $\tilde{t}^{mrw} \leftarrow$ MOSTRECENTWRITETIME($\tilde{\mathrm{x}}', x$)
  if $\tilde{t}^{mrw} \neq \tilde{\bot}_t$ **then**
    for all $\mathrm{T}' \in$ **Thrd** **do**
      $\tilde{t}^{mrw}_{\mathrm{T}'} \leftarrow$ MOSTRECENTWRITETIMETHREAD($(\tilde{\mathrm{x}}'\ x)\ \mathrm{T}'$)
      $\tilde{W} \leftarrow \tilde{W} \cup \{(\tilde{v}', \tilde{t}') \in ((\tilde{\mathrm{x}}'\ x)\ \mathrm{T}') \mid \tilde{t}' \tilde{\sqcap}_t \tilde{t}^{mrw}_{\mathrm{T}'} \neq \tilde{\bot}_t \wedge \tilde{t}' \tilde{\sqcap}_t \tilde{t}^{mrw} \neq \tilde{\bot}_t\}$
    end for
  end if
  $\tilde{v} \leftarrow \begin{cases} \tilde{\bigsqcup}_{val}\{\tilde{v}' \mid \exists \tilde{t}' \in \tilde{\textbf{Time}} : (\tilde{v}', \tilde{t}') \in \tilde{W}\} & \textbf{if } \tilde{W} \neq \emptyset \\ [-\infty, \infty] & \textbf{otherwise} \end{cases}$
  return $\tilde{v}$
end function

Algorithm 5.7 Time of Most Recent Write

function MOSTRECENTWRITETIME($\tilde{\mathrm{x}}, x$)
  return MOSTRECENTWRITETIMETHREAD($\bigcup_{\mathrm{T} \in \textbf{Thrd}}((\tilde{\mathrm{x}}\ x)\ \mathrm{T})$)
end function

Algorithm 5.8 Time of Most Recent Write in Thread

function MOSTRECENTWRITETIMETHREAD($\tilde{W}$)
  if $\tilde{W} = \emptyset$ **then**
    return $\tilde{\bot}_t$
  end if
  $t_{min} \leftarrow \max(\{\min(\gamma_t(\tilde{t})) \mid \exists \tilde{v} \in \tilde{\textbf{Val}} : (\tilde{v}, \tilde{t}) \in \tilde{W}\})$
  $t_{max} \leftarrow \max(\bigcup\{\gamma_t(\tilde{t}) \mid \exists \tilde{v} \in \tilde{\textbf{Val}} : (\tilde{v}, \tilde{t}) \in \tilde{W} \wedge \min(\gamma_t(\tilde{t})) = t_{min}\})$
  return $\alpha_t(\{t_{min}, t_{max}\})$
end function

(and will thus usually not be included in $\tilde{\mathbb{x}}$ at all). An illustration of the time-stamps of the writes in T_1 and T_2 that must be considered by $\textsc{read}(\tilde{\mathbb{x}},x,\mathrm{T}_1,\tilde{t}_1)$ (lines with arrow heads pointing left) and $\textsc{read}(\tilde{\mathbb{x}},x,\mathrm{T}_2,\tilde{t}_2)$ (lines with arrow heads pointing right) is given in Figure 5.3. The returned value, $\tilde{v}$, is the least upper bound of the values of the considered writes. Note that MOSTRECENTWRITETIME and MOSTRECENTWRITETIMETHREAD are defined based on Definition 5.17 in Algorithms 5.7 and 5.8, respectively, and that these functions give the time of the most recent write among the writes in a set of writes (Lemmas 5.24 and 5.25).

Lemma 5.24 (Soundness of MOSTRECENTWRITETIMETHREAD):
MOSTRECENTWRITETIMETHREAD$(\tilde{W})$, *defined in Algorithm 5.8, gives the time of the most recent write in* $\tilde{W}$. □

PROOF. This proof will be conducted based on the structure of Algorithm 5.8.

If $\tilde{W} = \emptyset$, then $\tilde{\bot}_t$ is returned. Otherwise, t_{min} is the greatest lower limit of the time-stamp of any write in $\tilde{W}$ $(\max(\{\min(\gamma_t(\tilde{t}) \mid \exists \tilde{v} \in \mathbf{V\tilde{a}l} : (\tilde{v},\tilde{t}) \in \tilde{W}\}))$ and t_{max} is the greatest upper limit of the time-stamps of the writes in $\tilde{W}$ such that the lower limit of their time-stamps are equal to t_{min} $(\max(\bigcup\{\gamma_t(\tilde{t}) \mid \exists \tilde{v} \in \mathbf{V\tilde{a}l} : (\tilde{v},\tilde{t}) \in \tilde{W} \wedge \min(\gamma_t(\tilde{t})) = t_{min}\}))$. Thus, $\alpha(\{t_{min},t_{max}\})$ is the time of the most recent write in $\tilde{W}$, as given by Definition 5.17. ■

Lemma 5.25 (Soundness of MOSTRECENTWRITETIME):
MOSTRECENTWRITETIME$(\tilde{\mathbb{x}},x)$, *defined in Algorithm 5.7, gives the time of the globally most recent write on* x *in* $\tilde{\mathbb{x}}$. □

PROOF. This proof is trivial since MOSTRECENTWRITETIMETHREAD$(\tilde{W})$ is the time of the most recent write in $\tilde{W}$ (Lemma 5.24) and the set of writes, $\tilde{W}$, is $\bigcup_{\mathrm{T}\in\mathbf{Thrd}}((\tilde{\mathbb{x}}\ x)\ \mathrm{T})$; i.e., $\tilde{W}$ is a set containing the writes by all threads in **Thrd** on x. Thus the time of the globally most recent write, as given by Definition 5.17, is returned. ■

Lemma 5.26 (Soundness of READ):
Assuming that $\tilde{\mathbb{x}}$ *contains safe write history at* $\tilde{t}$ *(Definition 5.18), a safe value for* x *at* $\tilde{t}$ *as seen by thread* T *(Definition 5.19) is given by* $\textsc{read}(\tilde{\mathbb{x}},x,\mathrm{T},\tilde{t})$. □

PROOF. The proof amounts to showing that $\textsc{read}(\tilde{\mathbb{x}},x,\mathrm{T},\tilde{t})$ is an upper bound to the values of the writes given by Definition 5.19; i.e., to show that all writes given by Definition 5.19 are included in $\tilde{W}$.

On line 4, the new variable state, $\tilde{\mathbb{x}}'$, is defined to contain all writes, $(\tilde{v}',\tilde{t}') \in ((\tilde{\mathbb{x}}\ x)\ \mathrm{T})$, such that $\tilde{t} \not\precsim_t \tilde{t}'$, for each $\mathrm{T}' \in \mathbf{Thrd} \setminus \{\mathrm{T}\}$. On lines 9–11, the writes,

$(\tilde{v}',\tilde{t}') \in ((\tilde{\mathbb{x}}'\ x)\ \mathrm{T}')$, for all $\mathrm{T}' \in \mathbf{Thrd} \setminus \{\mathrm{T}\}$, such that $\tilde{t}' \not\precsim_t \tilde{t}$, are extracted (i.e., identified and removed) from $((\tilde{\mathbb{x}}'\ x)\ \mathrm{T}')$ and put in the set $\tilde{W}$. Thus, $\tilde{W}$ contains all the writes specified by 1 in Definition 5.19.

On line 6, $((\tilde{\mathbb{x}}'\ x)\ \mathrm{T})$ is defined to contain all writes, $(\tilde{v}',\tilde{t}') \in ((\tilde{\mathbb{x}}\ x)\ \mathrm{T})$, such that $\min(\gamma_t(\tilde{t})) \geq \min(\gamma_t(\tilde{t}'))$, and $((\tilde{\mathbb{x}}'\ x)\ \mathrm{T}')$, for each $\mathrm{T}' \in \mathbf{Thrd} \setminus \{\mathrm{T}\}$, now contains all the writes, $(\tilde{v}',\tilde{t}') \in ((\tilde{\mathbb{x}}\ x)\ \mathrm{T}')$, such that $\tilde{t}' \precsim_t \tilde{t}$.

On line 13, the time of the (global) most recent write on x among all threads, i.e., the most recent write in $\bigcup\{(\tilde{\mathbb{x}}'\ x)\ \mathrm{T}' \mid \mathrm{T}' \in \mathbf{Thrd}\}$, is determined (Lemma 5.25), while at line 16, the time of the most recent write for each thread is determined (Lemma 5.24). If the time of the most recent write for a thread overlaps with the time of the global most recent write, then all writes overlapping in time with the most recent write for that thread are added to $\tilde{W}$ (line 17). Thus, $\tilde{W}$ now also contains (at least) all the writes specified by 2 in Definition 5.19.

Finally, on line 20, the least upper bound of the values of the writes in $\tilde{W}$ is determined. On the next line, it is returned if $\tilde{W} \neq \emptyset$. If $\tilde{W} = \emptyset$, then $[-\infty,\infty]$ is returned, which trivially is a safe approximation of the corresponding value read (i.e., $v \in \gamma_{int}([-\infty,\infty])$) in the concrete case (c.f., Table 4.2). ∎

Since READ$(\tilde{\mathbb{x}},x,\mathrm{T},\tilde{t})$ discards writes from thread $\mathrm{T}' \in \mathbf{Thrd}$ that are too old to be valid at time $\tilde{t}$ (and writes occurring after $\tilde{t}$) for its returned value, and since time is assumed to never progress negatively (i.e., backwards; c.f., Lemma 4.2), the discarded writes can safely be removed from $((\tilde{\mathbb{x}}\ x)\ \mathrm{T}')$. TRIM, defined in Algorithm 5.9, safely (Lemma 5.27) removes the outdated writes from $((\tilde{\mathbb{x}}\ x)\ \mathrm{T}')$ for all $\mathrm{T}' \in \mathbf{Thrd}$. Thus, TRIM can be used to lower the space complexity of the analysis. Note that SPLITSET$(\tilde{W},\tilde{t})$, as defined in Algorithm 5.10, is used to split a set of writes into two sets where the first set contains all writes, $(\tilde{v},\tilde{t}')$, such that $\tilde{t}' \mathbin{\tilde{\sqcap}_t} \tilde{t} \neq \tilde{\bot}_t$, and the second set contains all other writes.

Lemma 5.27 (Soundness of TRIM):
If $\tilde{\mathbb{x}}$ contains safe write history at time $\tilde{t}$ (c.f., Definition 5.18), then so does TRIM$(\tilde{\mathbb{x}},\tilde{t})$. □

PROOF. Given that $\tilde{\mathbb{x}}$ is safe, it must be shown that, for any variable, $x \in \mathbf{Var}$, and any thread, $\mathrm{T} \in \mathbf{Thrd}$, $((\mathrm{TRIM}(\tilde{\mathbb{x}},\tilde{t})\ x)\ \mathrm{T})$ contains at least (c.f., Definition 5.18)

1. all writes, $(\tilde{v},\tilde{t}')$, of $((\tilde{\mathbb{x}}\ x)\ \mathrm{T})$ such that $\tilde{t}' \not\precsim_t \tilde{t} \wedge \tilde{t} \not\precsim_t \tilde{t}'$, and

Algorithm 5.9 Trim Variable State

function TRIM($\tilde{\mathbb{x}}, \tilde{t}$)
  $\tilde{\mathbb{x}}' \leftarrow \tilde{\bot}_{var}$
  $\tilde{\mathbb{x}}'' \leftarrow \tilde{\bot}_{var}$
  for all $x \in$ **Var do**
    $\langle [F_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}} \rangle \leftarrow \langle [\emptyset]_{\mathrm{T} \in \mathbf{Thrd}} \rangle$
    $\langle [O_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}} \rangle \leftarrow \langle [\emptyset]_{\mathrm{T} \in \mathbf{Thrd}} \rangle$
    $\langle [N_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}} \rangle \leftarrow \langle [\emptyset]_{\mathrm{T} \in \mathbf{Thrd}} \rangle$
    for all $\mathrm{T} \in$ **Thrd do**
      $F_{\mathrm{T}} \leftarrow \{(\tilde{v}, \tilde{t}') \in (\tilde{\mathbb{x}}\ x)\ \mathrm{T} \mid \tilde{t} \precsim_t \tilde{t}'\}$
      $(O_{\mathrm{T}}, N_{\mathrm{T}}) \leftarrow$ SPLITSET$((\tilde{\mathbb{x}}\ x)\ \mathrm{T}, \tilde{t})$
      $((\tilde{\mathbb{x}}'\ x)\ \mathrm{T}) \leftarrow N_{\mathrm{T}} \setminus F_{\mathrm{T}}$
    end for
    $\tilde{t}^{mrw} \leftarrow$ MOSTRECENTWRITETIME$(\tilde{\mathbb{x}}', x)$
    for all $\mathrm{T} \in$ **Thrd do**
      $\tilde{W}_{\mathrm{T}} \leftarrow \emptyset$
      $\tilde{t}_{\mathrm{T}}^{mrw} \leftarrow$ MOSTRECENTWRITETIMETHREAD$((\tilde{\mathbb{x}}'\ x)\ \mathrm{T})$
      if $\tilde{t}_{\mathrm{T}}^{mrw} \sqcap_t \tilde{t}^{mrw} = \tilde{\bot}_t \wedge F_{\mathrm{T}} = \emptyset \wedge O_{\mathrm{T}} = \emptyset$ **then**
        $\tilde{W}_{\mathrm{T}} \leftarrow \{(\tilde{\bot}_{val}, \tilde{\bot}_t)\}$
      else
        $\tilde{W}_{\mathrm{T}} \leftarrow \{(\tilde{v}, \tilde{t}') \in ((\tilde{\mathbb{x}}'\ x)\ \mathrm{T}) \mid \tilde{t}' \sqcap_t \tilde{t}_{\mathrm{T}}^{mrw} \neq \tilde{\bot}_t \wedge \tilde{t}^{mrw} \sqcap_t \tilde{t}_{\mathrm{T}}^{mrw} \neq \tilde{\bot}_t\}$
      end if
      $((\tilde{\mathbb{x}}''\ x)\ \mathrm{T}) \leftarrow F_{\mathrm{T}} \cup O_{\mathrm{T}} \cup \tilde{W}_{\mathrm{T}}$
    end for
  end for
  return $\tilde{\mathbb{x}}''$
end function

Algorithm 5.10 Split Set of Writes

function SPLITSET$(\tilde{W}, \tilde{t})$
  $O \leftarrow \{(\tilde{v}, \tilde{t}') \in \tilde{W} \mid \tilde{t} \not\precsim_t \tilde{t}' \wedge \tilde{t}' \not\precsim_t \tilde{t}\}$
  $N \leftarrow \{(\tilde{v}, \tilde{t}') \in \tilde{W} \mid \tilde{t} \precsim_t \tilde{t}' \vee \tilde{t}' \precsim_t \tilde{t}\}$
  return (O, N)
end function

2. any write, $(\tilde{v},\tilde{t}')$, of $((\tilde{\mathbb{x}}\ x)\ \mathrm{T})$ such that $\tilde{t}' \precsim_t \tilde{t}$, if $\tilde{t}' \tilde{\sqcap}_t \tilde{t}^{mrw} \neq \tilde{\bot}_t$, where $\tilde{t}^{mrw}$ is the time of the globally most recent write of the writes preceding $\tilde{t}$,

or,

3. $(\tilde{\bot}_{val},\tilde{\bot}_t)$, if there are no writes fitting the definition of the previous two categories (e.g., if all writes made by T are outdated or no writes have occurred by T on x; i.e., if $((\tilde{\mathbb{x}}\ x)\ \mathrm{T}) = \{(\tilde{\bot}_{val},\tilde{\bot}_t)\}$).

Before advancing to the proof procedure, note that $\neg(\tilde{t}' \not\precsim_t \tilde{t} \wedge \tilde{t} \not\precsim_t \tilde{t}')$ whenever $\tilde{t}$ or $\tilde{t}'$ is $\tilde{\bot}_t$ or $\tilde{\top}_t$. If they are not, note that (it is implicitly assumed that $\tilde{\mathbf{Time}} = \mathbf{Intv}$):

$$\begin{aligned}
\tilde{t}' \not\precsim_t \tilde{t} \wedge \tilde{t} \not\precsim_t \tilde{t}' &\overset{Def.\ 5.13}{\iff} \max(\gamma_t(\tilde{t}')) \not< \min(\gamma_t(\tilde{t})) \wedge \max(\gamma_t(\tilde{t})) \not< \min(\gamma_t(\tilde{t}')) \\
&\overset{calc.}{\iff} \max(\gamma_t(\tilde{t}')) \geq \min(\gamma_t(\tilde{t})) \wedge \max(\gamma_t(\tilde{t})) \geq \min(\gamma_t(\tilde{t}')) \\
&\overset{calc.}{\iff} \min(\{\max(\gamma_t(\tilde{t})), \max(\gamma_t(\tilde{t}'))\}) \geq \\
&\qquad\qquad \max(\{\min(\gamma_t(\tilde{t})), \min(\gamma_t(\tilde{t}'))\}) \\
&\overset{Def.\ 3.34}{\iff} \tilde{t} \tilde{\sqcap}_t \tilde{t}' \neq \tilde{\bot}_t
\end{aligned}$$

Now, assume that $\tilde{\mathbb{x}}$ contains safe write history. The structure of the algorithm gives that for each $x \in \mathbf{Var}$:

- For each thread, $\mathrm{T} \in \mathbf{Thrd}$, the set F_{T} contains the writes, $(\tilde{v},\tilde{t}')$, by T on x such that $\tilde{t} \precsim_t \tilde{t}'$; i.e., writes that occur after $\tilde{t}$. Note that this captures all writes, $(\tilde{v},\tilde{t}')$, such that $\tilde{t}' = \tilde{\top}_t$ as long as $\tilde{t} \neq \tilde{\top}_t$.
- For each thread, $\mathrm{T} \in \mathbf{Thrd}$, the set O_{T} contains the writes, $(\tilde{v},\tilde{t}')$, by T on x such that $\tilde{t}' \not\precsim_t \tilde{t} \wedge \tilde{t} \not\precsim_t \tilde{t}'$.
- For each thread, $\mathrm{T} \in \mathbf{Thrd}$, the set N_{T} contains the writes, $(\tilde{v},\tilde{t}')$, by T on x such that $\tilde{t}' \precsim_t \tilde{t} \vee \tilde{t} \precsim_t \tilde{t}'$. Note that this captures all writes, $(\tilde{v},\tilde{t}')$, such that $\tilde{t}' = \tilde{\top}_t$ or $\tilde{t}' = \tilde{\bot}_t$.
- $\tilde{t}^{mrw}$ is determined from $\tilde{\mathbb{x}}'$, for which all writes, $(\tilde{v},\tilde{t}') \in ((\tilde{\mathbb{x}}'\ x)\ \mathrm{T})$, on x by each thread, $\mathrm{T} \in \mathbf{Thrd}$, are such that $\tilde{t}' \precsim_t \tilde{t}$.
- For each thread $\mathrm{T} \in \mathbf{Thrd}$, $\tilde{W}_{\mathrm{T}} = \{(\tilde{\bot}_{val},\tilde{\bot}_t)\}$ whenever $\tilde{t}^{mrw}_{\mathrm{T}} \tilde{\sqcap}_t \tilde{t}^{mrw} = \tilde{\bot}_t \wedge F_{\mathrm{T}} = \emptyset \wedge O_{\mathrm{T}} = \emptyset$, otherwise, $\tilde{W}_{\mathrm{T}}$ contains all the writes by T on x such that $\tilde{t}' \precsim_t \tilde{t} \wedge \tilde{t}' \tilde{\sqcap}_t \tilde{t}^{mrw}_{\mathrm{T}} \neq \emptyset$ if $\tilde{t}^{mrw} \tilde{\sqcap}_t \tilde{t}^{mrw}_{\mathrm{T}} \neq \tilde{\bot}_t$, where $\tilde{t}^{mrw}_{\mathrm{T}}$ is the time of the most recent write of the writes, $(\tilde{v},\tilde{t}') \in ((\tilde{\mathbb{x}}'\ x)\ \mathrm{T})$; i.e., the writes, $(\tilde{v},\tilde{t}') \in ((\tilde{\mathbb{x}}\ x)\ \mathrm{T})$, such that $\tilde{t}' \precsim_t \tilde{t}$.

Assume that $\tilde{t}^{mrw}_{\mathrm{T}} \tilde{\sqcap}_t \tilde{t}^{mrw} \neq \tilde{\bot}_t \vee F_{\mathrm{T}} \neq \emptyset \vee O_{\mathrm{T}} \neq \emptyset$, then $((\tilde{\mathbb{x}}'' \, x) \, \mathrm{T})$ contains all writes, $(\tilde{v}, \tilde{t}')$, such that $\tilde{t} \tilde{<}_t \tilde{t}' \vee (\tilde{t}' \not\tilde{<}_t \tilde{t} \wedge \tilde{t} \not\tilde{<}_t \tilde{t}') \vee (\tilde{t}' \tilde{\sqcap}_t \tilde{t}^{mrw}_{\mathrm{T}} \neq \tilde{\bot}_t \wedge \tilde{t}^{mrw} \tilde{\sqcap}_t \tilde{t}^{mrw}_{\mathrm{T}} \neq \tilde{\bot}_t)$. Thus conditions 1 and 2 above are fulfilled. Note that all writes, $(\tilde{v}, \tilde{t}')$, occurring after $\tilde{t}$ (i.e., $\tilde{t} \tilde{<}_t \tilde{t}'$) are present in $\tilde{\mathbb{x}}''$; i.e., they are not trimmed away from $\tilde{\mathbb{x}}$.

Next, assume that $\tilde{t}^{mrw}_{\mathrm{T}} \tilde{\sqcap}_t \tilde{t}^{mrw} = \tilde{\bot}_t \wedge F_{\mathrm{T}} = \emptyset \wedge O_{\mathrm{T}} = \emptyset$, then $((\tilde{\mathbb{x}}'' \, x) \, \mathrm{T}) = \{(\tilde{\bot}_{val}, \tilde{\bot}_t)\}$. Thus condition 3 above is fulfilled.

Thus, $\tilde{\mathbb{x}}''$ (and hence $\textsc{trim}(\tilde{\mathbb{x}}, \tilde{t})$) contains safe write history for all variables, $x \in \mathbf{Var}$ and threads, $\mathrm{T} \in \mathbf{Thrd}$. ■

5.6 Abstract Lock States

In this section, a Galois connection, $\langle \alpha_{lock}, \gamma_{lock} \rangle$, between the concrete domain $\mathscr{P}(\mathbf{Lck} \to (\mathbf{Lck_{stt}} \times \mathbf{Thrd}_\bot \times \mathbf{Time} \times \mathbf{Thrd}_\bot \times \mathbf{Time}))$ and the abstract domain $(\mathbf{Lck} \to (\mathbf{Lck_{stt}} \times \mathbf{Thrd}_\bot \times \tilde{\mathbf{Time}} \times \mathbf{Thrd}_\bot \times \tilde{\mathbf{Time}})) \cup \{\tilde{\bot}_{lock}, \tilde{\top}_{lock}\}$, for lock states, will be defined (Theorem 5.34). The definitions of γ_{lock} (which is monotone by Lemma 5.33) and α_{lock} are presented in Definitions 5.28 and 5.29, respectively. Note that $\tilde{\textsc{stt}}$, $\tilde{\textsc{own}}$, $\tilde{\textsc{dl}}$, $\tilde{\textsc{pown}}$ and $\tilde{\textsc{rel}}$ are defined in Table 5.4, and that STT, OWN, DL, POWN and REL were defined in Table 4.5 on page 45.

$\tilde{\mathbb{l}}$ is the bottom element, $\tilde{\bot}_{lock}$, if $\exists lck \in \mathbf{Lck} : (\tilde{\mathbb{l}} \, lck = (u, \mathrm{T}, \tilde{\bot}_t, \mathrm{T}', \tilde{t}) \vee \tilde{\mathbb{l}} \, lck = (u, \mathrm{T}, \tilde{t}, \mathrm{T}', \tilde{\bot}_t))$ for some lock state, $u \in \{unlocked, locked\}$, owner, $\mathrm{T} \in \mathbf{Thrd}$, and previous owner, $\mathrm{T}' \in \mathbf{Thrd}$. The top element, $\tilde{\top}_{lock}$, identifies the mappings, $\tilde{\mathbb{l}}$, such that $\forall lck \in \mathbf{Lck} : \tilde{\mathbb{l}} \, lck = (u, \mathrm{T}, \tilde{\top}_t, \mathrm{T}', \tilde{\top}_t)$, for any lock state, $u \in \{unlocked, locked\}$, owner, $\mathrm{T} \in \mathbf{Thrd}$, and previous owner, $\mathrm{T}' \in \mathbf{Thrd}$.

Definition 5.28 (Concretization of an abstract lock state):

$$\begin{cases} \gamma_{lock}(\tilde{\top}_{lock}) = \mathbf{Lck} \to (\mathbf{Lck_{stt}} \times \mathbf{Thrd}_\bot \times \mathbf{Time} \times \mathbf{Thrd}_\bot \times \mathbf{Time}) \\ \gamma_{lock}(\tilde{\bot}_{lock}) = \emptyset \\ \gamma_{lock}(\tilde{\mathbb{l}}) = \gamma_{lock}(\lambda lck \in \mathbf{Lck}.(u_{lck}, \mathrm{T}_{lck}, \tilde{t}_{lck}, \mathrm{T}'_{lck}, \tilde{t}'_{lck})) \\ \qquad = \{\lambda lck \in \mathbf{Lck}.(u_{lck}, \mathrm{T}_{lck}, t_{lck}, \mathrm{T}'_{lck}, t'_{lck}) \mid \\ \qquad\qquad t_{lck} \in \gamma_t(\tilde{t}_{lck}) \wedge t'_{lck} \in \gamma_t(\tilde{t}'_{lck})\} \end{cases}$$ □

Definition 5.29 (Abstraction of a set of lock states):

$$\alpha_{lock}(\mathbb{L}) = \tilde{\bigsqcap}_{lock} \{\tilde{\mathbb{l}} \mid \mathbb{L} \subseteq \gamma_{lock}(\tilde{\mathbb{l}})\}$$ □

$$\text{S}\tilde{\text{T}}\text{T} : (\mathbf{Lck_{stt}} \times \mathbf{Thrd}_\perp \times \tilde{\mathbf{Time}} \times \mathbf{Thrd}_\perp \times \tilde{\mathbf{Time}}) \to \mathbf{Lck_{stt}}$$
$$\text{S}\tilde{\text{T}}\text{T}((u, \text{T}, \tilde{t}, \text{T}', \tilde{t}')) = u$$

$$\text{O}\tilde{\text{W}}\text{N} : (\mathbf{Lck_{stt}} \times \mathbf{Thrd}_\perp \times \tilde{\mathbf{Time}} \times \mathbf{Thrd}_\perp \times \tilde{\mathbf{Time}}) \to \mathbf{Thrd}_\perp$$
$$\text{O}\tilde{\text{W}}\text{N}((u, \text{T}, \tilde{t}, \text{T}', \tilde{t}')) = \text{T}$$

$$\text{D}\tilde{\text{L}} : (\mathbf{Lck_{stt}} \times \mathbf{Thrd}_\perp \times \tilde{\mathbf{Time}} \times \mathbf{Thrd}_\perp \times \tilde{\mathbf{Time}}) \to \tilde{\mathbf{Time}}$$
$$\text{D}\tilde{\text{L}}((u, \text{T}, \tilde{t}, \text{T}', \tilde{t}')) = \tilde{t}$$

$$\text{PO}\tilde{\text{W}}\text{N} : (\mathbf{Lck_{stt}} \times \mathbf{Thrd}_\perp \times \tilde{\mathbf{Time}} \times \mathbf{Thrd}_\perp \times \tilde{\mathbf{Time}}) \to \mathbf{Thrd}_\perp$$
$$\text{PO}\tilde{\text{W}}\text{N}((u, \text{T}, \tilde{t}, \text{T}', \tilde{t}')) = \text{T}'$$

$$\text{R}\tilde{\text{E}}\text{L} : (\mathbf{Lck_{stt}} \times \mathbf{Thrd}_\perp \times \tilde{\mathbf{Time}} \times \mathbf{Thrd}_\perp \times \tilde{\mathbf{Time}}) \to \tilde{\mathbf{Time}}$$
$$\text{R}\tilde{\text{E}}\text{L}((u, \text{T}, \tilde{t}, \text{T}', \tilde{t}')) = \tilde{t}'$$

Table 5.4: Definition of S$\tilde{\text{T}}$T, O$\tilde{\text{W}}$N, D$\tilde{\text{L}}$, PO$\tilde{\text{W}}$N and R$\tilde{\text{E}}$L – abstract versions of STT, OWN, DL, POWN and REL.

The partial order, $\tilde{\sqsubseteq}_{lock}$, greatest lower bound, $\tilde{\sqcap}_{lock}$, and least upper bound, $\tilde{\bigsqcup}_{lock}$, for abstract lock states follow naturally from Definitions 3.26, 3.27 and 3.28 and are presented in Definitions 5.30, 5.31 and 5.32, respectively.

Definition 5.30 (Partial order of abstract lock states):

$$\begin{cases} \tilde{\mathbb{l}} \;\tilde{\sqsubseteq}_{lock}\; \tilde{\top}_{lock} \\ \tilde{\bot}_{lock} \;\tilde{\sqsubseteq}_{lock}\; \tilde{\mathbb{l}} \\ \tilde{\mathbb{l}} \;\tilde{\sqsubseteq}_{lock}\; \tilde{\mathbb{l}}' \iff \forall lck \in \mathbf{Lck} : (\tilde{\text{ST}}(\tilde{\mathbb{l}}\ lck) = \tilde{\text{ST}}(\tilde{\mathbb{l}}'\ lck) \wedge \\ \qquad\qquad\qquad\qquad \tilde{\text{OWN}}(\tilde{\mathbb{l}}\ lck) = \tilde{\text{OWN}}(\tilde{\mathbb{l}}'\ lck) \wedge \\ \qquad\qquad\qquad\qquad \tilde{\text{DL}}(\tilde{\mathbb{l}}\ lck) \;\tilde{\sqsubseteq}_t\; \tilde{\text{DL}}(\tilde{\mathbb{l}}'\ lck) \wedge \\ \qquad\qquad\qquad\qquad \tilde{\text{POWN}}(\tilde{\mathbb{l}}\ lck) = \tilde{\text{POWN}}(\tilde{\mathbb{l}}'\ lck) \wedge \\ \qquad\qquad\qquad\qquad \tilde{\text{REL}}(\tilde{\mathbb{l}}\ lck) \;\tilde{\sqsubseteq}_t\; \tilde{\text{REL}}(\tilde{\mathbb{l}}'\ lck)) \end{cases}$$

□

Definition 5.31 (Greatest lower bound of abstract lock states):

$$\begin{cases} \tilde{\mathbb{l}} \;\tilde{\sqcap}_{lock}\; \tilde{\top}_{lock} = \tilde{\top}_{lock} \;\tilde{\sqcap}_{lock}\; \tilde{\mathbb{l}} = \tilde{\mathbb{l}} \\ \tilde{\mathbb{l}} \;\tilde{\sqcap}_{lock}\; \tilde{\bot}_{lock} = \tilde{\bot}_{lock} \;\tilde{\sqcap}_{lock}\; \tilde{\mathbb{l}} = \tilde{\bot}_{lock} \\ \lambda lck \in \mathbf{Lck}.(u_1^{lck}, \mathrm{T}_1^{lck}, \tilde{t}_1^{lck}, {\mathrm{T}'_1}^{lck}, \tilde{t}'^{lck}_1) \;\tilde{\sqcap}_{lock}\; \lambda lck \in \mathbf{Lck}.(u_2^{lck}, \mathrm{T}_2^{lck}, \tilde{t}_2^{lck}, {\mathrm{T}'_2}^{lck}, \tilde{t}'^{lck}_2) = \\ \quad \begin{cases} \lambda lck \in \mathbf{Lck}. & \textbf{if } \forall lck \in \mathbf{Lck} : \\ \quad (u_1^{lck}, \mathrm{T}_1^{lck}, \tilde{t}_1^{lck} \;\tilde{\sqcap}_t\; \tilde{t}_2^{lck}, {\mathrm{T}'_1}^{lck}, \tilde{t}'^{lck}_1 \;\tilde{\sqcap}_t\; \tilde{t}'^{lck}_2) & \quad (u_1^{lck} = u_2^{lck} \wedge \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad & \qquad \mathrm{T}_1^{lck} = \mathrm{T}_2^{lck} \wedge \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad & \qquad {\mathrm{T}'_1}^{lck} = {\mathrm{T}'_2}^{lck}) \\ \tilde{\bot}_{lock} & \textbf{otherwise} \end{cases} \end{cases}$$

□

Definition 5.32 (Least upper bound of abstract lock states):

$$
\begin{cases}
\tilde{\mathbb{I}} \,\tilde{\sqcup}_{lock}\, \tilde{\top}_{lock} = \tilde{\top}_{lock} \,\tilde{\sqcup}_{lock}\, \tilde{\mathbb{I}} = \tilde{\top}_{lock} \\
\tilde{\mathbb{I}} \,\tilde{\sqcup}_{lock}\, \tilde{\bot}_{lock} = \tilde{\bot}_{lock} \,\tilde{\sqcup}_{lock}\, \tilde{\mathbb{I}} = \tilde{\mathbb{I}} \\
\lambda\, lck \in \mathbf{Lck}.(u_1^{lck}, \mathrm{T}_1^{lck}, \tilde{t}_1^{lck}, \mathrm{T}_1'^{\,lck}, \tilde{t}_1'^{\,lck}) \,\tilde{\sqcup}_{lock} \\
\qquad\qquad \lambda\, lck \in \mathbf{Lck}.(u_2^{lck}, \mathrm{T}_2^{lck}, \tilde{t}_2^{lck}, \mathrm{T}_2'^{\,lck}, \tilde{t}_2'^{\,lck}) = \\
\quad \begin{cases}
\lambda\, lck \in \mathbf{Lck}. & \textbf{if } \forall lck \in \mathbf{Lck}: \\
\quad (u_1^{lck}, \mathrm{T}_1^{lck}, \tilde{t}_1^{lck} \,\tilde{\sqcup}_t\, \tilde{t}_2^{lck}, \mathrm{T}_1'^{\,lck}, \tilde{t}_1'^{\,lck} \,\tilde{\sqcup}_t\, \tilde{t}_2'^{\,lck}) & \quad (u_1^{lck} = u_2^{lck} \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad & \quad \mathrm{T}_1^{lck} = \mathrm{T}_2^{lck} \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad & \quad \mathrm{T}_1'^{\,lck} = \mathrm{T}_2'^{\,lck}) \\
\tilde{\top}_{lock} & \textbf{otherwise}
\end{cases}
\end{cases}
$$

□

Lemma 5.33 (Monotonicity of γ_{lock}):
γ_{lock}, as given by Definition 5.28, is monotone. □

PROOF. Assume that $\tilde{\mathbb{I}} \,\tilde{\sqsubseteq}_{lock}\, \tilde{\mathbb{I}}'$. If $\tilde{\mathbb{I}} = \tilde{\bot}_{lock}$ or $\tilde{\mathbb{I}}' = \tilde{\top}_{lock}$, then trivially, $\gamma_{lock}(\tilde{\mathbb{I}}) \subseteq \gamma_{lock}(\tilde{\mathbb{I}}')$. Otherwise, assume that $\mathbb{I} \in \gamma_{lock}(\tilde{\mathbb{I}})$, $\tilde{\mathbb{I}}\, lck = (u_{lck}, \mathrm{T}_{lck}, \tilde{t}_{lck}, \mathrm{T}'_{lck}, \tilde{t}'_{lck})$ and $\tilde{\mathbb{I}}'\, lck = (u'_{lck}, \mathrm{T}''_{lck}, \tilde{t}''_{lck}, \mathrm{T}'''_{lck}, \tilde{t}'''_{lck})$. Since $\tilde{\mathbb{I}} \,\tilde{\sqsubseteq}_{lock}\, \tilde{\mathbb{I}}'$, it must be that $\forall lck \in \mathbf{Lck} : (u_{lck} = u'_{lck} \wedge \mathrm{T}_{lck} = \mathrm{T}''_{lck} \wedge \mathrm{T}'_{lck} = \mathrm{T}'''_{lck} \wedge \tilde{t}_{lck} \,\tilde{\sqsubseteq}_t\, \tilde{t}''_{lck} \wedge \tilde{t}'_{lck} \,\tilde{\sqsubseteq}_t\, \tilde{t}'''_{lck})$. But then, since $\mathbb{I} \in \gamma_{lock}(\tilde{\mathbb{I}})$ and γ_t is monotone (Theorem 3.39), it must be that $\mathbb{I} \in \gamma_{lock}(\tilde{\mathbb{I}}')$. This proves the lemma. ■

Theorem 5.34 (Galois connection – Lock states):
$\langle \alpha_{lock}, \gamma_{lock} \rangle$, where γ_{lock} and α_{lock} are given by Definitions 5.28 and 5.29, respectively, is a Galois connection. □

PROOF. First it will be shown that γ_{lock} is completely multiplicative. Thus note that γ_{lock} is monotone (Lemma 5.33). Next observe that $\gamma_{lock}(\tilde{\top}_{lock}) = \mathbf{Lck} \to (\mathbf{Lck_{stt}} \times \mathbf{Thrd}_\bot \times \mathbf{Time} \times \mathbf{Thrd}_\bot \times \mathbf{Time}) = \top_{lock}$.

Now, assume that $\tilde{\mathbb{I}}, \tilde{\mathbb{I}}' \in \mathbf{Lck} \to (\mathbf{Lck_{stt}} \times \mathbf{Thrd}_\bot \times \tilde{\mathbf{Time}} \times \mathbf{Thrd}_\bot \times \tilde{\mathbf{Time}})$ are such that $\tilde{\mathbb{I}} \not\tilde{\sqsubseteq}_{lock} \tilde{\mathbb{I}}' \wedge \tilde{\mathbb{I}}' \not\tilde{\sqsubseteq}_{lock} \tilde{\mathbb{I}}$. From Definition 5.30, it follows that neither of $\tilde{\mathbb{I}}$ and $\tilde{\mathbb{I}}'$ can be $\tilde{\bot}_{lock}$ or $\tilde{\top}_{lock}$. Thus, it is safe to assume that these states can be expressed as $\tilde{\mathbb{I}}\, lck = (u_{lck}, \mathrm{T}_{lck}, \tilde{t}_{lck}, \mathrm{T}'_{lck}, \tilde{t}'_{lck})$ and $\tilde{\mathbb{I}}'\, lck = (u'_{lck}, \mathrm{T}''_{lck}, \tilde{t}''_{lck}, \mathrm{T}'''_{lck}, \tilde{t}'''_{lck})$.

Based on the above assumptions, it will be shown that:

$$\gamma_{lock}(\tilde{\mathbb{I}} \,\tilde{\sqcap}_{lock}\, \tilde{\mathbb{I}}') = \gamma_{lock}(\tilde{\mathbb{I}}) \cap \gamma_{lock}(\tilde{\mathbb{I}}')$$

First, assume that $\exists lck \in \mathbf{Lck} : (u_{lck} \neq u'_{lck} \vee \mathrm{T}_{lck} \neq \mathrm{T}''_{lck} \vee \mathrm{T}'_{lck} \neq \mathrm{T}'''_{lck})$. Then, $\tilde{\mathbb{I}} \tilde{\sqcap}_{lock} \tilde{\mathbb{I}}' = \tilde{\bot}_{lock}$, and thus the L.H.S. becomes $\gamma_{lock}(\tilde{\mathbb{I}} \tilde{\sqcap}_{lock} \tilde{\mathbb{I}}') = \gamma_{lock}(\tilde{\bot}_{lock}) = \emptyset$. The R.H.S. becomes $\gamma_{lock}(\tilde{\mathbb{I}}) \cap \gamma_{lock}(\tilde{\mathbb{I}}') = \emptyset$, because it must be that $\forall \mathbb{I} \in \gamma_{lock}(\tilde{\mathbb{I}}) : \forall \mathbb{I}' \in \gamma_{lock}(\tilde{\mathbb{I}}') : \mathbb{I} \neq \mathbb{I}'$ since $\exists lck \in \mathbf{Lck} : (u_{lck} \neq u'_{lck} \vee \mathrm{T}_{lck} \neq \mathrm{T}''_{lck} \vee \mathrm{T}'_{lck} \neq \mathrm{T}'''_{lck})$. Thus, L.H.S. = R.H.S.

Next, assume that $\forall lck \in \mathbf{Lck} : (u_{lck} = u'_{lck} \wedge \mathrm{T}_{lck} = \mathrm{T}''_{lck} \wedge \mathrm{T}'_{lck} = \mathrm{T}'''_{lck})$ and note that $\langle \alpha_t, \gamma_t \rangle = \langle \alpha_{int}, \gamma_{int} \rangle$ is a Galois connection (Theorem 3.39). Then, $(\tilde{\mathbb{I}} \tilde{\sqcap}_{lock} \tilde{\mathbb{I}}')\ lck = (u_{lck}, \mathrm{T}_{lck}, \tilde{t}_{lck} \tilde{\sqcap}_t \tilde{t}''_{lck}, \mathrm{T}'_{lck}, \tilde{t}'_{lck} \tilde{\sqcap}_t \tilde{t}'''_{lck}))$ and

$$
\begin{aligned}
\gamma_{lock}(\tilde{\mathbb{I}} \tilde{\sqcap}_{lock} \tilde{\mathbb{I}}') &\overset{Def.\ 5.28}{=} \{\lambda lck \in \mathbf{Lck}.(u_{lck}, \mathrm{T}_{lck}, t_{lck}, \mathrm{T}'_{lck}, t'_{lck}) \mid \\
&\qquad t_{lck} \in \gamma_t(\tilde{t}_{lck} \tilde{\sqcap}_t \tilde{t}''_{lck}) \wedge t'_{lck} \in \gamma_t(\tilde{t}'_{lck} \tilde{\sqcap}_t \tilde{t}'''_{lck})\} \\
&\overset{Lem.\ 3.14}{=} \{\lambda lck \in \mathbf{Lck}.(u_{lck}, \mathrm{T}_{lck}, t_{lck}, \mathrm{T}'_{lck}, t'_{lck}) \mid \\
&\qquad t_{lck} \in \gamma_t(\tilde{t}_{lck}) \cap \gamma_t(\tilde{t}''_{lck}) \wedge t'_{lck} \in \gamma_t(\tilde{t}'_{lck}) \cap \gamma_t(\tilde{t}'''_{lck})\} \\
&\overset{calc.}{=} \{\lambda lck \in \mathbf{Lck}.(u_{lck}, \mathrm{T}_{lck}, t_{lck}, \mathrm{T}'_{lck}, t'_{lck}) \mid \\
&\qquad t_{lck} \in \gamma_t(\tilde{t}_{lck}) \wedge t'_{lck} \in \gamma_t(\tilde{t}'_{lck})\} \cap \\
&\quad \{\lambda lck \in \mathbf{Lck}.(u_{lck}, \mathrm{T}_{lck}, t_{lck}, \mathrm{T}'_{lck}, t'_{lck}) \mid \\
&\qquad t_{lck} \in \gamma_t(\tilde{t}''_{lck}) \wedge t_{lck} \in \gamma_t(\tilde{t}'''_{lck})\} \\
&\overset{Def.\ 5.28}{=} \gamma_{lock}(\tilde{\mathbb{I}}) \cap \gamma_{lock}(\tilde{\mathbb{I}}')
\end{aligned}
$$

Thus, it has been shown that $\gamma_{lock}(\tilde{\mathbb{I}} \tilde{\sqcap}_{lock} \tilde{\mathbb{I}}') = \gamma_{lock}(\tilde{\mathbb{I}}) \cap \gamma_{lock}(\tilde{\mathbb{I}}')$. Now, all the three conditions in Lemma 3.4 are fulfilled, which means that γ_{lock} is completely multiplicative. Then, by Lemma 3.15, it is obvious that an abstraction function, α, such that $\langle \alpha, \gamma_{lock} \rangle$ is a Galois connection can be defined. Using Lemma 3.14, the definition of this α is the same as that of α_{lock} in Definition 5.29. Thus, $\langle \alpha_{lock}, \gamma_{lock} \rangle$ is a Galois connection. ∎

5.7 Abstract Configurations

In this section, a Galois connection (c.f., Theorem 5.41) between the concrete and abstract domains for configurations, $\mathscr{P}(\mathbf{Conf})$ and $\tilde{\mathbf{Conf}}$, respectively, will be defined. $\tilde{\mathbf{Conf}}$ is defined as:

$$
\begin{aligned}
\tilde{\mathbf{Conf}} = (&\mathscr{P}_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}(\{\mathrm{T}\} \times \mathbf{Lbl}_{\mathrm{T}} \times (\mathbf{Reg}_{\mathrm{T}} \to \tilde{\mathbf{Val}}) \times \tilde{\mathbf{Time}}) \times \\
&(\mathbf{Var} \to \mathbf{Thrd} \to \mathscr{P}(\tilde{\mathbf{Val}} \times \tilde{\mathbf{Time}})) \times \\
&(\mathbf{Lck} \to (\mathbf{Lck}_{\mathbf{stt}} \times \mathbf{Thrd}_{\bot} \times \tilde{\mathbf{Time}} \times \mathbf{Thrd}_{\bot} \times \tilde{\mathbf{Time}}))) \cup \\
&\{\tilde{\bot}_{conf}, \tilde{\top}_{conf}\}
\end{aligned}
$$

where $\mathbf{Thrd}_{\tilde{c}} \subseteq \mathbf{Thrd}$ (the reason for this will become apparent when the analysis is presented in Chapter 6). The abstract configuration, $\tilde{c} \in \tilde{\mathbf{Conf}}$, will be

denoted in the same manner as concrete configurations:

$$\tilde{c} ::= \langle [\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle$$

The concretization function for abstract configurations, $\gamma_{conf} : \tilde{\mathbf{Conf}} \to \mathscr{P}(\mathbf{Conf})$, is given by Definition 5.35.

Definition 5.35 (Concretization of an abstract configuration):

$$\begin{cases} \gamma_{conf}(\tilde{\top}_{conf}) = \mathbf{Conf} \\ \gamma_{conf}(\tilde{\bot}_{conf}) = \emptyset \\ \gamma_{conf}(\langle [\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle) = \\ \quad \{ \langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \mathbb{x}, \mathbb{l} \rangle \mid \\ \qquad\qquad \mathbb{r}_{\mathrm{T}} \in \gamma_{reg}(\tilde{\mathbb{r}}_{\mathrm{T}}) \wedge t^{a}_{\mathrm{T}} \in \gamma_{t}(\tilde{t}^{a}_{\mathrm{T}}) \wedge \mathbb{x} \in \gamma_{var}(\tilde{\mathbb{x}}) \wedge \mathbb{l} \in \gamma_{lock}(\tilde{\mathbb{l}}) \} \end{cases} \quad \square$$

The partial ordering of abstract configurations, $\tilde{\sqsubseteq}_{conf}$, follows naturally using Definition 3.26 and is given by Definition 5.36. Note that this relation cannot be directly used within the analysis since $\tilde{\sqsubseteq}_{var}$ cannot. A safe relation, $\tilde{\sqsubseteq}'_{conf}$, is obtained by replacing $\tilde{\sqsubseteq}_{var}$ with $\tilde{\sqsubseteq}'_{var}$ in the definition of $\tilde{\sqsubseteq}_{conf}$.

Definition 5.36 (Partial ordering of two abstract configurations):

$$\begin{cases} \tilde{c} \,\tilde{\sqsubseteq}_{conf}\, \tilde{\top}_{conf} \\ \tilde{\bot}_{conf} \,\tilde{\sqsubseteq}_{conf}\, \tilde{c} \\ \langle [\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle \,\tilde{\sqsubseteq}_{conf} \\ \qquad\qquad \langle [\mathrm{T}, pc'_{\mathrm{T}}, \tilde{\mathbb{r}}'_{\mathrm{T}}, \tilde{t}^{a\prime}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}'}}, \tilde{\mathbb{x}}', \tilde{\mathbb{l}}' \rangle \\ \qquad\qquad \Longleftrightarrow \\ \quad \tilde{\mathbb{x}} \,\tilde{\sqsubseteq}_{var}\, \tilde{\mathbb{x}}' \wedge \tilde{\mathbb{l}} \,\tilde{\sqsubseteq}_{lock}\, \tilde{\mathbb{l}}' \wedge \mathbf{Thrd}_{\tilde{c}} = \mathbf{Thrd}_{\tilde{c}'} \wedge \\ \quad \forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : (pc_{\mathrm{T}} = pc'_{\mathrm{T}} \wedge \tilde{\mathbb{r}}_{\mathrm{T}} \,\tilde{\sqsubseteq}_{reg}\, \tilde{\mathbb{r}}'_{\mathrm{T}} \wedge \tilde{t}^{a}_{\mathrm{T}} \,\tilde{\sqsubseteq}_{t}\, \tilde{t}^{a\prime}_{\mathrm{T}}) \end{cases} \quad \square$$

The function γ_{conf} is monotone with respect to $\tilde{\sqsubseteq}_{conf}$ (Lemma 5.37).

Lemma 5.37 (Monotonicity of γ_{conf}):
The function $\gamma_{conf} : \tilde{\mathbf{Conf}} \to \mathscr{P}(\mathbf{Conf})$ is monotone with respect to $\tilde{\sqsubseteq}_{conf}$. I.e., if $\tilde{c}, \tilde{c}' \in \tilde{\mathbf{Conf}}$ and $\tilde{c} \,\tilde{\sqsubseteq}_{conf}\, \tilde{c}'$, then $\gamma_{conf}(\tilde{c}) \subseteq \gamma_{conf}(\tilde{c}')$. $\square$

PROOF. Assume that $\tilde{c}, \tilde{c}' \in \tilde{\mathbf{Conf}}$ such that $\tilde{c} \,\tilde{\sqsubseteq}_{conf}\, \tilde{c}'$. If $\tilde{c} = \tilde{\bot}_{conf}$ or $\tilde{c}' = \tilde{\top}_{conf}$, the lemma holds trivially. Otherwise, $\tilde{c}$ and $\tilde{c}'$ can be expressed as $\tilde{c} =$

$\langle [\mathrm{T}, pc_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle$ and $\tilde{c}' = \langle [\mathrm{T}, pc'_\mathrm{T}, \tilde{\mathbb{r}}'_\mathrm{T}, \tilde{t}^{a\prime}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}'}}, \tilde{\mathbb{x}}', \tilde{\mathbb{l}}' \rangle$. Assume that $c \in \gamma_{conf}(\tilde{c})$. Since $\tilde{c} \,\tilde{\sqsubseteq}_{conf}\, \tilde{c}'$, it must be that:

$$\begin{array}{c} \mathbf{Thrd}_{\tilde{c}} = \mathbf{Thrd}_{\tilde{c}'} \wedge \tilde{\mathbb{l}} \,\tilde{\sqsubseteq}_{lock}\, \tilde{\mathbb{l}}' \wedge \tilde{\mathbb{x}} \,\tilde{\sqsubseteq}_{var}\, \tilde{\mathbb{x}}' \wedge \\ \forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : (pc_\mathrm{T} = pc'_\mathrm{T} \wedge \tilde{\mathbb{r}}_\mathrm{T} \,\tilde{\sqsubseteq}_{reg}\, \tilde{\mathbb{r}}'_\mathrm{T} \wedge \tilde{t}^a_\mathrm{T} \,\tilde{\sqsubseteq}_{t}\, \tilde{t}^{a\prime}_\mathrm{T}) \end{array}$$

The monotonicity of γ_t, γ_{reg}, γ_{var} and γ_{lock} (Theorems 3.39, 5.6 and 5.10, and Lemma 5.33, respectively) then implies that $c \in \gamma_{conf}(\tilde{c}')$ as well. Thus, $\gamma_{conf}(\tilde{c}) \subseteq \gamma_{conf}(\tilde{c}')$ and the lemma holds. ■

The greatest lower bound operator for abstract configurations, $\tilde{\sqcap}_{conf}$, follows naturally using Definition 3.27 and is given by Definition 5.38. Note that this operator cannot be directly used within the analysis since $\tilde{\sqcap}_{var}$ cannot. A safe operator, $\tilde{\sqcap}'_{conf}$, is obtained by replacing $\tilde{\sqcap}_{var}$ by $\tilde{\sqcap}'_{var}$ in the definition of $\tilde{\sqcap}_{conf}$.

Definition 5.38 (Greatest lower bound for two abstract configurations):

$$\begin{cases} \tilde{c} \,\tilde{\sqcap}_{conf}\, \tilde{\top}_{conf} = \tilde{\top}_{conf} \,\tilde{\sqcap}_{conf}\, \tilde{c} = \tilde{c} \\ \tilde{c} \,\tilde{\sqcap}_{conf}\, \tilde{\bot}_{conf} = \tilde{\bot}_{conf} \,\tilde{\sqcap}_{conf}\, \tilde{c} = \tilde{\bot}_{conf} \\ \langle [\mathrm{T}, pc_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle \,\tilde{\sqcap}_{conf} \\ \qquad \langle [\mathrm{T}, pc'_\mathrm{T}, \tilde{\mathbb{r}}'_\mathrm{T}, \tilde{t}^{a\prime}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}'}}, \tilde{\mathbb{x}}', \tilde{\mathbb{l}}' \rangle = \\ \quad \begin{cases} \langle [\mathrm{T}, pc_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T} \,\tilde{\sqcap}_{reg}\, \tilde{\mathbb{r}}'_\mathrm{T}, & \textbf{if } \mathbf{Thrd}_{\tilde{c}} = \mathbf{Thrd}_{\tilde{c}'} \wedge \\ \quad \tilde{t}^a_\mathrm{T} \,\tilde{\sqcap}_{t}\, \tilde{t}^{a\prime}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, & \quad \forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : pc_\mathrm{T} = pc'_\mathrm{T} \\ \quad \tilde{\mathbb{x}} \,\tilde{\sqcap}_{var}\, \tilde{\mathbb{x}}', \tilde{\mathbb{l}} \,\tilde{\sqcap}_{lock}\, \tilde{\mathbb{l}}' \rangle & \\ \tilde{\bot}_{conf} & \textbf{otherwise} \end{cases} \end{cases} \qquad \square$$

The least upper bound operator for abstract configurations, $\tilde{\sqcup}_{conf}$, follows naturally using Definition 3.28 and is given by Definition 5.39. Note that this operator cannot be directly used within the analysis since $\tilde{\sqcup}_{var}$ cannot. A safe operator, $\tilde{\sqcup}'_{conf}$, is obtained by replacing $\tilde{\sqcup}_{var}$ by $\tilde{\sqcup}'_{var}$ in the definition of $\tilde{\sqcup}_{conf}$.

Definition 5.39 (Least upper bound for two abstract configurations):

$$
\begin{cases}
\tilde{c} \mathbin{\tilde{\sqcup}_{conf}} \tilde{\top}_{conf} = \tilde{\top}_{conf} \mathbin{\tilde{\sqcup}_{conf}} \tilde{c} = \tilde{\top}_{conf} \\
\tilde{c} \mathbin{\tilde{\sqcup}_{conf}} \tilde{\bot}_{conf} = \tilde{\bot}_{conf} \mathbin{\tilde{\sqcup}_{conf}} \tilde{c} = \tilde{c} \\
\langle [\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle \mathbin{\tilde{\sqcup}_{conf}} \\
\qquad \langle [\mathrm{T}, pc'_{\mathrm{T}}, \tilde{\mathbb{r}}'_{\mathrm{T}}, \tilde{t}^{a\prime}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}'}}, \tilde{\mathbb{x}}', \tilde{\mathbb{l}}' \rangle = \\
\quad \begin{cases}
\langle [\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}} \mathbin{\tilde{\sqcup}_{reg}} \tilde{\mathbb{r}}'_{\mathrm{T}}, & \textbf{if } \mathbf{Thrd}_{\tilde{c}} = \mathbf{Thrd}_{\tilde{c}'} \wedge \\
\quad \tilde{t}^{a}_{\mathrm{T}} \mathbin{\tilde{\sqcup}_{t}} \tilde{t}^{a\prime}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, & \quad \forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : pc_{\mathrm{T}} = pc'_{\mathrm{T}} \\
\quad \tilde{\mathbb{x}} \mathbin{\tilde{\sqcup}_{var}} \tilde{\mathbb{x}}', \tilde{\mathbb{l}} \mathbin{\tilde{\sqcup}_{lock}} \tilde{\mathbb{l}}' \rangle & \\
\tilde{\top}_{conf} & \textbf{otherwise}
\end{cases}
\end{cases}
$$

□

The abstraction function, $\alpha_{conf} : \mathscr{P}(\mathbf{Conf}) \to \mathbf{C\tilde{o}nf}$, is given by Definition 5.40 and $\langle \alpha_{conf}, \gamma_{conf} \rangle$ is a Galois connection (Theorem 5.41).

Definition 5.40 (Abstraction of a set of configurations):

$$\alpha_{conf}(C) = \tilde{\bigsqcap}_{conf} \{ \tilde{c} \mid C \subseteq \gamma_{conf}(\tilde{c}) \}$$

□

Theorem 5.41 (Galois connection – Configurations):
$\langle \alpha_{conf}, \gamma_{conf} \rangle$, where γ_{conf} and α_{conf} are given by Definitions 5.35 and 5.40, respectively, is a Galois connection. □

PROOF. First it will be shown that γ_{conf} is completely multiplicative. Thus note that γ_{conf} is monotone (Lemma 5.37). Next observe that $\gamma_{conf}(\tilde{\top}_{conf}) = \mathbf{Conf} = \top_{conf}$.

Now, assume that $\tilde{c}, \tilde{c}' \in \mathbf{C\tilde{o}nf}$ are such that $\tilde{c} \not\tilde{\sqsubseteq}_{conf} \tilde{c}' \wedge \tilde{c}' \not\tilde{\sqsubseteq}_{conf} \tilde{c}$. From Definition 5.36, it follows that neither of $\tilde{c}$ and $\tilde{c}'$ can be $\tilde{\bot}_{conf}$ or $\tilde{\top}_{conf}$. Thus, it is safe to assume that these configurations can be expressed as $\tilde{c} = \langle [\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle$ and $\tilde{c}' = \langle [\mathrm{T}, pc'_{\mathrm{T}}, \tilde{\mathbb{r}}'_{\mathrm{T}}, \tilde{t}^{a\prime}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}'}}, \tilde{\mathbb{x}}', \tilde{\mathbb{l}}' \rangle$.

Based on the above assumptions, it will be shown that:

$$\gamma_{conf}(\tilde{c} \mathbin{\tilde{\sqcap}_{conf}} \tilde{c}') = \gamma_{conf}(\tilde{c}) \cap \gamma_{conf}(\tilde{c}')$$

First, assume that $\mathbf{Thrd}_{\tilde{c}} \neq \mathbf{Thrd}_{\tilde{c}'} \vee \exists \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : pc_{\mathrm{T}} \neq pc'_{\mathrm{T}}$. Then, $\tilde{c} \mathbin{\tilde{\sqcap}_{conf}} \tilde{c}' = \tilde{\bot}_{conf}$, and thus the L.H.S. becomes $\gamma_{conf}(\tilde{c} \mathbin{\tilde{\sqcap}_{conf}} \tilde{c}') = \gamma_{conf}(\tilde{\bot}_{conf}) = \emptyset$. The R.H.S. becomes $\gamma_{conf}(\tilde{c}) \cap \gamma_{conf}(\tilde{c}') = \emptyset$, because it must be that $\forall c \in \gamma_{conf}(\tilde{c}) : \forall c' \in \gamma_{conf}(\tilde{c}') : c \neq c'$, since $\mathbf{Thrd}_{\tilde{c}} \neq \mathbf{Thrd}_{\tilde{c}'} \vee \exists \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : pc_{\mathrm{T}} \neq pc'_{\mathrm{T}}$. Thus, L.H.S. = R.H.S.

Next, assume that $\mathbf{Thrd}_{\tilde{c}} = \mathbf{Thrd}_{\tilde{c}'} \wedge \forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : pc_{\mathrm{T}} = pc'_{\mathrm{T}}$ and note that $\langle \alpha_t, \gamma_t \rangle = \langle \alpha_{int}, \gamma_{int} \rangle$, $\langle \alpha_{reg}, \gamma_{reg} \rangle$, $\langle \alpha_{var}, \gamma_{var} \rangle$ and $\langle \alpha_{lock}, \gamma_{lock} \rangle$ are Galois connections (Theorems 3.39, 5.6, 5.10 and 5.34, respectively). Then, $\tilde{c} \tilde{\sqcap}_{conf} \tilde{c}' = \langle [\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}} \tilde{\sqcap}_{reg} \tilde{\mathbb{r}}'_{\mathrm{T}}, \tilde{t}^a_{\mathrm{T}} \tilde{\sqcap}_t \tilde{t}^{a\prime}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}} \tilde{\sqcap}_{var} \tilde{\mathbb{x}}', \tilde{\mathbb{l}} \tilde{\sqcap}_{lock} \tilde{\mathbb{l}}' \rangle$ and

$$
\begin{aligned}
\gamma_{conf}(\tilde{c} \tilde{\sqcap}_{conf} \tilde{c}') &\overset{Def.\ 5.35}{=} \{\langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \mathbb{x}, \mathbb{l} \rangle \mid \\
&\qquad \mathbb{r}_{\mathrm{T}} \in \gamma_{reg}(\tilde{\mathbb{r}}_{\mathrm{T}} \tilde{\sqcap}_{reg} \tilde{\mathbb{r}}'_{\mathrm{T}}) \wedge t^a_{\mathrm{T}} \in \gamma_t(\tilde{t}^a_{\mathrm{T}} \tilde{\sqcap}_t \tilde{t}^{a\prime}_{\mathrm{T}}) \wedge \\
&\qquad\quad \mathbb{x} \in \gamma_{var}(\tilde{\mathbb{x}} \tilde{\sqcap}_{var} \tilde{\mathbb{x}}') \wedge \mathbb{l} \in \gamma_{lock}(\tilde{\mathbb{l}} \tilde{\sqcap}_{lock} \tilde{\mathbb{l}}')\} \\
&\overset{Lem.\ 3.14}{=} \{\langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \mathbb{x}, \mathbb{l} \rangle \mid \\
&\qquad \mathbb{r}_{\mathrm{T}} \in \gamma_{reg}(\tilde{\mathbb{r}}_{\mathrm{T}}) \cap \gamma_{reg}(\tilde{\mathbb{r}}'_{\mathrm{T}}) \wedge t^a_{\mathrm{T}} \in \gamma_t(\tilde{t}^a_{\mathrm{T}}) \cap \gamma_t(\tilde{t}^{a\prime}_{\mathrm{T}}) \wedge \\
&\qquad\quad \mathbb{x} \in \gamma_{var}(\tilde{\mathbb{x}}) \cap \gamma_{var}(\tilde{\mathbb{x}}') \wedge \mathbb{l} \in \gamma_{lock}(\tilde{\mathbb{l}}) \cap \gamma_{lock}(\tilde{\mathbb{l}}')\} \\
&\overset{calc.}{=} \{\langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \mathbb{x}, \mathbb{l} \rangle \mid \\
&\qquad\qquad \mathbb{r}_{\mathrm{T}} \in \gamma_{reg}(\tilde{\mathbb{r}}_{\mathrm{T}}) \wedge t^a_{\mathrm{T}} \in \gamma_t(\tilde{t}^a_{\mathrm{T}}) \wedge \\
&\qquad\qquad\quad \mathbb{x} \in \gamma_{var}(\tilde{\mathbb{x}}) \wedge \mathbb{l} \in \gamma_{lock}(\tilde{\mathbb{l}})\} \cap \\
&\qquad \{\langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \mathbb{x}, \mathbb{l} \rangle \mid \\
&\qquad\qquad \mathbb{r}_{\mathrm{T}} \in \gamma_{reg}(\tilde{\mathbb{r}}'_{\mathrm{T}}) \wedge t^a_{\mathrm{T}} \in \gamma_t(\tilde{t}^{a\prime}_{\mathrm{T}}) \wedge \\
&\qquad\qquad\quad \mathbb{x} \in \gamma_{var}(\tilde{\mathbb{x}}') \wedge \mathbb{l} \in \gamma_{lock}(\tilde{\mathbb{l}}')\} \\
&\overset{Def.\ 5.35}{=} \gamma_{conf}(\tilde{c}) \cap \gamma_{conf}(\tilde{c}')
\end{aligned}
$$

Thus, it has been shown that $\gamma_{conf}(\tilde{c} \tilde{\sqcap}_{conf} \tilde{c}') = \gamma_{conf}(\tilde{c}) \cap \gamma_{conf}(\tilde{c}')$. Now, all the three conditions in Lemma 3.4 are fulfilled, which means that γ_{conf} is completely multiplicative. Then, by Lemma 3.15, it is obvious that an abstraction function, α, such that $\langle \alpha, \gamma_{conf} \rangle$ is a Galois connection can be defined. Using Lemma 3.14, the definition of this α is the same as that of α_{conf} in Definition 5.40. Thus, $\langle \alpha_{conf}, \gamma_{conf} \rangle$ is a Galois connection. ∎

An alternative approach to derive a Galois connection here could be to use Theorems 3.16, 3.17, 3.20, 3.22, 3.24, 3.25 and 3.39, but the presented Galois connection is easier to understand.

Now, consider the abstract domains, $\mathbf{Cõnf}^{\mathbf{ax}}_{\mathbf{in}} \ni \tilde{c}^{ax}_{in}$ and $\mathbf{Cõnf}^{\mathbf{ax}}_{\mathbf{out}} \ni \tilde{c}^{ax}_{out}$, which will be used for the abstract axiom transition rules in Table 5.5. These domains are defined as:

$$
\begin{aligned}
\mathbf{Cõnf}^{\mathbf{ax}}_{\mathbf{in}} = (&\mathbf{Thrd} \times \mathbf{Lbl} \times (\mathbf{Reg} \to \mathbf{Vãl}) \times \\
&(\mathbf{Var} \to \mathbf{Thrd} \to \mathscr{P}(\mathbf{Vãl} \times \mathbf{Timẽ})) \times \\
&(\mathbf{Lck} \to (\mathbf{Lck_{stt}} \times \mathbf{Thrd}_{\perp} \times \mathbf{Timẽ} \times \mathbf{Thrd}_{\perp} \times \mathbf{Timẽ})) \times \\
&\mathbf{Timẽ}) \cup \{\tilde{\perp}^{ax}_{in}, \tilde{\top}^{ax}_{in}\}
\end{aligned}
$$

$$
\tilde{c}^{ax}_{in} ::= \langle \mathrm{T}, pc, \tilde{\mathbb{r}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}, \tilde{t} \rangle
$$

and:

$$\begin{aligned}\mathbf{C\tilde{o}nf^{ax}_{out}} = (&\mathbf{Lbl} \times (\mathbf{Reg} \to \mathbf{V\tilde{a}l}) \times \\ &(\mathbf{Var} \to \mathbf{Thrd} \to \mathscr{P}(\mathbf{V\tilde{a}l} \times \mathbf{Ti\tilde{m}e})) \times \\ &(\mathbf{Lck} \to (\mathbf{Lck_{stt}} \times \mathbf{Thrd}_\perp \times \mathbf{Ti\tilde{m}e} \times \mathbf{Thrd}_\perp \times \mathbf{Ti\tilde{m}e}))) \cup \\ &\{\tilde{\perp}^{ax}_{out}, \tilde{\top}^{ax}_{out}\}\end{aligned}$$

$$\tilde{c}^{ax}_{out} ::= \langle pc, \tilde{\mathtt{r}}, \tilde{\mathtt{x}}, \tilde{\mathtt{l}} \rangle$$

It is easy to see that $\langle \alpha^{ax}_{in}, \gamma^{ax}_{in} \rangle$ and $\langle \alpha^{ax}_{out}, \gamma^{ax}_{out} \rangle$, where $\alpha^{ax}_{in} : \mathscr{P}(\mathbf{Conf^{ax}_{in}}) \to \mathbf{C\tilde{o}nf^{ax}_{in}}$, $\gamma^{ax}_{in} : \mathbf{C\tilde{o}nf^{ax}_{in}} \to \mathscr{P}(\mathbf{Conf^{ax}_{in}})$, $\alpha^{ax}_{out} : \mathscr{P}(\mathbf{Conf^{ax}_{out}}) \to \mathbf{C\tilde{o}nf^{ax}_{out}}$ and $\gamma^{ax}_{out} : \mathbf{C\tilde{o}nf^{ax}_{out}} \to \mathscr{P}(\mathbf{Conf^{ax}_{out}})$ are given by Definitions 5.42, 5.43, 5.44 and 5.45, respectively, are Galois connections (c.f., Theorems 5.46 and 5.47).

Definition 5.42 (Abstraction of a set of axiom input configurations):

$$\alpha^{ax}_{in}(C^{ax}_{in}) = \tilde{\bigsqcap}^{ax}_{in}\{\tilde{c}^{ax}_{in} \mid C^{ax}_{in} \subseteq \gamma^{ax}_{in}(\tilde{c}^{ax}_{in})\}$$ □

Definition 5.43 (Concretization of an abstract axiom input configuration):

$$\begin{aligned}\gamma^{ax}_{in}(\langle \mathrm{T}, pc, \tilde{\mathtt{r}}, \tilde{\mathtt{x}}, \tilde{\mathtt{l}}, \tilde{t} \rangle) = \{\langle \mathrm{T}, pc, \mathtt{r}, \mathtt{x}, \mathtt{l}, t \rangle \mid\ & \mathtt{r} \in \gamma_{reg}(\tilde{\mathtt{r}}) \wedge \mathtt{x} \in \gamma_{var}(\tilde{\mathtt{x}}) \wedge \\ & \mathtt{l} \in \gamma_{lock}(\tilde{\mathtt{l}}) \wedge t \in \gamma_t(\tilde{t})\}\end{aligned}$$ □

Definition 5.44 (Abstraction of a set of axiom output configurations):

$$\alpha^{ax}_{out}(C^{ax}_{out}) = \tilde{\bigsqcap}^{ax}_{out}\{\tilde{c}^{ax}_{out} \mid C^{ax}_{out} \subseteq \gamma^{ax}_{out}(\tilde{c}^{ax}_{out})\}$$ □

Definition 5.45 (Concretization of an abstract axiom output configuration):

$$\gamma^{ax}_{out}(\langle pc, \tilde{\mathtt{r}}, \tilde{\mathtt{x}}, \tilde{\mathtt{l}} \rangle) = \{\langle pc, \mathtt{r}, \mathtt{x}, \mathtt{l} \rangle \mid \mathtt{r} \in \gamma_{reg}(\tilde{\mathtt{r}}) \wedge \mathtt{x} \in \gamma_{var}(\tilde{\mathtt{x}}) \wedge \mathtt{l} \in \gamma_{lock}(\tilde{\mathtt{l}})\}$$ □

Theorem 5.46 (Galois connection – Axiom input configurations):
$\langle \alpha^{ax}_{in}, \gamma^{ax}_{in} \rangle$, where α^{ax}_{in} and γ^{ax}_{in} are given by Definitions 5.42 and 5.43, respectively, is a Galois connection. □

PROOF. Similar to the proof of Theorem 5.41. ■

Theorem 5.47 (Galois connection – Axiom output configurations):
$\langle \alpha^{ax}_{out}, \gamma^{ax}_{out} \rangle$, where α^{ax}_{out} and γ^{ax}_{out} are given by Definitions 5.44 and 5.45, respectively, is a Galois connection. □

PROOF. Similar to the proof of Theorem 5.41. ■

$\textsc{stm}(\mathrm{T}, pc)$	$\langle pc', \tilde{\mathrm{r}}', \tilde{\mathrm{x}}', \tilde{\mathrm{l}}' \rangle$	If
$[\texttt{halt}]^{pc}$	$\langle pc, \tilde{\mathrm{r}}, \tilde{\mathrm{x}}, \tilde{\mathrm{l}} \rangle$	
$[\texttt{skip}]^{pc}$	$\langle pc+1, \tilde{\mathrm{r}}, \tilde{\mathrm{x}}, \tilde{\mathrm{l}} \rangle$	
$[r := a]^{pc}$	$\langle pc+1, \tilde{\mathrm{r}}[r \mapsto \tilde{\mathscr{A}}[\![a]\!]\tilde{\mathrm{r}}], \tilde{\mathrm{x}}, \tilde{\mathrm{l}} \rangle$	
$[\texttt{if } b \texttt{ goto } l]^{pc}$	$\langle pc+1, \tilde{\mathscr{B}}_{\mathscr{R}}[\![!\, b]\!]\tilde{\mathrm{r}}, \tilde{\mathrm{x}}, \tilde{\mathrm{l}} \rangle$	$\tilde{\mathscr{B}}_{\mathscr{R}}[\![!\, b]\!]\tilde{\mathrm{r}} \neq \tilde{\bot}_{reg}$
$[\texttt{if } b \texttt{ goto } l]^{pc}$	$\langle l, \tilde{\mathscr{B}}_{\mathscr{R}}[\![b]\!]\tilde{\mathrm{r}}, \tilde{\mathrm{x}}, \tilde{\mathrm{l}} \rangle$	$\tilde{\mathscr{B}}_{\mathscr{R}}[\![b]\!]\tilde{\mathrm{r}} \neq \tilde{\bot}_{reg}$
$[\texttt{store } r \texttt{ to } x]^{pc}$	$\langle pc+1, \tilde{\mathrm{r}}, \textsc{write}(\mathrm{T}, \tilde{\mathrm{x}}, x, (\tilde{\mathrm{r}}\ r, \tilde{t})), \tilde{\mathrm{l}} \rangle$	
$[\texttt{load } r \texttt{ from } x]^{pc}$	$\langle pc+1, \tilde{\mathrm{r}}[r \mapsto \textsc{read}(\tilde{\mathrm{x}}, x, \mathrm{T}, \tilde{t})], \tilde{\mathrm{x}}, \tilde{\mathrm{l}} \rangle$	
$[\texttt{lock } lck]^{pc}$	$\langle pc+1, \tilde{\mathrm{r}}, \tilde{\mathrm{x}}, \tilde{\mathrm{l}}[lck \mapsto (locked, \mathrm{T}, \tilde{\textsc{dl}}(\tilde{\mathrm{l}}\ lck), \tilde{\textsc{pown}}(\tilde{\mathrm{l}}\ lck), \tilde{\textsc{rel}}(\tilde{\mathrm{l}}\ lck))] \rangle$	$\tilde{\textsc{own}}(\tilde{\mathrm{l}}\ lck) = \mathrm{T} \wedge$ $(\tilde{\textsc{stt}}(\tilde{\mathrm{l}}\ lck) = unlocked \Rightarrow$ $(\tilde{t} \not\precsim_t \tilde{\textsc{rel}}(\tilde{\mathrm{l}}\ lck) \wedge$ $\tilde{\textsc{dl}}(\tilde{\mathrm{l}}\ lck) \not\precsim_t \tilde{t}))$
$[\texttt{lock } lck]^{pc}$	$\langle pc, \tilde{\mathrm{r}}, \tilde{\mathrm{x}}, \tilde{\mathrm{l}} \rangle$	$\tilde{\textsc{own}}(\tilde{\mathrm{l}}\ lck) \neq \mathrm{T} \vee$ $(\tilde{\textsc{stt}}(\tilde{\mathrm{l}}\ lck) = unlocked \wedge$ $(\tilde{t} \precsim_t \tilde{\textsc{rel}}(\tilde{\mathrm{l}}\ lck) \vee$ $\tilde{\textsc{dl}}(\tilde{\mathrm{l}}\ lck) \precsim_t \tilde{t}))$
$[\texttt{unlock } lck]^{pc}$	$\langle pc+1, \tilde{\mathrm{r}}, \tilde{\mathrm{x}}, \tilde{\mathrm{l}}[lck \mapsto (unlocked, \bot_{thrd}, \tilde{\textsc{dl}}(\tilde{\mathrm{l}}\ lck), \mathrm{T}, \tilde{t})] \rangle$	$\tilde{\textsc{own}}(\tilde{\mathrm{l}}\ lck) = \mathrm{T} \wedge$ $\tilde{\textsc{stt}}(\tilde{\mathrm{l}}\ lck) = locked$
$[\texttt{unlock } lck]^{pc}$	$\langle pc+1, \tilde{\mathrm{r}}, \tilde{\mathrm{x}}, \tilde{\mathrm{l}} \rangle$	$\tilde{\textsc{own}}(\tilde{\mathrm{l}}\ lck) \neq \mathrm{T} \vee$ $\tilde{\textsc{stt}}(\tilde{\mathrm{l}}\ lck) = unlocked$

Table 5.5: $\langle \mathrm{T}, pc, \tilde{\mathrm{r}}, \tilde{\mathrm{x}}, \tilde{\mathrm{l}}, \tilde{t} \rangle \xrightarrow[ax]{\sim} \langle pc', \tilde{\mathrm{r}}', \tilde{\mathrm{x}}', \tilde{\mathrm{l}}' \rangle$, semantics of abstract axiom transitions.

5.8 Abstract Semantics

The abstract transition rules for axiom statements in Table 5.5 are safe approximations of the rules in Table 4.2 with respect to Definition 5.48 (Lemma 5.49).

Definition 5.48 (Soundness of the abstract axiom transition relation): *Assuming that $\tilde{\mathbb{x}}$ contains safe write history (c.f., Definition 5.18), the transition relation $\xrightarrow[ax]{\sim}$ is a safe approximation of $\xrightarrow[ax]{}$, as used by $\xrightarrow[prg]{}$, iff*

$$\forall \tilde{c}^{ax}_{in} \in \tilde{\mathbf{Conf}}^{\mathbf{ax}}_{\mathbf{in}} : \forall c^{ax}_{in} \in \gamma^{ax}_{in}(\tilde{c}^{ax}_{in}) : \forall c^{ax}_{out} \in \mathbf{Conf}^{\mathbf{ax}}_{\mathbf{out}} : \\ (c^{ax}_{in} \xrightarrow[ax]{} c^{ax}_{out} \Rightarrow \exists \tilde{c}^{ax}_{out} \in \tilde{\mathbf{Conf}}^{\mathbf{ax}}_{\mathbf{out}} : (\tilde{c}^{ax}_{in} \xrightarrow[ax]{\sim} \tilde{c}^{ax}_{out} \wedge c^{ax}_{out} \in \gamma^{ax}_{out}(\tilde{c}^{ax}_{out})))$$

where c^{ax}_{in} is generated (c.f., Table 4.3) from a valid configuration (c.f., Definition 4.4); i.e., the lock state is valid with respect to the accumulated time of the given thread. □

Lemma 5.49 (Soundness of $\xrightarrow[ax]{\sim}$):

$\xrightarrow[ax]{\sim}$ is a safe approximation of $\xrightarrow[ax]{}$, with respect to Definition 5.48. □

PROOF. This proof will be conducted by showing for each defined transition that it is safe according to Definition 5.48.

Assume that $\tilde{c}^{ax}_{in} @ \langle \mathrm{T}, pc, \tilde{\mathbb{r}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}, \tilde{t} \rangle \in \tilde{\mathbf{Conf}}^{\mathbf{ax}}_{\mathbf{in}}$ and $c^{ax}_{in} @ \langle \mathrm{T}, pc, \mathbb{r}, \mathbb{x}, \mathbb{l}, t \rangle \in \mathbf{Conf}^{\mathbf{ax}}_{\mathbf{in}}$, such that $c^{ax}_{in} \in \gamma^{ax}_{in}(\tilde{c}^{ax}_{in})$, that $\mathbb{l}$ is valid with respect to t and that $\tilde{\mathbb{x}}$ contains safe write history. Now consider each defined axiom statement.

1. Assume that $\textsc{stm}(\mathrm{T}, pc) = [\texttt{halt}]^{pc}$. From the concrete semantics, it must be that $c^{ax}_{in} \xrightarrow[ax]{} c^{ax}_{out}$, where $c^{ax}_{out} = \langle pc, \mathbb{r}, \mathbb{x}, \mathbb{l} \rangle$. Choose $\tilde{c}^{ax}_{out}$ so that $\tilde{c}^{ax}_{in} \xrightarrow[ax]{\sim} \tilde{c}^{ax}_{out}$, i.e., $\tilde{c}^{ax}_{out} = \langle pc, \tilde{\mathbb{r}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle$. Thus, $c^{ax}_{out} \in \gamma^{ax}_{out}(\tilde{c}^{ax}_{out})$.

2. Assume that $\textsc{stm}(\mathrm{T}, pc) = [\texttt{skip}]^{pc}$. From the concrete semantics, it must be that $c^{ax}_{in} \xrightarrow[ax]{} c^{ax}_{out}$, where $c^{ax}_{out} = \langle pc+1, \mathbb{r}, \mathbb{x}, \mathbb{l} \rangle$. Choose $\tilde{c}^{ax}_{out}$ so that $\tilde{c}^{ax}_{in} \xrightarrow[ax]{\sim} \tilde{c}^{ax}_{out}$, i.e., $\tilde{c}^{ax}_{out} = \langle pc+1, \tilde{\mathbb{r}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle$. Thus, $c^{ax}_{out} \in \gamma^{ax}_{out}(\tilde{c}^{ax}_{out})$.

3. Assume that $\textsc{stm}(\mathrm{T}, pc) = [r := a]^{pc}$. From the concrete semantics, it must be that $c^{ax}_{in} \xrightarrow[ax]{} c^{ax}_{out}$, where $c^{ax}_{out} = \langle pc+1, \mathbb{r}[r \mapsto \mathscr{A}[\![a]\!]\mathbb{r}], \mathbb{x}, \mathbb{l} \rangle$. Choose $\tilde{c}^{ax}_{out}$ so that $\tilde{c}^{ax}_{in} \xrightarrow[ax]{\sim} \tilde{c}^{ax}_{out}$, i.e., $\tilde{c}^{ax}_{out} = \langle pc+1, \tilde{\mathbb{r}}[r \mapsto \tilde{\mathscr{A}}[\![a]\!]\tilde{\mathbb{r}}], \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle$. Since $\tilde{\mathscr{A}}$ is safely induced from $\mathscr{A}$ (see Section 5.3), it must be that $\mathscr{A}[\![a]\!]\mathbb{r} \in \gamma_{val}(\tilde{\mathscr{A}}[\![a]\!]\tilde{\mathbb{r}})$, and hence, $\mathbb{r}[r \mapsto \mathscr{A}[\![a]\!]\mathbb{r}] \in \gamma_{reg}(\tilde{\mathbb{r}}[r \mapsto \tilde{\mathscr{A}}[\![a]\!]\tilde{\mathbb{r}}])$. Thus, $c^{ax}_{out} \in \gamma^{ax}_{out}(\tilde{c}^{ax}_{out})$.

4. Assume that $\text{STM}(\text{T}, pc) = [\texttt{if } b \texttt{ goto } l]^{pc}$. Then two cases must be considered.

   (a) In the first case, $\mathscr{B}[\![b]\!]\mathbb{r}$. This means that $c_{in}^{ax} \xrightarrow[ax]{} c_{out}^{ax}$, where $c_{out}^{ax} = \langle l, \mathbb{r}, \mathbb{x}, \mathbb{l} \rangle$. Now, choose $\tilde{c}_{out}^{ax}$ so that $\tilde{c}_{in}^{ax} \xrightarrow[ax]{\sim} \tilde{c}_{out}^{ax}$ by the corresponding branch (i.e., $\tilde{\mathscr{B}}_{\mathscr{R}}[\![b]\!]\tilde{\mathbb{r}} \neq \tilde{\bot}_{reg}$); i.e., $\tilde{c}_{out}^{ax} = \langle l, \tilde{\mathscr{B}}_{\mathscr{R}}[\![b]\!]\tilde{\mathbb{r}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle$. Since $\tilde{\mathscr{B}}_{\mathscr{R}}$ is safely induced from $\mathscr{B}$ (see Section 5.4), it must be that $\tilde{\mathscr{B}}_{\mathscr{R}}[\![b]\!]\tilde{\mathbb{r}} \neq \tilde{\bot}_{reg}$ and that $\gamma_{out}^{ax}(\tilde{c}_{out}^{ax})$ contains, at least, all cases where $\mathscr{B}[\![b]\!]\mathbb{r}$. Thus, it must be the case that $c_{out}^{ax} \in \gamma_{out}^{ax}(\tilde{c}_{out}^{ax})$.

   (b) In the second case, $\neg\mathscr{B}[\![b]\!]\mathbb{r}$. This means that $c_{in}^{ax} \xrightarrow[ax]{} c_{out}^{ax}$, where $c_{out}^{ax} = \langle pc+1, \mathbb{r}, \mathbb{x}, \mathbb{l} \rangle$. Now, choose $\tilde{c}_{out}^{ax}$ so that $\tilde{c}_{in}^{ax} \xrightarrow[ax]{\sim} \tilde{c}_{out}^{ax}$ by the corresponding branch (i.e., $\tilde{\mathscr{B}}_{\mathscr{R}}[\![!\, b]\!]\tilde{\mathbb{r}} \neq \tilde{\bot}_{reg}$); i.e., $\tilde{c}_{out}^{ax} = \langle pc+1, \tilde{\mathscr{B}}_{\mathscr{R}}[\![!\, b]\!]\tilde{\mathbb{r}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle$. Since $\tilde{\mathscr{B}}_{\mathscr{R}}$ is safely induced from $\mathscr{B}$ (see Section 5.4), it must be that $\tilde{\mathscr{B}}_{\mathscr{R}}[\![!\, b]\!]\tilde{\mathbb{r}} \neq \tilde{\bot}_{reg}$ and that $\gamma_{out}^{ax}(\tilde{c}_{out}^{ax})$ contains, at least, all cases where $\neg\mathscr{B}[\![b]\!]\mathbb{r}$. Thus, it must be the case that $c_{out}^{ax} \in \gamma_{out}^{ax}(\tilde{c}_{out}^{ax})$.

5. Assume that $\text{STM}(\text{T}, pc) = [\texttt{store } r \texttt{ to } x]^{pc}$. From the concrete semantics, it must be that $c_{in}^{ax} \xrightarrow[ax]{} c_{out}^{ax}$, where $c_{out}^{ax} = \langle pc+1, \mathbb{r}, \mathbb{x}[x \mapsto (\mathbb{x}\, x)[\text{T} \mapsto \{(\mathbb{r}\, r, t)\}]], \mathbb{l} \rangle$. Choose $\tilde{c}_{out}^{ax}$ so that $\tilde{c}_{in}^{ax} \xrightarrow[ax]{\sim} \tilde{c}_{out}^{ax}$, i.e., $\tilde{c}_{out}^{ax} = \langle pc+1, \tilde{\mathbb{r}}, \text{WRITE}(\text{T}, \tilde{\mathbb{x}}, x, (\tilde{\mathbb{r}}\, r, \tilde{t})), \tilde{\mathbb{l}} \rangle$. It is easy to see that (c.f., Algorithm 5.5) $\mathbb{x}[x \mapsto (\mathbb{x}\, x)[\text{T} \mapsto \{(\mathbb{r}\, r, t)\}]] \in \gamma_{var}(\text{WRITE}(\text{T}, \tilde{\mathbb{x}}, x, (\tilde{\mathbb{r}}\, r, \tilde{t})))$, thus $c_{out}^{ax} \in \gamma_{out}^{ax}(\tilde{c}_{out}^{ax})$.

6. Assume that $\text{STM}(\text{T}, pc) = [\texttt{load } r \texttt{ from } x]^{pc}$. From the concrete semantics, $c_{in}^{ax} \xrightarrow[ax]{} c_{out}^{ax}$, where $c_{out}^{ax} = \langle pc+1, \mathbb{r}[r \mapsto v], \mathbb{x}, \mathbb{l} \rangle$ for some v such that $\exists t' \in \textbf{Time} : (v, t') \in \bigcup_{\text{T}' \in \textbf{Thrd}}((\mathbb{x}\, x)\ \text{T}')$ if $\bigcup_{\text{T}' \in \textbf{Thrd}}((\mathbb{x}\, x)\ \text{T}') \neq \emptyset$ and $v \in \gamma_{int}([-\infty, \infty])$ otherwise. Choose $\tilde{c}_{out}^{ax}$ so that $\tilde{c}_{in}^{ax} \xrightarrow[ax]{\sim} \tilde{c}_{out}^{ax}$, i.e., $\tilde{c}_{out}^{ax} = \langle pc+1, \tilde{\mathbb{r}}[r \mapsto \text{READ}(\tilde{\mathbb{x}}, x, \text{T}, \tilde{t})], \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle$. Since $\tilde{\mathbb{x}}$ is safe at time $\tilde{t}$ and READ then returns a safe value (Lemma 5.26), it must be that $v \in \gamma_{val}(\text{READ}(\tilde{\mathbb{x}}, x, \text{T}, \tilde{t}))$ and thus $c_{out}^{ax} \in \gamma_{out}^{ax}(\tilde{c}_{out}^{ax})$.

7. Assume that $\text{STM}(\text{T}, pc) = [\texttt{lock } lck]^{pc}$. Then two cases must be considered.

   (a) In the first case, $\text{OWN}(\mathbb{l}\ lck) = \text{T}$. From the concrete semantics, it must be that $c_{in}^{ax} \xrightarrow[ax]{} c_{out}^{ax}$, where $c_{out}^{ax} = \langle pc+1, \mathbb{r}, \mathbb{x}, \mathbb{l}[lck \mapsto$

$(\mathit{locked}, \mathrm{T}, \mathrm{DL}(\mathbb{l}\ \mathit{lck}), \mathrm{POWN}(\mathbb{l}\ \mathit{lck}), \mathrm{REL}(\mathbb{l}\ \mathit{lck}))]\rangle$. Choose $\tilde{c}^{ax}_{out}$ so that $\tilde{c}^{ax}_{in} \xrightarrow[ax]{\sim} \tilde{c}^{ax}_{out}$ by the corresponding branch, $(\tilde{\mathrm{OWN}}(\tilde{\mathbb{l}}\ \mathit{lck}) = \mathrm{T} \wedge (\tilde{\mathrm{STT}}(\tilde{\mathbb{l}}\ \mathit{lck}) = \mathit{unlocked} \Rightarrow (\tilde{t} \not\tilde{<}_t \tilde{\mathrm{REL}}(\tilde{\mathbb{l}}\ \mathit{lck}) \wedge \tilde{\mathrm{DL}}(\tilde{\mathbb{l}}\ \mathit{lck}) \not\tilde{<}_t \tilde{t})))$; i.e., $\tilde{c}^{ax}_{out} = \langle pc+1, \tilde{\mathbb{r}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}[\mathit{lck} \mapsto (\mathit{locked}, \mathrm{T}, \tilde{\mathrm{DL}}(\tilde{\mathbb{l}}\ \mathit{lck}), \tilde{\mathrm{POWN}}(\tilde{\mathbb{l}}\ \mathit{lck}), \tilde{\mathrm{REL}}(\tilde{\mathbb{l}}\ \mathit{lck}))]\rangle$. Note that if $\mathrm{STT}(\mathbb{l}\ \mathit{lck}) = \mathit{unlocked}$, it is implied that $t \not< \mathrm{REL}(\mathbb{l}\ \mathit{lck}) \wedge \mathrm{DL}(\mathbb{l}\ \mathit{lck}) \not< t$ (Lemma 4.6). Thus, it must be the case that $c^{ax}_{out} \in \gamma^{ax}_{out}(\tilde{c}^{ax}_{out})$.

(b) In the second case, $\mathrm{OWN}(\mathbb{l}\ \mathit{lck}) \neq \mathrm{T}$. From the concrete semantics, it must be that $c^{ax}_{in} \xrightarrow[ax]{} c^{ax}_{out}$, where $c^{ax}_{out} = \langle pc, \mathbb{r}, \mathbb{x}, \mathbb{l}\rangle$. Choose $\tilde{c}^{ax}_{out}$ so that $\tilde{c}^{ax}_{in} \xrightarrow[ax]{\sim} \tilde{c}^{ax}_{out}$ by the corresponding branch, $\tilde{\mathrm{OWN}}(\tilde{\mathbb{l}}\ \mathit{lck}) \neq \mathrm{T} \vee (\tilde{\mathrm{STT}}(\tilde{\mathbb{l}}\ \mathit{lck}) = \mathit{unlocked} \wedge (\tilde{t} \tilde{<}_t \tilde{\mathrm{REL}}(\tilde{\mathbb{l}}\ \mathit{lck}) \vee \tilde{\mathrm{DL}}(\tilde{\mathbb{l}}\ \mathit{lck}) \tilde{<}_t \tilde{t}))$; i.e., $\tilde{c}^{ax}_{out} = \langle pc, \tilde{\mathbb{r}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle$. Thus, it must be the case that $c^{ax}_{out} \in \gamma^{ax}_{out}(\tilde{c}^{ax}_{out})$.

8. Assume that $\mathrm{STM}(\mathrm{T}, pc) = [\texttt{unlock}\ \mathit{lck}]^{pc}$. Then two cases must be considered.

(a) In the first case, $\mathrm{OWN}(\mathbb{l}\ \mathit{lck}) = \mathrm{T}$. From the concrete semantics, it must be that $c^{ax}_{in} \xrightarrow[ax]{} c^{ax}_{out}$, where $c^{ax}_{out} = \langle pc+1, \mathbb{r}, \mathbb{x}, \mathbb{l}[\mathit{lck} \mapsto (\mathit{unlocked}, \bot_{\mathit{thrd}}, \mathrm{DL}(\mathbb{l}\ \mathit{lck}), \mathrm{T}, t)]\rangle$. Choose $\tilde{c}^{ax}_{out}$ so that $\tilde{c}^{ax}_{in} \xrightarrow[ax]{\sim} \tilde{c}^{ax}_{out}$ by the corresponding branch, $\tilde{\mathrm{OWN}}(\tilde{\mathbb{l}}\ \mathit{lck}) = \mathrm{T} \wedge \tilde{\mathrm{STT}}(\tilde{\mathbb{l}}\ \mathit{lck}) = \mathit{locked}$; i.e., $\tilde{c}^{ax}_{out} = \langle pc+1, \tilde{\mathbb{r}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}[\mathit{lck} \mapsto (\mathit{unlocked}, \bot_{\mathit{thrd}}, \tilde{\mathrm{DL}}(\tilde{\mathbb{l}}\ \mathit{lck}), \mathrm{T}, \tilde{t})]\rangle$. Note that in the concrete case, $\mathrm{STT}(\mathbb{l}\ \mathit{lck}) = \mathit{locked}$ whenever $\mathrm{OWN}(\mathbb{l}\ \mathit{lck}) \neq \bot_{\mathit{thrd}}$ for a valid configuration (Definition 4.4). Thus, it must be the case that $c^{ax}_{out} \in \gamma^{ax}_{out}(\tilde{c}^{ax}_{out})$.

(b) In the second case, $\mathrm{OWN}(\mathbb{l}\ \mathit{lck}) \neq \mathrm{T}$. From the concrete semantics, it must be that $c^{ax}_{in} \xrightarrow[ax]{} c^{ax}_{out}$, where $c^{ax}_{out} = \langle pc+1, \mathbb{r}, \mathbb{x}, \mathbb{l}\rangle$. Choose $\tilde{c}^{ax}_{out}$ so that $\tilde{c}^{ax}_{in} \xrightarrow[ax]{\sim} \tilde{c}^{ax}_{out}$ by the corresponding branch, $\tilde{\mathrm{OWN}}(\tilde{\mathbb{l}}\ \mathit{lck}) \neq \mathrm{T} \vee \tilde{\mathrm{STT}}(\tilde{\mathbb{l}}\ \mathit{lck}) = \mathit{unlocked}$; i.e., $\tilde{c}^{ax}_{out} = \langle pc+1, \tilde{\mathbb{r}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle$. Thus, it must be the case that $c^{ax}_{out} \in \gamma^{ax}_{out}(\tilde{c}^{ax}_{out})$. ∎

$$\mathbf{Thrd}_{exe} \neq \emptyset \wedge \forall \mathrm{T} \in \mathbf{Thrd}_{exe} : \langle \mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{\mathbb{x}}, \tilde{\mathbb{I}}'', \tilde{\imath}^{a\prime}_{\mathrm{T}} \rangle \xrightarrow[ax]{\sim} \langle pc'_{\mathrm{T}}, \tilde{\mathbb{r}}'_{\mathrm{T}}, \tilde{\mathbb{x}}'_{\mathrm{T}}, \tilde{\mathbb{I}}'_{\mathrm{T}} \rangle$$

$$\tilde{c} @ \langle [\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{\imath}^{a}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{I}} \rangle \xrightarrow[prg]{\sim} \tilde{c}' @ \langle [\mathrm{T}, (\mathrm{T} \in \mathbf{Thrd}_{exe} \ ? \ pc'_{\mathrm{T}} : pc_{\mathrm{T}}), (\mathrm{T} \in \mathbf{Thrd}_{exe} \ ? \ \tilde{\mathbb{r}}'_{\mathrm{T}} : \tilde{\mathbb{r}}_{\mathrm{T}}), \tilde{\imath}^{a\prime}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}', \tilde{\mathbb{I}}' \rangle$$

where

$$\tilde{\imath}^{r}_{\mathrm{T}} = \textsc{absTime}(\tilde{c}, \mathrm{T})$$

$$\tilde{\imath}_{all} = \alpha_t(\{\min(\{\min(\gamma_t(\tilde{\imath}^{a}_{\mathrm{T}} \mathbin{\tilde{+}_t} \tilde{\imath}^{r}_{\mathrm{T}})) \mid B\}), \min(\{\max(\gamma_t(\tilde{\imath}^{a}_{\mathrm{T}} \mathbin{\tilde{+}_t} \tilde{\imath}^{r}_{\mathrm{T}})) \mid B\})\}) \quad \textbf{where } B \Longleftrightarrow \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \wedge \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) \neq [\texttt{halt}]^{pc_{\mathrm{T}}} \wedge \forall lck \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{lock } lck]^{pc_{\mathrm{T}}} \Rightarrow \textsc{oŵn}(\tilde{\mathbb{I}}\ lck) \in \{\bot_{thrd}, \mathrm{T}\})$$

$$\mathbf{Thrd}^{all}_{exe} = \{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \mid \tilde{\imath}_{all} \mathbin{\tilde{\sqcap}_t} (\tilde{\imath}^{a}_{\mathrm{T}} \mathbin{\tilde{+}_t} \tilde{\imath}^{r}_{\mathrm{T}}) \neq \tilde{\bot}_t \wedge \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) \neq [\texttt{halt}]^{pc_{\mathrm{T}}}\}$$

$$\tilde{\mathbb{I}}''\ lck = \begin{cases} (\mathit{unlocked}, \mathrm{T}', \textsc{dlLock}(\tilde{c}, lck), \textsc{poŵn}(\tilde{\mathbb{I}}\ lck), \textsc{rẽl}(\tilde{\mathbb{I}}\ lck)) & \textbf{for some } \mathrm{T}' \in \{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \mid \exists l \in \mathbf{Lbl}_{\mathrm{T}} : \textsc{stm}(\mathrm{T}, l) = [\texttt{lock } lck]^{l}\}, \textbf{if } \exists \mathrm{T} \in \mathbf{Thrd}^{all}_{exe} : \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{lock } lck]^{pc_{\mathrm{T}}} \wedge \textsc{oŵn}(\tilde{\mathbb{I}}\ lck) = \bot_{thrd} \\ \tilde{\mathbb{I}}\ lck & \textbf{otherwise} \end{cases}$$

$$\mathbf{Thrd}_{\mathbf{hold}} = \{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \mid \exists lck \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{lock } lck]^{pc_{\mathrm{T}}} \wedge \textsc{oŵn}(\tilde{\mathbb{I}}''\ lck) \neq \mathrm{T}) \vee \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{halt}]^{pc_{\mathrm{T}}}\}$$

$$\tilde{\imath} = \alpha_t(\{\min(\{\min(\gamma_t(\tilde{\imath}^{a}_{\mathrm{T}} \mathbin{\tilde{+}_t} \tilde{\imath}^{r}_{\mathrm{T}})) \mid \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \setminus \mathbf{Thrd}_{\mathbf{hold}}\}), \min(\{\max(\gamma_t(\tilde{\imath}^{a}_{\mathrm{T}} \mathbin{\tilde{+}_t} \tilde{\imath}^{r}_{\mathrm{T}})) \mid \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \setminus \mathbf{Thrd}_{\mathbf{hold}}\})\})$$

$$\mathbf{Thrd}_{exe} = \{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \setminus \mathbf{Thrd}_{\mathbf{hold}} \mid \tilde{\imath} \mathbin{\tilde{\sqcap}_t} (\tilde{\imath}^{a}_{\mathrm{T}} \mathbin{\tilde{+}_t} \tilde{\imath}^{r}_{\mathrm{T}}) \neq \tilde{\bot}_t\}$$

$$\tilde{\mathbb{I}}'\ lck = \begin{cases} \tilde{\mathbb{I}}'_{\textsc{oŵn}(\tilde{\mathbb{I}}''\ lck)}\ lck & \textbf{if } \textsc{oŵn}(\tilde{\mathbb{I}}''\ lck) \in \mathbf{Thrd}_{exe} \\ \tilde{\mathbb{I}}''\ lck & \textbf{otherwise} \end{cases}$$

$$\tilde{\mathbb{x}}' = \begin{cases} \textsc{trim}(\tilde{\mathbb{x}}'', \tilde{\imath}) & \textbf{if } \mathbf{Thrd}_{\tilde{c}} = \mathbf{Thrd} \\ \tilde{\mathbb{x}}'' & \textbf{otherwise} \end{cases} \quad \textbf{where } (\tilde{\mathbb{x}}''\ x)\ \mathrm{T} = \begin{cases} (\tilde{\mathbb{x}}'_{\mathrm{T}}\ x)\ \mathrm{T} & \textbf{if } \mathrm{T} \in \mathbf{Thrd}_{exe} \\ (\tilde{\mathbb{x}}\ x)\ \mathrm{T} & \textbf{otherwise} \end{cases}$$

$$\tilde{\imath}^{a\prime}_{\mathrm{T}} = \textsc{accTime}(\langle [\mathrm{T}', pc_{\mathrm{T}'}, \tilde{\mathbb{r}}_{\mathrm{T}'}, \tilde{\imath}^{a}_{\mathrm{T}'}]_{\mathrm{T}' \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{I}}'' \rangle, \mathbf{Thrd}_{exe}, \mathrm{T})$$

Table 5.6: $\tilde{c} \xrightarrow[prg]{\sim} \tilde{c}'$, semantics of abstract program transitions.

The abstract transition rule for program configurations in Table 5.6 is an approximation of the concrete rule in Table 4.3. The abstract rule now defines a window in time, $\tilde{t}$, (since $\tilde{\mathbf{Time}} = \mathbf{Intv}$) that determines which threads are included in $\mathbf{Thrd}_{exe}$. The window reaches from the earliest point in time when some thread might execute its active statements, to the earliest point in time when some thread must execute its active statements. Note that DLLOCK and ACCTIME are defined in Algorithms 5.11 and 5.12, respectively, and that ABSTIME is assumed to be a safe approximation of TIME, as specified in Assumption 5.50; however, the definitions of these functions are out of the scope for this thesis.

Assumption 5.50 (ABSTIME is safe and non-negative):
It is assumed that ABSTIME *is a "non-negative" function that safely approximates* TIME *in the interval domain. More formally, it is assumed that*

$$\forall \tilde{c} @ \langle [\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle \in \tilde{\mathbf{Conf}} : \forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : 0 \leq \min(\gamma_t(\textsc{absTime}(\tilde{c}, \mathrm{T})))$$

and

$$\begin{aligned} &\forall \tilde{c} @ \langle [\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle \in \tilde{\mathbf{Conf}} : \\ &\forall c @ \langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^{a}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}}, \mathbb{x}, \mathbb{l} \rangle \in \mathbf{Conf} : \\ &\quad (\mathbf{Thrd}_{\tilde{c}} \subseteq \mathbf{Thrd} \Rightarrow \forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : ((pc_{\mathrm{T}} = pc^{\tilde{c}}_{\mathrm{T}} \wedge t^{a}_{\mathrm{T}} \in \gamma_t(\tilde{t}^{a}_{\mathrm{T}})) \Rightarrow \\ &\qquad \textsc{time}(c, \mathrm{T}) \in \gamma_t(\textsc{absTime}(\tilde{c}, \mathrm{T})))) \end{aligned}$$

□

Algorithm 5.11 Determine Deadline for Lock Owner Assignment

function DLLOCK($\tilde{c}@\langle[\mathrm{T}', pc_{\mathrm{T}'}, \tilde{\mathbb{r}}_{\mathrm{T}'}, \tilde{t}^{a}_{\mathrm{T}'}]_{\mathrm{T}' \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle, lck$)
  $\tilde{t}_{dl} \leftarrow \tilde{\top}_t$
  for all $\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}$ **do**
    if STM$(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc_{\mathrm{T}}}$ **then**
      $\tilde{c}' \leftarrow \tilde{c}$
      $\tilde{t}^{a\prime}_{\mathrm{T}} \leftarrow \tilde{t}^{a}_{\mathrm{T}}$
      repeat
        $\tilde{c}' \leftarrow \langle[\mathrm{T}', pc_{\mathrm{T}'}, \tilde{\mathbb{r}}_{\mathrm{T}'}, (\mathrm{T} = \mathrm{T}'\ ?\ \tilde{t}^{a\prime}_{\mathrm{T}} : \tilde{t}^{a}_{\mathrm{T}'})]_{\mathrm{T}' \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle$
        $\tilde{t}^{a\prime}_{\mathrm{T}} \leftarrow \tilde{t}^{a\prime}_{\mathrm{T}} \dot{+}_t$ ABSTIME$(\tilde{c}', \mathrm{T})$
      until RĚL$(\tilde{\mathbb{l}}\ lck) \tilde{<}_t \tilde{t}^{a\prime}_{\mathrm{T}} \vee 0 \in \gamma_t($ABSTIME$(\tilde{c}', \mathrm{T}))$
      if RĚL$(\tilde{\mathbb{l}}\ lck) \not\tilde{<}_t \tilde{t}^{a\prime}_{\mathrm{T}} \wedge 0 \in \gamma_t($ABSTIME$(\tilde{c}', \mathrm{T}))$ **then**
        $\tilde{t} \leftarrow \tilde{t}^{a\prime}_{\mathrm{T}}\ \tilde{\sqcup}_t$ RĚL$(\tilde{\mathbb{l}}\ lck)$
        $\tilde{c}' \leftarrow \langle[\mathrm{T}', pc_{\mathrm{T}'}, \tilde{\mathbb{r}}_{\mathrm{T}'}, (\mathrm{T} = \mathrm{T}'\ ?\ \tilde{t} : \tilde{t}^{a}_{\mathrm{T}'})]_{\mathrm{T}' \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle$
        $\tilde{t}_{dl} \leftarrow \tilde{t}_{dl}\ \tilde{\sqcap}_t\ ((\tilde{t} \dot{+}_t$ ABSTIME$(\tilde{c}', \mathrm{T}))\ \tilde{\sqcup}_t\ \alpha_t(\{-\infty\}))$
      else
        $\tilde{t}_{dl} \leftarrow \tilde{t}_{dl}\ \tilde{\sqcap}_t\ (\tilde{t}^{a\prime}_{\mathrm{T}}\ \tilde{\sqcup}_t\ \alpha_t(\{-\infty\}))$
      end if
    end if
  end for
  return $\tilde{t}_{dl}$
end function

Algorithm 5.12 Determine Accumulated Execution Time

function ACCTIME($\tilde{c}@\langle[\mathrm{T}', pc_{\mathrm{T}'}, \tilde{\mathbb{r}}_{\mathrm{T}'}, \tilde{t}^{a}_{\mathrm{T}'}]_{\mathrm{T}' \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle, \mathbf{Thrd}_{exe}, \mathrm{T}$)
  $\tilde{t}^{a\prime}_{\mathrm{T}} \leftarrow \tilde{t}^{a}_{\mathrm{T}}$
  if $\mathrm{T} \in \mathbf{Thrd}_{exe}$ **then**
    $\tilde{t}^{r}_{\mathrm{T}} \leftarrow$ ABSTIME$(\tilde{c}, \mathrm{T})$
    if $\forall lck \in \mathbf{Lck} :$ STM$(\mathrm{T}, pc_{\mathrm{T}}) \neq [\texttt{lock}\ lck]^{pc_{\mathrm{T}}}$ **then**
      $\tilde{t}^{a\prime}_{\mathrm{T}} \leftarrow \tilde{t}^{a}_{\mathrm{T}} \dot{+}_t \tilde{t}^{r}_{\mathrm{T}}$
    else

Algorithm 5.12 ***Cont.*** Determine Accumulated Execution Time

```
8:          for all lck ∈ Lck do
9:            if STM(T, pc_T) = [lock lck]^{pc_T} ∧ OWN(Ĩ lck) = T then
10:             if STT(Ĩ lck) = locked then
11:               t̃^{a'}_T ← t̃^a_T +̃_t t̃^r_T
12:             else if DL(Ĩ lck) ≲_t (t̃^a_T +̃_t t̃^r_T) then
13:               t̃^{a'}_T ← t̃^a_T +̃_t t̃^r_T
14:             else if (t̃^a_T +̃_t t̃^r_T) ≲_t REL(Ĩ lck) then
15:               c̃' ← c̃
16:               while (t̃^{a'}_T +̃_t ABSTIME(c̃', T)) ≲_t REL(Ĩ lck) do
17:                 t̃^{a'}_T ← t̃^{a'}_T +̃_t ABSTIME(c̃', T)
18:                 c̃' ← ⟨[T', pc_T', r̃_T', (T = T' ? t̃^{a'}_T : t̃^a_T')]_{T'∈Thrd_c̃}, x̃, Ĩ⟩
19:               end while
20:             else if POWN(Ĩ lck) = T ∨ REL(Ĩ lck) ≲_t (t̃^a_T +̃_t t̃^r_T) then
21:               t̃^{a'}_T ← (t̃^a_T +̃_t t̃^r_T) ⊓̃_t DL(Ĩ lck)
22:             else        ▷ STT(Ĩ lck) = unlocked ∧ REL(Ĩ lck) ⊓̃_t (t̃^a_T +̃_t t̃^r_T) ≠ ⊥̃_t ∧
                                POWN(Ĩ lck) ≠ T ∧ DL(Ĩ lck) ≴_t (t̃^a_T +̃_t t̃^r_T))
23:               t̃^{a''}_T ← t̃^a_T +̃_t t̃^r_T
24:               c̃' ← c̃
25:               repeat
26:                 if DL(Ĩ lck) ≲_t (t̃^{a'}_T +̃_t ABSTIME(c̃', T)) then
27:                   t̃^r_T ← ⊤̃_t
28:                 else if 0 ∈ γ_t(ABSTIME(c̃', T)) then
29:                   t̃ ← (t̃^{a'}_T ⊔̃_t α_t({∞})) ⊓̃_t REL(Ĩ lck)
30:                   c̃' ← ⟨[T', pc_T', r̃_T', (T = T' ? t̃ : t̃^a_T')]_{T'∈Thrd_c̃}, x̃, Ĩ⟩
31:                   t̃^{a'}_T ← t̃ +̃_t ABSTIME(c̃', T)
32:                   t̃^r_T ← ⊤̃_t
33:                 else
34:                   t̃^{a'}_T ← t̃^{a'}_T +̃_t ABSTIME(c̃', T)
35:                   c̃' ← ⟨[T', pc_T', r̃_T', (T = T' ? t̃^{a'}_T : t̃^a_T')]_{T'∈Thrd_c̃}, x̃, Ĩ⟩
36:                 end if
37:               until t̃^r_T = ⊤̃_t ∨ REL(Ĩ lck) ≲_t t̃^{a'}_T
38:               t̃^{a'}_T ← (t̃^{a''}_T ⊔̃_t t̃^{a'}_T) ⊓̃_t DL(Ĩ lck) ⊓̃_t (REL(Ĩ lck) ⊔̃_t α_t({∞}))
39:             end if
40:           end if
41:         end for
42:       end if
43:     end if
44:     return t̃^{a'}_T
45: end function
```

Since **Time** is approximated using **Tiñe**, there are some consequences, rendering $\xrightarrow[prg]{\sim}$ an unsafe approximation of $\xrightarrow[prg]{}$ (c.f., Tables 4.3 and 5.6) in the general case.

1. The sets of threads to execute, i.e., $\mathbf{Thrd}_{exe}$, might differ between $c \in \mathbf{Conf}$ and $\tilde{c} \in \mathbf{Cõnf}$, even if $c \in \gamma_{conf}(\tilde{c})$. Because of this, different program points might be "visited" in the concrete and abstract cases, and thus, fixed-point calculations on $\xrightarrow[prg]{\sim}$ in the traditional sense cannot be used to find a safe over-approximation of the concrete collecting semantics (see for example [14, 23]).

2. The execution of `load`-statements cannot be safely approximated using $\xrightarrow[prg]{\sim}$ if $|\mathbf{Thrd}_{exe}| > 1$ and the value of a global variable is to be loaded. The reason for this is that executing `load`-statements introduces data-dependencies between the threads, and the READ-function could return a value for which all possible writes have not been taken into account; i.e., all `store`-statements that could affect the variable have not yet been executed (and thus, $\tilde{\mathbb{x}}$ does not contain safe write history). To see this, assume that for some abstract configuration, $\mathbf{Thrd}_{exe} = \{\mathrm{T}_1, \mathrm{T}_2\}$, $\textsc{stm}(\mathrm{T}_1, pc_{\mathrm{T}_1}) = [\texttt{load}\ r\ \texttt{from}\ x]^{pc_{\mathrm{T}_1}}$, $\textsc{stm}(\mathrm{T}_2, pc_{\mathrm{T}_2}) = [\texttt{skip}]^{pc_{\mathrm{T}_2}}$ and $\textsc{stm}(\mathrm{T}_2, pc_{\mathrm{T}_2} + 1) = [\texttt{store}\ r'\ \texttt{to}\ x]^{pc_{\mathrm{T}_2}+1}$. When a transition occurs, the `load`- and `skip`-statements are considered. However, since the execution time of the `store`-statement (the abstract "point" in time when the thread's *pc* is updated) overlaps with the execution time of the `load`-statement, the resulting value of r in T_1 should be affected by the value of r' in T_2, but this will not be the case.

3. A similar reasoning to that for `load`-statements holds for `lock`-statements – an unlocked lock, $lck \in \mathbf{Lck}$, cannot simply be assigned to one of the threads in $\mathbf{Thrd}_{exe}$ that issues `lock` *lck*. This is because in the concrete case, the lock might be assigned to another thread in $\mathbf{Thrd}_{exe}$ (that might not yet be executing `lock` *lck* in the abstract case). Thus, the only safe option is to make assignments to, at least, each thread specified in the considered abstract configuration that at some point might acquire *lck*. This is because these threads (even if currently not in $\mathbf{Thrd}_{exe}$) could compete for *lck* with subsequent statements. If a thread that has been assigned *lck* actually does not compete for *lck*, this can be detected if the thread reaches a `halt`-statement or using the deadline parameter in the state for *lck*.

4. A transition sequence containing deadlocked configurations will not be safely approximated. In the concrete case, the threads included in the deadlock are spinning on the locks they are waiting to acquire. This means that time moves forward for these threads (given that TIME is non-zero). However, in the abstract case, the threads will be frozen and their accumulated times do not increase on a transition.

To handle these issues, the analysis will be proven to safely approximate the timing bounds of any concrete configuration, $c@\langle[\mathrm{T}, pc_\mathrm{T}, \mathbb{r}_\mathrm{T}, t^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle \in \mathbf{Conf}$, in the finite collecting semantics, $\mathscr{C}(C)$, of a program in the initial states described by the configurations in C, such that $\forall \mathrm{T} \in \mathbf{Thrd} : \textsc{stm}(\mathrm{T}, pc_\mathrm{T}) = [\texttt{halt}]^{pc_\mathrm{T}}$. The analysis will also be proven to safely approximate the timing bounds of some (but not all) collecting semantics that might be infinite. More on this in Chapter 6.

$\xrightarrow[prg]{\sim}$ will be proven to be a safe approximation of $\xrightarrow[prg]{}$ in any finite collecting semantics, with respect to each thread individually, for any configuration, $\tilde{c}@\langle[\mathrm{T}, pc_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \tilde{\mathbf{Conf}}$, such that $|\mathbf{Thrd}_{exe}| \not> 1 \vee \{\mathrm{T} \in \mathbf{Thrd}_{exe} \mid \exists r \in \mathbf{Reg}_\mathrm{T} : \exists x \in \mathbf{Var_g} : \textsc{stm}(\mathrm{T}, pc_\mathrm{T}) = [\texttt{load}\ r\ \texttt{from}\ x]^{pc_\mathrm{T}}\} = \emptyset$, where $\mathbf{Var_g}$ is the set of all global variables (i.e., variables that might transfer data between threads); i.e., either no thread issues a `load`-statement on a global variable, or there is such a thread and it is the sole thread that is executed, which means that $\tilde{\mathbb{x}}$ must contain safe write history since no more writes on x can occur before the given `load`-statement has been executed.

One thing to notice from how $\xrightarrow[prg]{\sim}$ is defined is that an abstract configuration cannot have the same restrictions for it being valid as a concrete configuration does (c.f., Definition 4.4). When a thread (in $\mathbf{Thrd}_{exe}$) wants to acquire some unlocked lock, $\xrightarrow[prg]{\sim}$ can assign the lock to any thread that at some point in the program wants to acquire the lock, as discussed in 3 above. However (quite obviously), the assigned thread might not acquire the lock with its current statement (it is also possible that the thread never acquires the lock at all with its future statements). Therefore, an abstract configuration, $\tilde{c}@\langle[\mathrm{T}, pc_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \tilde{\mathbf{Conf}}$, must be considered temporarily valid even if $\exists lck \in \mathbf{Lck} : (\tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck) \neq \bot_{thrd} \wedge \tilde{\textsc{stt}}(\tilde{\mathbb{l}}\ lck) = \mathit{unlocked})$. As also discussed in 3 above, however, such an abstract configuration can be considered invalid if $\exists lck \in \mathbf{Lck} : (\tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck) \neq \bot_{thrd} \wedge \tilde{\textsc{stt}}(\tilde{\mathbb{l}}\ lck) = \mathit{unlocked} \wedge (\tilde{\textsc{dl}}(\tilde{\mathbb{l}}\ lck) \tilde{<}_t (\tilde{t}^a_{\tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck)} \tilde{+}_t \textsc{absTime}(\tilde{c}, \tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck))) \vee \textsc{stm}(\tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck), pc_{\tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck)}) = [\texttt{halt}]^{pc_{\tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck)}}))$, given that $\tilde{\textsc{dl}}(\tilde{\mathbb{l}}\ lck)$ is a safe approximation of when lck must have been taken

by some thread in the corresponding concrete cases (c.f., Lemma 5.53), if any.

In the concrete case, a free (*unlocked*) lock is acquired as soon as some thread tries to do so (c.f., Tables 4.2 and 4.3 and Lemma 4.5). The purpose of DLLOCK, defined in Algorithm 5.11, is to derive a safe approximation of this point in time (Lemma 5.53). Note that Lemma 5.51 states that accumulating time for each thread individually is safe and that Lemma 5.52 states that the timing of a thread can be analyzed in isolation from all other threads.

Lemma 5.51 (Time accumulation):
Given the two configurations $c@\langle[\mathrm{T}, pc_\mathrm{T}, \mathbb{r}_\mathrm{T}, t^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle \in \mathbf{Conf}$ *and* $\tilde{c}@\langle[\mathrm{T}, pc^{\tilde{c}}_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \tilde{\mathbf{Conf}}$, *such that* $\mathbf{Thrd}_{\tilde{c}} \subseteq \mathbf{Thrd}$, *let* $\mathbf{Thrd}' = \{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \mid t^a_\mathrm{T} \in \gamma_t(\tilde{t}^a_\mathrm{T}) \wedge pc_\mathrm{T} = pc^{\tilde{c}}_\mathrm{T}\}$. *Then* $\forall \mathrm{T} \in \mathbf{Thrd}' : (t^a_\mathrm{T} + \textsc{time}(c,\mathrm{T})) \in \gamma_t(\tilde{t}^a_\mathrm{T} \mathbin{\tilde{+}_t} \textsc{absTime}(\tilde{c},\mathrm{T}))$. □

PROOF. Assume that the configurations $c@\langle[\mathrm{T}, pc_\mathrm{T}, \mathbb{r}_\mathrm{T}, t^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle \in \mathbf{Conf}$ and $\tilde{c}@\langle[\mathrm{T}, pc^{\tilde{c}}_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \tilde{\mathbf{Conf}}$ are such that $\mathbf{Thrd}_{\tilde{c}} \subseteq \mathbf{Thrd}$, and let $\mathbf{Thrd}' = \{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \mid t^a_\mathrm{T} \in \gamma_t(\tilde{t}^a_\mathrm{T}) \wedge pc_\mathrm{T} = pc^{\tilde{c}}_\mathrm{T}\}$. Then, according to Assumption 5.50, $\forall \mathrm{T} \in \mathbf{Thrd}' : \textsc{time}(c,\mathrm{T}) \in \gamma_t(\textsc{absTime}(\tilde{c},\mathrm{T}))$. Since $\forall \mathrm{T} \in \mathbf{Thrd}' : t^a_\mathrm{T} \in \gamma_t(\tilde{t}^a_\mathrm{T})$, it is easy to see that $\forall \mathrm{T} \in \mathbf{Thrd}' : (t^a_\mathrm{T} + \textsc{time}(c,\mathrm{T})) \in \gamma_t(\tilde{t}^a_\mathrm{T} \mathbin{\tilde{+}_t} \textsc{absTime}(\tilde{c},\mathrm{T}))$. ■

Lemma 5.52 (Thread isolation):
Given the two configurations $c^0@\langle[\mathrm{T}, pc^0_\mathrm{T}, \mathbb{r}^0_\mathrm{T}, t^{a^0}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^0, \mathbb{l}^0\rangle \in \mathbf{Conf}$ *and* $\tilde{c}^0@\langle[\mathrm{T}, pc^{\tilde{c}^0}_\mathrm{T}, \tilde{\mathbb{r}}^0_\mathrm{T}, \tilde{t}^{a^0}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}^0, \tilde{\mathbb{l}}^0\rangle \in \tilde{\mathbf{Conf}}$, *and some thread,* $\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}$, *such that* $\mathbf{Thrd}_{\tilde{c}} \subseteq \mathbf{Thrd}$, $t^{a^0}_\mathrm{T} \in \gamma_t(\tilde{t}^{a^0}_\mathrm{T})$ *and* $pc^0_\mathrm{T} = pc^{\tilde{c}^0}_\mathrm{T}$, *and some configuration* $c^{n+1}@\langle[\mathrm{T}, pc^{n+1}_\mathrm{T}, \mathbb{r}^{n+1}_\mathrm{T}, t^{a^{n+1}}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^{n+1}, \mathbb{l}^{n+1}\rangle$, *where*

$$c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^1 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^{n+1}$$

for some $n \geq 0$,

$$t^{a^{n+1}}_\mathrm{T} \in \gamma_t(\tilde{t}^{a^0}_\mathrm{T} \mathbin{\tilde{+}_t} \textsc{absTime}(\tilde{c}^0,\mathrm{T}) \mathbin{\tilde{+}_t} \ldots \mathbin{\tilde{+}_t} \textsc{absTime}(\tilde{c}^n@\langle[\mathrm{T}, pc^{\tilde{c}^n}_\mathrm{T}, \tilde{\mathbb{r}}^n_\mathrm{T}, \tilde{t}^{a^n}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}}, \tilde{\mathbb{x}}^n, \tilde{\mathbb{l}}^n\rangle, \mathrm{T}))$$

given that $\forall i \in \{1,\ldots,n\} : \tilde{t}^{a^i}_\mathrm{T} = \tilde{t}^{a^{i-1}}_\mathrm{T} \mathbin{\tilde{+}_t} \textsc{absTime}(\tilde{c}^{i-1},\mathrm{T})$, $\forall i \in \{0,\ldots,n\} : pc^i_\mathrm{T} = pc^{\tilde{c}^i}_\mathrm{T}$, $\forall i \in \{0,\ldots,n\} : \mathrm{T} \in \mathbf{Thrd}^{c^i}_{exe}$, *and* $\forall c \in \{c^0,\ldots,c^1,\ldots,c^n\} \setminus \{c^0,c^1,\ldots,c^n\} : \mathrm{T} \notin \mathbf{Thrd}^{c}_{exe}$, *where* $\mathbf{Thrd}^{c^i}_{exe}$ *and* $\mathbf{Thrd}^{c}_{exe}$ *are as defined in Table 4.3 for all* c^i *and all other* c *on the trace from* c^0 *to* c^{n+1}. □

PROOF. Assume that the configurations $c^0@\langle[\mathrm{T},pc_\mathrm{T}^0,\mathbb{r}_\mathrm{T}^0,t_\mathrm{T}^{a^0}]_{\mathrm{T}\in\mathbf{Thrd}},\mathbb{x}^0,\mathbb{l}^0\rangle \in$ **Conf** and $\tilde{c}^0@\langle[\mathrm{T},pc_\mathrm{T}^{\tilde{c}^0},\tilde{\mathbb{r}}_\mathrm{T}^0,\tilde{t}_\mathrm{T}^{a^0}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}},\tilde{\mathbb{x}}^0,\tilde{\mathbb{l}}^0\rangle \in$ **Cõnf**, and some thread, $\mathrm{T}\in$ **Thrd**, are such that $\mathbf{Thrd}_{\tilde{c}}\subseteq$ **Thrd**, $t_\mathrm{T}^{a^0}\in\gamma_t(\tilde{t}_\mathrm{T}^{a^0})$ and $pc_\mathrm{T}^0=pc_\mathrm{T}^{\tilde{c}^0}$. Also assume that $c^0\xrightarrow[prg]{}\ldots\xrightarrow[prg]{}c^1\xrightarrow[prg]{}\ldots\xrightarrow[prg]{}c^{n+1}$ for some configuration $c^{n+1}@\langle[\mathrm{T},pc_\mathrm{T}^{n+1},\mathbb{r}_\mathrm{T}^{n+1},t_\mathrm{T}^{a^{n+1}}]_{\mathrm{T}\in\mathbf{Thrd}},\mathbb{x}^{n+1},\mathbb{l}^{n+1}\rangle$ and $n\geq 0$, for which $\forall i\in\{0,\ldots,n\}:$ $\mathrm{T}\in\mathbf{Thrd}_{exe}^{c^i}$, and $\forall c\in\{c^0,\ldots,c^1,\ldots,c^n\}\setminus\{c^0,c^1,\ldots,c^n\}:\mathrm{T}\notin\mathbf{Thrd}_{exe}^{c}$, where $\mathbf{Thrd}_{exe}^{c^i}$ and $\mathbf{Thrd}_{exe}^{c}$ are as defined in Table 4.3 for all c^i and all other c on the trace from c^0 to c^{n+1}.

From Table 4.3, it is easy to see that:

$$\begin{aligned}
t_\mathrm{T}^{a^0} &= t_\mathrm{T}^{a^0}\\
t_\mathrm{T}^{a^1} &= t_\mathrm{T}^{a^0}+\textsc{time}(c^0,\mathrm{T})\\
&\vdots\\
t_\mathrm{T}^{a^{n+1}} &= t_\mathrm{T}^{a^n}+\textsc{time}(c^n,\mathrm{T})=\\
&\qquad t_\mathrm{T}^{a^0}+\textsc{time}(c^0,\mathrm{T})+\ldots+\textsc{time}(c^n,\mathrm{T})
\end{aligned}$$

Let $\{\tilde{c}^0,\ldots,\tilde{c}^n@\langle[\mathrm{T},pc_\mathrm{T}^{\tilde{c}^n},\tilde{\mathbb{r}}_\mathrm{T}^n,\tilde{t}_\mathrm{T}^{a^n}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}},\tilde{\mathbb{x}}^n,\tilde{\mathbb{l}}^n\rangle\}$ be a set of some abstract configurations such that $\tilde{c}^0$ has the properties assumed above, $\forall i\in\{1,\ldots,n\}:$ $pc_\mathrm{T}^i=pc_\mathrm{T}^{\tilde{c}^i}$ and $\forall i\in\{1,\ldots,n\}:\tilde{t}_\mathrm{T}^{a^i}=\tilde{t}_\mathrm{T}^{a^{i-1}}\mathbin{\tilde{+}_t}\textsc{absTime}(\tilde{c}^{i-1},\mathrm{T})$. Then, according to Lemma 5.51:

$$\begin{aligned}
(t_\mathrm{T}^{a^0}+\textsc{time}(c^0,\mathrm{T}))&\in\gamma_t(\tilde{t}_\mathrm{T}^{a^0}\mathbin{\tilde{+}_t}\textsc{absTime}(\tilde{c}^0,\mathrm{T}))\\
&\vdots\\
(t_\mathrm{T}^{a^n}+\textsc{time}(c^n,\mathrm{T}))&\in\gamma_t(\tilde{t}_\mathrm{T}^{a^n}\mathbin{\tilde{+}_t}\textsc{absTime}(\tilde{c}^n,\mathrm{T}))
\end{aligned}$$

Since $t_\mathrm{T}^{a^{n+1}}=t_\mathrm{T}^{a^n}+\textsc{time}(c^n,\mathrm{T})$, this concludes the proof. ∎

Lemma 5.53 (Soundness of DLLOCK):
Given the valid concrete configurations (c.f., Definition 4.4), abstract configurations and lock

$$\begin{aligned}
&c^0@\langle[\mathrm{T},pc_\mathrm{T}^0,\mathbb{r}_\mathrm{T}^0,t_\mathrm{T}^{a^0}]_{\mathrm{T}\in\mathbf{Thrd}},\mathbb{x}^0,\mathbb{l}^0\rangle\in\mathbf{Conf},\\
&c^m@\langle[\mathrm{T},pc_\mathrm{T}^m,\mathbb{r}_\mathrm{T}^m,t_\mathrm{T}^{a^m}]_{\mathrm{T}\in\mathbf{Thrd}},\mathbb{x}^m,\mathbb{l}^m\rangle\in\mathbf{Conf},\\
&c^n@\langle[\mathrm{T},pc_\mathrm{T}^n,\mathbb{r}_\mathrm{T}^n,t_\mathrm{T}^{a^n}]_{\mathrm{T}\in\mathbf{Thrd}},\mathbb{x}^n,\mathbb{l}^n\rangle\in\mathbf{Conf},\\
&\tilde{c}^0@\langle[\mathrm{T},pc_\mathrm{T}^{\tilde{c}^0},\tilde{\mathbb{r}}_\mathrm{T}^0,\tilde{t}_\mathrm{T}^{a^0}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^0}},\tilde{\mathbb{x}}^0,\tilde{\mathbb{l}}^0\rangle\in\mathbf{Cõnf},\\
&\tilde{c}^j@\langle[\mathrm{T},pc_\mathrm{T}^{\tilde{c}^j},\tilde{\mathbb{r}}_\mathrm{T}^j,\tilde{t}_\mathrm{T}^{a^j}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^j}},\tilde{\mathbb{x}}^j,\tilde{\mathbb{l}}^j\rangle\in\mathbf{Cõnf},\ \textit{and}\\
&\mathit{lck}\in\mathbf{Lck},
\end{aligned}$$

such that

$$
\begin{gathered}
0 \leq m \leq n, \\
c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^m \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^n, \\
0 \leq j, \\
\mathbf{Thrd}_{\tilde{c}^j} \subseteq \mathbf{Thrd}_{\tilde{c}^0} \subseteq \mathbf{Thrd}, \\
\forall i \in \{m, \ldots, n\} : \textsc{own}(\mathbb{l}^i\ lck) = \bot_{thrd}, \\
\textsc{rel}(\mathbb{l}^m\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^j\ lck)), \\
\exists \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}^j} : (\textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}^0) = [\texttt{lock}\ lck]^{pc_{\mathrm{T}}^0} \wedge t_{\mathrm{T}}^{a^0} \in \gamma_t(\tilde{t}_{\mathrm{T}}^{a^0}) \wedge \tilde{t}_{\mathrm{T}}^{a^j} = \tilde{t}_{\mathrm{T}}^{a^0} \wedge \\
pc_{\mathrm{T}}^{\tilde{c}^j} = pc_{\mathrm{T}}^{\tilde{c}^0} = pc_{\mathrm{T}}^n = pc_{\mathrm{T}}^0 \wedge \mathrm{T} \in \mathbf{Thrd}_{exe}^{c^n} \wedge \\
\forall i \in \{0, \ldots, n\} : \textsc{own}(\mathbb{l}^i\ lck) \neq \mathrm{T}), \\
\forall i \in \{m, \ldots, n-1\} : \forall \mathrm{T} \in \mathbf{Thrd}_{exe}^{c^i} : \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}^i) \neq [\texttt{lock}\ lck]^{pc_{\mathrm{T}}^i},
\end{gathered}
$$

where $\mathbf{Thrd}_{exe}^{c^i}$ *is as defined in Table 4.3 for* c^i, DLLOCK *satisfies:*

$$
\min(\{t_{\mathrm{T}}^{a^n} + \textsc{time}(c^n, \mathrm{T}) \mid \mathrm{T} \in \mathbf{Thrd}\}) \in \gamma_t(\textsc{dllock}(\tilde{c}^j, lck)) \qquad \square
$$

PROOF. Given the valid concrete configurations (c.f., Definition 4.4), abstract configurations and lock

$$
\begin{gathered}
c^0 @ \langle [\mathrm{T}, pc_{\mathrm{T}}^0, \mathbb{r}_{\mathrm{T}}^0, t_{\mathrm{T}}^{a^0}]_{\mathrm{T} \in \mathbf{Thrd}}, \mathbb{x}^0, \mathbb{l}^0 \rangle \in \mathbf{Conf}, \\
c^m @ \langle [\mathrm{T}, pc_{\mathrm{T}}^m, \mathbb{r}_{\mathrm{T}}^m, t_{\mathrm{T}}^{a^m}]_{\mathrm{T} \in \mathbf{Thrd}}, \mathbb{x}^m, \mathbb{l}^m \rangle \in \mathbf{Conf}, \\
c^n @ \langle [\mathrm{T}, pc_{\mathrm{T}}^n, \mathbb{r}_{\mathrm{T}}^n, t_{\mathrm{T}}^{a^n}]_{\mathrm{T} \in \mathbf{Thrd}}, \mathbb{x}^n, \mathbb{l}^n \rangle \in \mathbf{Conf}, \\
\tilde{c}^0 @ \langle [\mathrm{T}, pc_{\mathrm{T}}^{\tilde{c}^0}, \tilde{\mathbb{r}}_{\mathrm{T}}^0, \tilde{t}_{\mathrm{T}}^{a^0}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}^0}}, \tilde{\mathbb{x}}^0, \tilde{\mathbb{l}}^0 \rangle \in \tilde{\mathbf{Conf}}, \\
\tilde{c}^j @ \langle [\mathrm{T}, pc_{\mathrm{T}}^{\tilde{c}^j}, \tilde{\mathbb{r}}_{\mathrm{T}}^j, \tilde{t}_{\mathrm{T}}^{a^j}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}^j}}, \tilde{\mathbb{x}}^j, \tilde{\mathbb{l}}^j \rangle \in \tilde{\mathbf{Conf}}, \text{ and} \\
lck \in \mathbf{Lck},
\end{gathered}
$$

are such that

$$
\begin{gathered}
0 \leq m \leq n, \\
c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^m \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^n, \\
0 \leq j, \\
\mathbf{Thrd}_{\tilde{c}^j} \subseteq \mathbf{Thrd}_{\tilde{c}^0} \subseteq \mathbf{Thrd}, \\
\forall i \in \{m, \ldots, n\} : \textsc{own}(\mathbb{l}^i\ lck) = \bot_{thrd}, \\
\textsc{rel}(\mathbb{l}^m\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^j\ lck)), \\
\exists \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}^j} : (\textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}^0) = [\texttt{lock}\ lck]^{pc_{\mathrm{T}}^0} \wedge t_{\mathrm{T}}^{a^0} \in \gamma_t(\tilde{t}_{\mathrm{T}}^{a^0}) \wedge \tilde{t}_{\mathrm{T}}^{a^j} = \tilde{t}_{\mathrm{T}}^{a^0} \wedge \\
pc_{\mathrm{T}}^{\tilde{c}^j} = pc_{\mathrm{T}}^{\tilde{c}^0} = pc_{\mathrm{T}}^n = pc_{\mathrm{T}}^0 \wedge \mathrm{T} \in \mathbf{Thrd}_{exe}^{c^n} \wedge \\
\forall i \in \{0, \ldots, n\} : \textsc{own}(\mathbb{l}^i\ lck) \neq \mathrm{T}), \\
\forall i \in \{m, \ldots, n-1\} : \forall \mathrm{T} \in \mathbf{Thrd}_{exe}^{c^i} : \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}^i) \neq [\texttt{lock}\ lck]^{pc_{\mathrm{T}}^i},
\end{gathered}
$$

where $\mathbf{Thrd}^{c^i}_{exe}$ is as defined in Table 4.3 for c^i. First note that:

- Since $\forall i \in \{m,\ldots,n-1\} : \forall \mathrm{T} \in \mathbf{Thrd}^{c^i}_{exe} : (\textsc{stm}(\mathrm{T}, pc^i_{\mathrm{T}}) \neq [\texttt{lock}\ lck]^{pc^i_{\mathrm{T}}} \wedge \textsc{own}(\mathbb{l}^i\ lck) = \bot_{thrd})$, it must be that $\textsc{rel}(\mathbb{l}^n\ lck) = \textsc{rel}(\mathbb{l}^m\ lck)$ (Tables 4.2 and 4.3).

- Since $\forall i \in \{m,\ldots,n\} : \textsc{own}(\mathbb{l}^i\ lck) = \bot_{thrd}$ and $\textsc{rel}(\mathbb{l}^n\ lck) = \textsc{rel}(\mathbb{l}^m\ lck)$, it must be that $\forall \mathrm{T} \in \mathbf{Thrd} : t^{a^m}_{\mathrm{T}} \leq \textsc{rel}(\mathbb{l}^n\ lck)$ (Tables 4.2 and 4.3 and Lemma 4.2).

- Since time only moves forward (Lemma 4.2), it must be that for c^n, $\forall \mathrm{T} \in \mathbf{Thrd} : \textsc{rel}(\mathbb{l}^n\ lck) \leq t^{a^n}_{\mathrm{T}} + \textsc{time}(c^n, \mathrm{T})$.

- Since $\forall i \in \{m,\ldots,n-1\} : \forall \mathrm{T} \in \mathbf{Thrd}^{c^i}_{exe} : \textsc{stm}(\mathrm{T}, pc^i_{\mathrm{T}}) \neq [\texttt{lock}\ lck]^{pc^i_{\mathrm{T}}}$, it must be that $\forall \mathrm{T} \in \mathbf{Thrd} : (\textsc{stm}(\mathrm{T}, pc^n_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc^n_{\mathrm{T}}} \Rightarrow t^{a^n}_{\mathrm{T}} = t^{a^m}_{\mathrm{T}})$ (Tables 4.2 and 4.3).

- Since $\exists \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}^j} : (\textsc{stm}(\mathrm{T}, pc^0_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc^0_{\mathrm{T}}} \wedge t^{a^0}_{\mathrm{T}} \in \gamma_t(\tilde{t}^{a^0}_{\mathrm{T}}) \wedge \tilde{t}^{a^j}_{\mathrm{T}} = \tilde{t}^{a^0}_{\mathrm{T}} \wedge pc^{\tilde{c}^j}_{\mathrm{T}} = pc^{\tilde{c}^0}_{\mathrm{T}} = pc^n_{\mathrm{T}} = pc^0_{\mathrm{T}} \wedge \mathrm{T} \in \mathbf{Thrd}^{c^n}_{exe} \wedge \forall i \in \{0,\ldots,n\} : \textsc{own}(\mathbb{l}^i\ lck) \neq \mathrm{T})$, $\mathbf{Thrd}_{\tilde{c}^j} \subseteq \mathbf{Thrd}$, $\forall \mathrm{T} \in \mathbf{Thrd} : t^{a^m}_{\mathrm{T}} \leq \textsc{rel}(\mathbb{l}^n\ lck)$, $\forall \mathrm{T} \in \mathbf{Thrd} : \textsc{rel}(\mathbb{l}^n\ lck) \leq t^{a^n}_{\mathrm{T}} + \textsc{time}(c^n, \mathrm{T})$ and $\forall \mathrm{T} \in \mathbf{Thrd} : (\textsc{stm}(\mathrm{T}, pc^n_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc^n_{\mathrm{T}}} \Rightarrow t^{a^n}_{\mathrm{T}} = t^{a^m}_{\mathrm{T}})$ it must be that $\exists \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}^j} : t^{a^m}_{\mathrm{T}} \leq \textsc{rel}(\mathbb{l}^n\ lck) \leq t^{a^m}_{\mathrm{T}} + \textsc{time}(c^n, \mathrm{T}) \wedge \textsc{stm}(\mathrm{T}, pc^n_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc^n_{\mathrm{T}}} \wedge t^{a^0}_{\mathrm{T}} \in \gamma_t(\tilde{t}^{a^0}_{\mathrm{T}}) \wedge \tilde{t}^{a^j}_{\mathrm{T}} = \tilde{t}^{a^0}_{\mathrm{T}} \wedge pc^{\tilde{c}^j}_{\mathrm{T}} = pc^{\tilde{c}^0}_{\mathrm{T}} = pc^n_{\mathrm{T}} = pc^0_{\mathrm{T}} \wedge \mathrm{T} \in \mathbf{Thrd}^{c^n}_{exe} \wedge \forall i \in \{0,\ldots,n\} : \textsc{own}(\mathbb{l}^i\ lck) \neq \mathrm{T})$.

From here on, it will be assumed that $\mathrm{T}' \in \mathbf{Thrd}_{\tilde{c}^j}$ is one of the threads such that $\textsc{stm}(\mathrm{T}', pc^0_{\mathrm{T}'}) = [\texttt{lock}\ lck]^{pc^0_{\mathrm{T}'}} \wedge t^{a^m}_{\mathrm{T}'} \leq \textsc{rel}(\mathbb{l}^n\ lck) \leq t^{a^m}_{\mathrm{T}'} + \textsc{time}(c^n, \mathrm{T}') \wedge t^{a^0}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a^0}_{\mathrm{T}'}) \wedge \tilde{t}^{a^j}_{\mathrm{T}'} = \tilde{t}^{a^0}_{\mathrm{T}'} \wedge pc^{\tilde{c}^j}_{\mathrm{T}'} = pc^{\tilde{c}^0}_{\mathrm{T}'} = pc^n_{\mathrm{T}'} = pc^0_{\mathrm{T}'} \wedge \mathrm{T}' \in \mathbf{Thrd}^{c^n}_{exe} \wedge \forall i \in \{0,\ldots,n\} : \textsc{own}(\mathbb{l}^i\ lck) \neq \mathrm{T}'))$.

- Let $\{m_1,\ldots,m_2\}$ be the set of indices, such that $0 \leq m_1 \leq m_2 < m$, $\forall i \in \{m_1,\ldots,m_2\} : \mathrm{T}' \in \mathbf{Thrd}^{c^i}_{exe}$ and $\forall i \in \{0,\ldots,m\} \setminus \{m_1,\ldots,m_2\} : \mathrm{T}' \notin \mathbf{Thrd}^{c^i}_{exe}$, where $\mathbf{Thrd}^{c^i}_{exe}$ is as defined in Table 4.3 for c^i (note that it is possible that $\{m_1,\ldots,m_2\} = \emptyset$; it should also be noted that $\mathrm{T}' \notin \mathbf{Thrd}^{c^m}_{exe}$). Since $\mathbf{Thrd}_{\tilde{c}^j} \subseteq \mathbf{Thrd}$, $t^{a^0}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a^0}_{\mathrm{T}'})$, $\tilde{t}^{a^j}_{\mathrm{T}'} = \tilde{t}^{a^0}_{\mathrm{T}'}$, $c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^m$ and $0 \leq m$, it is easy to see that every configuration, $\tilde{c}'$, created by the **repeat**-loop within $\textsc{dllock}(\tilde{c}^j, lck)$ fulfills the assumptions of

Lemma 5.52. Furthermore, it is easy to see that (according to Lemma 5.52)

$$t_{\mathrm{T}'}^{a^m} \in \gamma_t(\tilde{t}_{\mathrm{T}'}^{a^0} \tilde{+}_t \textsc{absTime}(\tilde{c}^{m_1}, \mathrm{T}') \tilde{+}_t \ldots \tilde{+}_t \textsc{absTime}(\tilde{c}^{m_2}, \mathrm{T}'))$$

where $\tilde{c}^{m_1}, \ldots, \tilde{c}^{m_2}$ correspond to a $\tilde{c}'$ derived by the **repeat**-loop (in total, there are $|\{m_1, \ldots, m_2\}|$ $\tilde{c}'$-configurations for the expression above).

For the sake of readability, let

$$\tilde{t}_{\mathrm{T}'}^{a''} = \tilde{t}_{\mathrm{T}'}^{a^0} \tilde{+}_t \textsc{absTime}(\tilde{c}^{m_1}, \mathrm{T}') \tilde{+}_t \ldots \tilde{+}_t \textsc{absTime}(\tilde{c}^{m_2}, \mathrm{T}')$$
$$\tilde{c}'' = \langle [\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, (\mathrm{T} = \mathrm{T}' \ ? \ \tilde{t}_{\mathrm{T}'}^{a''} : \tilde{t}_{\mathrm{T}}^{a^j})]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle$$

where $\tilde{c}^{m_1}, \ldots, \tilde{c}^{m_2}$ are defined as in the bullet above.

Assuming that $\tilde{t}_{dl}$ is safe at the start of each iteration of the **for all** $\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}$-loop, where T is such that $\textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}^n) = [\texttt{lock}\ lck]^{pc_{\mathrm{T}}^n}$, it should be shown that $\min(\{t_{\mathrm{T}}^{a^n} + \textsc{time}(c^n, \mathrm{T}) \mid \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}\}) \in \gamma_t(\tilde{t}_{dl})$ is always fulfilled at the end of each loop-iteration. It is easy to see that the initial value of $\tilde{t}_{dl}$ (i.e., $\tilde{\top}_t$) is trivially safe since $\forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : t_{\mathrm{T}}^{a^n} + \textsc{time}(c^n, \mathrm{T}) \in \gamma_t(\tilde{\top}_t)$. Note that $\forall c \in \mathbf{Conf} : \forall \mathrm{T} \in \mathbf{Thrd} : \textsc{time}(c, \mathrm{T}) \geq 0$ (Assumption 4.1). Several cases need to be considered.

1. If $\textsc{time}(c^n, \mathrm{T}') = 0$, then $t_{\mathrm{T}'}^{a^n} + \textsc{time}(c^n, \mathrm{T}') = t_{\mathrm{T}'}^{a^m} = \textsc{rel}(\mathbb{l}^n\ lck)$ (remember that $t_{\mathrm{T}'}^{a^n} = t_{\mathrm{T}'}^{a^m}$ and $\textsc{rel}(\mathbb{l}^n\ lck) = \textsc{rel}(\mathbb{l}^m\ lck)$) and $0 \in \textsc{absTime}(\tilde{c}'', \mathrm{T}')$ (Assumption 5.50). Since $0 \in \gamma_t(\textsc{absTime}(\tilde{c}'', \mathrm{T}'))$, it will not be possible to determine a $\tilde{t}_{\mathrm{T}'}^{a'}$ such that $\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^j\ lck) \tilde{<}_t \tilde{t}_{\mathrm{T}'}^{a'}$. However, for the iteration of the **repeat**-loop for which $0 \in \textsc{absTime}(\tilde{c}', \mathrm{T}')$, it must be that $\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^j\ lck) \not\tilde{<}_t \tilde{t}_{\mathrm{T}'}^{a'}$. $\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^j\ lck)$ provides a safe base for when T' would acquire lck since $\textsc{rel}(\mathbb{l}^n\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^j\ lck))$. Thus, according to Assumption 5.50, it must be that $(t_{\mathrm{T}'}^{a^n} + \textsc{time}(c^n, \mathrm{T}')) \in \gamma_t((\tilde{t} \tilde{+}_t \textsc{absTime}(\tilde{c}''', \mathrm{T}')) \tilde{\sqcup}_t \alpha_t(\{-\infty\}))$, where $\tilde{c}''' = \langle [\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, (\mathrm{T} = \mathrm{T}' \ ? \ \tilde{t} : \tilde{t}_{\mathrm{T}}^{a^j})]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle$ and $\tilde{t} = \tilde{t}_{\mathrm{T}'}^{a'} \tilde{\sqcup}_t \tilde{\textsc{rel}}(\tilde{\mathbb{l}}^j\ lck)$. But then, it is easy to see that $\min(\{t_{\mathrm{T}}^{a^n} + \textsc{time}(c^n, \mathrm{T}) \mid \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}\}) \in \gamma_t(((\tilde{t} \tilde{+}_t \textsc{absTime}(\tilde{c}''', \mathrm{T}')) \tilde{\sqcup}_t \alpha_t(\{-\infty\})) \tilde{\sqcap}_t \tilde{t}_{dl})$.

2. If $\textsc{time}(c^n, \mathrm{T}') > 0$, then two cases must be considered.

   (a) If $0 \in \gamma_t(\textsc{absTime}(\tilde{c}'', \mathrm{T}'))$ (or for any $\tilde{c}'$ of the **repeat**-loop), it will not be possible to determine a $\tilde{t}_{\mathrm{T}'}^{a'}$ such that $\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^j\ lck) \tilde{<}_t \tilde{t}_{\mathrm{T}'}^{a'}$. However, this proof is the same as that of 1 above.

(b) If $0 \notin \gamma_t(\text{ABSTIME}(\tilde{c}'', \mathrm{T}'))$ (as well as for all $\tilde{c}'$ of the **repeat**-loop), then a $\tilde{t}^{a\prime}_{\mathrm{T}'}$ such that $\tilde{\text{REL}}(\tilde{\mathbb{I}}^j\ lck) \tilde{<}_t \tilde{t}^{a\prime}_{\mathrm{T}'}$ can be derived. Since $t^{a^m}_{\mathrm{T}'} \leq \text{REL}(\mathbb{I}^n\ lck)$, $\text{REL}(\mathbb{I}^n\ lck) \in \gamma_t(\tilde{\text{REL}}(\tilde{\mathbb{I}}^j\ lck))$ and $t^{a^m}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a\prime\prime}_{\mathrm{T}'})$, it must be that $\tilde{\text{REL}}(\tilde{\mathbb{I}}^j\ lck) \not\tilde{<}_t \tilde{t}^{a\prime\prime}_{\mathrm{T}'}$, and thus, the **repeat**-loop will iterate *at least* once more when $\tilde{t}^{a\prime}_{\mathrm{T}'} = \tilde{t}^{a\prime\prime}_{\mathrm{T}'}$. Since $\text{REL}(\mathbb{I}^n\ lck) < t^{a^m}_{\mathrm{T}'} + \text{TIME}(c^n, \mathrm{T}')$, it must be that (since $t^{a^n}_{\mathrm{T}'} = t^{a^m}_{\mathrm{T}'}$) $t^{a^n}_{\mathrm{T}'} + \text{TIME}(c^n, \mathrm{T}) \in \gamma_t(\tilde{t}^{a\prime\prime}_{\mathrm{T}'} \tilde{+}_t \text{ABSTIME}(\tilde{c}'', \mathrm{T}'))$. Thus, $t^{a^n}_{\mathrm{T}'} + \text{TIME}(c^n, \mathrm{T}) \in \gamma_t(\tilde{t}^{a\prime}_{\mathrm{T}'} \tilde{\sqcup}_t \alpha_t(\{-\infty\}))$, where $\tilde{t}^{a\prime}_{\mathrm{T}'}$ and $\tilde{c}'$ are derived from $\tilde{c}''$ by the **repeat**-loop, and thus $\max(\gamma_t(\tilde{t}^{a\prime\prime}_{\mathrm{T}'})) \leq \max(\gamma_t(\tilde{t}^{a\prime}_{\mathrm{T}'}))$ (c.f., Assumption 5.50). But then, it is easy to see that $\min(\{t^{a^n}_{\mathrm{T}} + \text{TIME}(c^n, \mathrm{T}) \mid \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}\}) \in \gamma_t((\tilde{t}^{a\prime}_{\mathrm{T}'} \tilde{\sqcup}_t \alpha_t(\{-\infty\})) \tilde{\sqcap}_t \tilde{t}_{dl})$.

Thus, it must be that:

$$\min(\{t^{a^n}_{\mathrm{T}} + \text{TIME}(c^n, \mathrm{T}) \mid \mathrm{T} \in \mathbf{Thrd}\}) \in \gamma_t(\text{DLLOCK}(\tilde{c}^j, lck)) \qquad \blacksquare$$

The accumulated time, $\tilde{t}^{a\prime}_{\mathrm{T}} \in \tilde{\mathbf{Time}}$, for a thread, $\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}$, is determined using ACCTIME, defined in Algorithm 5.12, which is partially a safe approximation of the concrete accumulated time of T (Lemma 5.54). This is because the way that time accumulates for threads executing `lock` *lck* for some lock, $lck \in \mathbf{Lck}$, that is currently acquired by some other thread differs in the concrete and abstract semantics. In the concrete semantics, the `lock`-statement is just considered to finish its execution, without successfully acquiring *lck*, after the (relative) time given by TIME, then a new instance of the same `lock`-statement is executed (c.f., Tables 4.2 and 4.3); i.e., the thread is actively spinning on the lock.

In the abstract semantics (c.f., Tables 5.5 and 5.6 and Algorithms 5.11 and 5.12), a thread issuing `lock` *lck* for some lock, $lck \in \mathbf{Lck}$, that is currently acquired by some other thread would be frozen until it is assigned *lck*, if this ever occurs; i.e., the thread's accumulated time would not be increased while it is waiting to be assigned *lck*. When (and if) the thread is later assigned *lck*, its accumulated execution time is advanced based on when *lck* became free (*unlocked*).

If the lock, $lck \in \mathbf{Lck}$, is not currently assigned to some other thread when some thread issues `lock` *lck*, the behavior is the same in both the concrete and abstract semantics in case the `lock`-issuing thread successfully acquires *lck*; i.e., the thread's execution time will be accumulated based on TIME and ABSTIME, respectively.

NOTE. ACCTIME *is not directly safe for the case that* $\text{STM}(\text{T}, pc_\text{T}) = [\texttt{lock}\ lck]^{pc_\text{T}} \wedge \text{OWN}(\mathbb{l}''\ lck) \neq \text{T}$. *In the concrete case,* T *will be executed in a spin-lock fashion, while in the corresponding abstract case,* T *will be frozen (i.e., its accumulated time will not be updated). This case is further considered in the proof of Lemma 5.57.*

ACCTIME *is also not directly safe for the case that* T *has been waiting but is now assigned lck, i.e., a catchup will be performed in the abstract case, since there could be an extra abstract configuration such that* $\tilde{t}^a_\text{T} \tilde{<}_t \text{RE}\tilde{\text{L}}(\tilde{\mathbb{l}}'', lck)$ *and* $\tilde{t}^a_\text{T} \tilde{+}_t \text{ABSTIME}(\tilde{c}^0, \text{T}) \tilde{\not>}_t \text{RE}\tilde{\text{L}}(\tilde{\mathbb{l}}'', lck)$. *This case is also further considered in the proof of Lemma 5.57.*

Lemma 5.54 (Partial soundness of ACCTIME):
Given the valid concrete configuration $c@\langle[\text{T}', pc_{\text{T}'}, \mathbb{r}_{\text{T}'}, t^a_{\text{T}'}]_{\text{T}' \in \mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle \in$ **Conf** *(c.f., Definition 4.4), the abstract configuration* $\tilde{c}^0@\langle[\text{T}', pc^{\tilde{c}}_{\text{T}'}, \tilde{\mathbb{r}}_{\text{T}'}, \tilde{t}^a_{\text{T}'}]_{\text{T}' \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \tilde{\mathbf{Conf}}$, *and some thread,* $\text{T} \in \mathbf{Thrd}_{\tilde{c}}$, *such that*

$$
\begin{array}{l}
\mathbf{Thrd}_{\tilde{c}} \subseteq \mathbf{Thrd} \wedge \\
pc_\text{T} = pc^{\tilde{c}}_\text{T} \wedge \\
t^a_\text{T} \in \gamma_t(\tilde{t}^a_\text{T}) \wedge \\
(\text{T} \in \mathbf{Thrd}^c_{exe} \wedge \forall lck \in \mathbf{Lck} : (\text{STM}(\text{T}, pc_\text{T}) = [\texttt{lock}\ lck]^{pc_\text{T}} \Rightarrow \\
\qquad (\text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}''\ lck) = \text{T} \wedge \text{OWN}(\mathbb{l}''\ lck) = \text{T}))) \Longleftrightarrow \text{T} \in \mathbf{Thrd}^{\tilde{c}}_{exe} \wedge \\
\forall lck \in \mathbf{Lck} : (\text{OWN}(\mathbb{l}''\ lck) = \text{T} \Rightarrow (\text{OWN}(\mathbb{l}''\ lck) = \text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}''\ lck) \wedge \\
\qquad\qquad\qquad\qquad \text{DL}(\mathbb{l}''\ lck) \in \gamma_t(\text{D}\tilde{\text{L}}(\tilde{\mathbb{l}}''\ lck)) \wedge \\
\qquad\qquad\qquad\qquad \text{POWN}(\mathbb{l}''\ lck) = \text{PO}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}''\ lck) \wedge \\
\qquad\qquad\qquad\qquad \text{REL}(\mathbb{l}''\ lck) \in \gamma_t(\text{RE}\tilde{\text{L}}(\tilde{\mathbb{l}}''\ lck)) \wedge \\
\qquad\qquad\qquad\qquad \min(\gamma_t(\text{D}\tilde{\text{L}}(\tilde{\mathbb{l}}''\ lck))) = -\infty)),
\end{array}
$$

where $\mathbf{Thrd}^c_{exe}$ *and* $\mathbb{l}''$, *and* $\mathbf{Thrd}^{\tilde{c}}_{exe}$ *and* $\tilde{\mathbb{l}}''$, *are as defined in Tables 4.3 and 5.6, respectively,*

$$t^{a\prime}_\text{T} \in \gamma_t(\text{ACCTIME}(\langle[\text{T}', pc^{\tilde{c}}_{\text{T}'}, \tilde{\mathbb{r}}_{\text{T}'}, \tilde{t}^a_{\text{T}'}]_{\text{T}' \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}''\rangle, \mathbf{Thrd}^{\tilde{c}}_{exe}, \text{T}))$$

where $t^{a\prime}_\text{T}$ *is as defined in Table 4.3.* □

PROOF. Assume that the (valid; c.f., Definition 4.4) configurations $c@\langle[\text{T}', pc_{\text{T}'}, \mathbb{r}_{\text{T}'}, t^a_{\text{T}'}]_{\text{T}' \in \mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle \in$ **Conf** and $\tilde{c}^0@\langle[\text{T}', pc^{\tilde{c}}_{\text{T}'}, \tilde{\mathbb{r}}_{\text{T}'}, \tilde{t}^a_{\text{T}'}]_{\text{T}' \in \mathbf{Thrd}_{\tilde{c}}}$,

$\tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \mathbf{C\tilde{o}nf}$ and the thread $\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}$ are such that

$$
\begin{array}{l}
\mathbf{Thrd}_{\tilde{c}} \subseteq \mathbf{Thrd} \wedge \\
pc_{\mathrm{T}} = pc^{\tilde{c}}_{\mathrm{T}} \wedge \\
t^{a}_{\mathrm{T}} \in \gamma_t(\tilde{t}^{a}_{\mathrm{T}}) \wedge \\
(\mathrm{T} \in \mathbf{Thrd}^{c}_{exe} \wedge \forall lck \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc_{\mathrm{T}}} \Rightarrow \\
\qquad (\textsc{o}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}''\ lck) = \mathrm{T} \wedge \textsc{own}(\mathbb{l}''\ lck) = \mathrm{T}))) \Longleftrightarrow \mathrm{T} \in \mathbf{Thrd}^{\tilde{c}}_{exe} \wedge \\
\forall lck \in \mathbf{Lck} : (\textsc{own}(\mathbb{l}''\ lck) = \mathrm{T} \Rightarrow (\textsc{own}(\mathbb{l}''\ lck) = \textsc{o}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}''\ lck) \wedge \\
\qquad\qquad \textsc{dl}(\mathbb{l}''\ lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}''\ lck)) \wedge \\
\qquad\qquad \textsc{powned}(\mathbb{l}''\ lck) = \textsc{po}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}''\ lck) \wedge \\
\qquad\qquad \textsc{rel}(\mathbb{l}''\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}''\ lck)) \wedge \\
\qquad\qquad \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}''\ lck))) = -\infty)),
\end{array}
$$

where $\mathbf{Thrd}^{c}_{exe}$ and $\mathbb{l}''$, and $\mathbf{Thrd}^{\tilde{c}}_{exe}$ and $\tilde{\mathbb{l}}''$, are as defined in Tables 4.3 and 5.6, respectively.

For the sake of readability, let $\tilde{c} = \langle [\mathrm{T}', pc^{\tilde{c}}_{\mathrm{T}'}, \tilde{\mathbb{r}}_{\mathrm{T}'}, \tilde{t}^{a}_{\mathrm{T}'}]_{\mathrm{T}' \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}'' \rangle$ when considering the following cases. Note that $\tilde{\mathbf{Time}} = \mathbf{Intv}$.

1. If $\mathrm{T} \notin \mathbf{Thrd}^{c}_{exe}$ (and thus, $\mathrm{T} \notin \mathbf{Thrd}^{\tilde{c}}_{exe}$), then $t^{a\prime}_{\mathrm{T}} = t^{a}_{\mathrm{T}}$ (Table 4.3) and $\textsc{accTime}(\tilde{c}, \mathbf{Thrd}^{\tilde{c}}_{exe}, \mathrm{T}) = \tilde{t}^{a}_{\mathrm{T}}$. Thus, $t^{a\prime}_{\mathrm{T}} \in \gamma_t(\textsc{accTime}(\tilde{c}, \mathbf{Thrd}^{\tilde{c}}_{exe}, \mathrm{T}))$.

2. If $\mathrm{T} \in \mathbf{Thrd}^{c}_{exe}$ and for some $a \in \mathbf{Aexp}$, $b \in \mathbf{Bexp}$, $l \in \mathbf{Lbl}_{\mathrm{T}}$, $r \in \mathbf{Reg}_{\mathrm{T}}$, $x \in \mathbf{Var}$ and $lck \in \mathbf{Lck}$, $\textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) \in \{[\texttt{skip}]^{pc_{\mathrm{T}}}, [r := a]^{pc_{\mathrm{T}}}, [\texttt{if}\ b\ \texttt{goto}\ l]^{pc_{\mathrm{T}}}, [\texttt{store}\ r\ \texttt{to}\ x]^{pc_{\mathrm{T}}}, [\texttt{load}\ r\ \texttt{from}\ x]^{pc_{\mathrm{T}}}, [\texttt{unlock}\ lck]^{pc_{\mathrm{T}}}\}$, i.e., $\forall lck' \in \mathbf{Lck} : \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) \neq [\texttt{lock}\ lck']^{pc_{\mathrm{T}}}$ (and thus, $\mathrm{T} \in \mathbf{Thrd}^{\tilde{c}}_{exe}$ since $(\mathrm{T} \in \mathbf{Thrd}^{c}_{exe} \wedge \forall lck' \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{lock}\ lck']^{pc_{\mathrm{T}}} \Rightarrow (\textsc{o}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}''\ lck') = \mathrm{T} \wedge \textsc{own}(\mathbb{l}''\ lck') = \mathrm{T}))) \Longleftrightarrow \mathrm{T} \in \mathbf{Thrd}^{\tilde{c}}_{exe})$, then $\textsc{accTime}(\tilde{c}, \mathbf{Thrd}^{\tilde{c}}_{exe}, \mathrm{T}) = \tilde{t}^{a}_{\mathrm{T}} \mathbin{\tilde{+}_t} \textsc{absTime}(\tilde{c}, \mathrm{T})$ and by Table 4.3, $t^{a\prime}_{\mathrm{T}} = t^{a}_{\mathrm{T}} + \textsc{time}(c, \mathrm{T})$. Thus, by Lemma 5.51, $t^{a\prime}_{\mathrm{T}} \in \gamma_t(\textsc{accTime}(\tilde{c}, \mathbf{Thrd}^{\tilde{c}}_{exe}, \mathrm{T}))$.

3. If $\mathrm{T} \in \mathbf{Thrd}^{c}_{exe}$ and for some $lck \in \mathbf{Lck}$, $\textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc_{\mathrm{T}}}$ and $\textsc{own}(\mathbb{l}''\ lck) = \mathrm{T}$, and thus, $\mathrm{T} \in \mathbf{Thrd}^{\tilde{c}}_{exe}$ since $(\mathrm{T} \in \mathbf{Thrd}^{c}_{exe} \wedge \forall lck' \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{lock}\ lck']^{pc_{\mathrm{T}}} \Rightarrow (\textsc{o}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}''\ lck') = \mathrm{T} \wedge \textsc{own}(\mathbb{l}''\ lck') = \mathrm{T}))) \Longleftrightarrow \mathrm{T} \in \mathbf{Thrd}^{\tilde{c}}_{exe}$, then several cases need to be considered. Note that $\min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}''\ lck))) = -\infty$, since $\forall lck' \in \mathbf{Lck} : (\textsc{own}(\mathbb{l}''\ lck') = \mathrm{T} \Rightarrow \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}''\ lck'))) = -\infty)$ and $\textsc{own}(\mathbb{l}''\ lck) = \mathrm{T}$, and that T cannot acquire lck at any time, $\tilde{t}$, such that $\tilde{t} \mathbin{\tilde{<}_t} \tilde{\textsc{rel}}(\tilde{\mathbb{l}}''\ lck)$, since lck has not been released at $\tilde{t}$, or $\tilde{\textsc{dl}}(\tilde{\mathbb{l}}''\ lck) \mathbin{\tilde{<}_t} \tilde{t}$, since by then some other thread would have taken lck (c.f., Tables 5.5 and 5.6).

(a) If $\widetilde{\text{STT}}(\tilde{\mathbb{l}}''\ lck) = locked$ (and $\text{STT}(\mathbb{l}''\ lck) = locked$ since c is valid and $\text{OWN}(\mathbb{l}''\ lck) \neq \bot_{thrd}$), then $t_{\text{T}}^{a\prime} = t_{\text{T}}^{a} + \text{TIME}(c,\text{T})$ and $\text{ACCTIME}(\tilde{c},\mathbf{Thrd}_{exe}^{\tilde{c}},\text{T}) = \tilde{t}_{\text{T}}^{a} \mathbin{\tilde{+}_t} \text{ABSTIME}(\tilde{c},\text{T})$. Thus, $t_{\text{T}}^{a\prime} \in \gamma_t(\text{ACCTIME}(\tilde{c},\mathbf{Thrd}_{exe}^{\tilde{c}},\text{T}))$ (Lemma 5.51).

(b) Assume that $\widetilde{\text{STT}}(\tilde{\mathbb{l}}''\ lck) = unlocked \wedge \widetilde{\text{DL}}(\tilde{\mathbb{l}}''\ lck) \mathbin{\tilde{\leq}_t} (\tilde{t}_{\text{T}}^{a} \mathbin{\tilde{+}_t} \text{ABSTIME}(\tilde{c},\text{T}))$. Then, in the concrete case, it must be that T cannot be the thread acquiring *lck* since $\text{DL}(\mathbb{l}''\ lck) \in \gamma_t(\widetilde{\text{DL}}(\tilde{\mathbb{l}}''\ lck))$, $t_{\text{T}}^{a} \in \gamma_t(\tilde{t}_{\text{T}}^{a})$, $\text{TIME}(c,\text{T}) \in \gamma_t(\text{ABSTIME}(\tilde{c},\text{T}))$ and $t_{\text{T}}^{a} + \text{TIME}(c,\text{T}) = \text{DL}(\mathbb{l}''\ lck)$ whenever T acquires *lck* (Tables 4.2 and 4.3). But, then it cannot be that $\widetilde{\text{STT}}(\tilde{\mathbb{l}}''\ lck) = unlocked \wedge \widetilde{\text{DL}}(\tilde{\mathbb{l}}''\ lck) \mathbin{\tilde{\leq}_t} (\tilde{t}_{\text{T}}^{a} \mathbin{\tilde{+}_t} \text{ABSTIME}(\tilde{c},\text{T}))$ since in the concrete case, T does successfully acquire *lck*, which means that the corresponding branch cannot apply for the given case. (Note that such a $\tilde{c}$ will not be further considered; c.f., Algorithm 6.6 and Tables 5.5 and 5.6.)

(c) Note that the $\widetilde{\text{STT}}(\tilde{\mathbb{l}}''\ lck) = unlocked \wedge \widetilde{\text{DL}}(\tilde{\mathbb{l}}''\ lck) \mathbin{\tilde{\not\leq}_t} (\tilde{t}_{\text{T}}^{a} \mathbin{\tilde{+}_t} \text{ABSTIME}(\tilde{c},\text{T})) \wedge (\tilde{t}_{\text{T}}^{a} \mathbin{\tilde{+}_t} \text{ABSTIME}(\tilde{c},\text{T})) \mathbin{\tilde{\leq}_t} \widetilde{\text{REL}}(\tilde{\mathbb{l}}''\ lck)$ conditioned branch, which applies to cases where T has been frozen for sure while waiting to acquire *lck* but has now been assigned *lck*, cannot be taken either. To see this, note that since c is valid, it must be that $\text{REL}(\mathbb{l}''\ lck) \leq t_{\text{T}}^{a} + \text{TIME}(c,\text{T})$ (Definition 4.4). Then, since $t_{\text{T}}^{a} \in \gamma_t(\tilde{t}_{\text{T}}^{a})$, $\text{TIME}(c,\text{T}) \in \gamma_t(\text{ABSTIME}(\tilde{c}^0,\text{T}))$ (c.f., Assumption 5.50), $\text{OWN}(\mathbb{l}''\ lck) = \text{T}$ and $\text{OWN}(\mathbb{l}''\ lck) = \text{T} \Rightarrow \text{REL}(\mathbb{l}''\ lck) \in \gamma_t(\widetilde{\text{REL}}(\tilde{\mathbb{l}}''\ lck))$, it must be that $\tilde{t}_{\text{T}}^{a} \mathbin{\tilde{+}_t} \text{ABSTIME}(\tilde{c}^0,\text{T}) \mathbin{\tilde{\not\leq}_t} \widetilde{\text{REL}}(\tilde{\mathbb{l}}''\ lck)$. This branch is further considered when the freezing of threads is proven to be safe (c.f., the proof of Lemma 5.57).

(d) If $\widetilde{\text{STT}}(\tilde{\mathbb{l}}''\ lck) = unlocked \wedge \widetilde{\text{DL}}(\tilde{\mathbb{l}}''\ lck) \mathbin{\tilde{\not\leq}_t} (\tilde{t}_{\text{T}}^{a} \mathbin{\tilde{+}_t} \text{ABSTIME}(\tilde{c},\text{T})) \wedge (\tilde{t}_{\text{T}}^{a} \mathbin{\tilde{+}_t} \text{ABSTIME}(\tilde{c},\text{T})) \mathbin{\tilde{\not\leq}_t} \widetilde{\text{REL}}(\tilde{\mathbb{l}}''\ lck) \wedge (\widetilde{\text{POWN}}(\tilde{\mathbb{l}}''\ lck) = \text{T} \vee \widetilde{\text{REL}}(\tilde{\mathbb{l}}''\ lck) \mathbin{\tilde{\leq}_t} (\tilde{t}_{\text{T}}^{a} \mathbin{\tilde{+}_t} \text{ABSTIME}(\tilde{c},\text{T})))$, then two cases must be considered.

   i. If $\widetilde{\text{POWN}}(\tilde{\mathbb{l}}''\ lck) = \text{T}$, then the sequential execution of the statements of a thread (c.f., Tables 4.2 and 4.3) gives that T must acquire *lck* at $\tilde{t}_{\text{T}}^{a} \mathbin{\tilde{+}_t} \text{ABSTIME}(\tilde{c},\text{T})$, but not at a point in time, $\tilde{t}$, such that $\widetilde{\text{DL}}(\tilde{\mathbb{l}}''\ lck) \mathbin{\tilde{\leq}_t} \tilde{t}$, because by then, some other thread must have already acquired *lck* (since $\text{DL}(\mathbb{l}''\ lck) \in \gamma_t(\widetilde{\text{DL}}(\tilde{\mathbb{l}}''\ lck))$). Thus, it must be that $t_{\text{T}}^{a\prime} \in \gamma_t((\tilde{t}_{\text{T}}^{a} \mathbin{\tilde{+}_t} \text{ABSTIME}(\tilde{c},\text{T})) \mathbin{\tilde{\sqcap}_t} \widetilde{\text{DL}}(\tilde{\mathbb{l}}''\ lck))$.

   ii. If $\widetilde{\text{REL}}(\tilde{\mathbb{l}}''\ lck) \mathbin{\tilde{\leq}_t} (\tilde{t}_{\text{T}}^{a} \mathbin{\tilde{+}_t} \text{ABSTIME}(\tilde{c},\text{T}))$, then Lemma 5.51

gives that $t_{\mathrm{T}}^{a\prime} \in \gamma_t((\tilde{t}_{\mathrm{T}}^{a} \mathbin{\tilde{+}_t} \textsc{absTime}(\tilde{c},\mathrm{T})) \mathbin{\tilde{\sqcap}_t} \tilde{\textsc{dl}}(\tilde{\mathbb{l}}'' \mathit{lck}))$ since $t_{\mathrm{T}}^{a\prime} = \textsc{dl}(\mathbb{l}'' \mathit{lck})$ (c.f., Tables 4.2 and 4.3), $\textsc{dl}(\mathbb{l}'' \mathit{lck}) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}'' \mathit{lck}))$ and $\textsc{rel}(\mathbb{l}'' \mathit{lck}) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}'' \mathit{lck}))$.

(e) If $\tilde{\textsc{stt}}(\tilde{\mathbb{l}}'' \mathit{lck}) = \mathit{unlocked} \wedge \tilde{\textsc{dl}}(\tilde{\mathbb{l}}'' \mathit{lck}) \not\tilde{<}_t (\tilde{t}_{\mathrm{T}}^{a} \mathbin{\tilde{+}_t} \textsc{absTime}(\tilde{c},\mathrm{T})) \wedge (\tilde{t}_{\mathrm{T}}^{a} \mathbin{\tilde{+}_t} \textsc{absTime}(\tilde{c},\mathrm{T})) \mathbin{\tilde{\sqcap}_t} \tilde{\textsc{rel}}(\tilde{\mathbb{l}}'' \mathit{lck}) \neq \tilde{\bot}_t \wedge \mathrm{T} \neq \tilde{\textsc{pown}}(\tilde{\mathbb{l}}'' \mathit{lck})$, then let $\tilde{t}_{\mathrm{T}}^{a\prime\prime} = \tilde{t}_{\mathrm{T}}^{a} \mathbin{\tilde{+}_t} \textsc{absTime}(\tilde{c},\mathrm{T})$, which is obviously a safe approximation of the first point in time at which T can acquire *lck*. Also let $\tilde{c}'$ be any configuration derived before (i.e., $\tilde{c}' = \tilde{c}$) or inside the **repeat**-loop. Note that $\tilde{t}_{\mathrm{T}}^{r} = \tilde{\top}_t$ is used to exit the loop in case $\tilde{\textsc{dl}}(\tilde{\mathbb{l}}'' \mathit{lck}) \not\tilde{<}_t (\tilde{t}_{\mathrm{T}}^{a\prime} \mathbin{\tilde{+}_t} \textsc{absTime}(\tilde{c}',\mathrm{T}))$ or $0 \in \gamma_t(\textsc{absTime}(\tilde{c}',\mathrm{T}))$, where the latter case means that a $\tilde{t}_{\mathrm{T}}^{a\prime}$ such that $\tilde{\textsc{rel}}(\tilde{\mathbb{l}}'' \mathit{lck}) \not\tilde{<}_t \tilde{t}_{\mathrm{T}}^{a\prime}$ cannot be derived.

i. If $\tilde{\textsc{dl}}(\tilde{\mathbb{l}}'' \mathit{lck}) \not\tilde{<}_t \tilde{t}_{\mathrm{T}}^{a\prime} \mathbin{\tilde{+}_t} \textsc{absTime}(\tilde{c}',\mathrm{T})$, then it must be that at $\tilde{t}_{\mathrm{T}}^{a\prime} \mathbin{\tilde{+}_t} \textsc{absTime}(\tilde{c}',\mathrm{T})$, some other thread will have acquired *lck* (hence, $\tilde{t}_{\mathrm{T}}^{a\prime}$ is the last point in time when T can acquire *lck*). Thus, it must be that $t_{\mathrm{T}}^{a\prime} \in \gamma_t((\tilde{t}_{\mathrm{T}}^{a\prime\prime} \mathbin{\tilde{\sqcup}_t} \tilde{t}_{\mathrm{T}}^{a\prime}) \mathbin{\tilde{\sqcap}_t} \tilde{\textsc{dl}}(\tilde{\mathbb{l}}'' \mathit{lck}) \mathbin{\tilde{\sqcap}_t} (\tilde{\textsc{rel}}(\tilde{\mathbb{l}}'' \mathit{lck}) \mathbin{\tilde{\sqcup}_t} \alpha_t(\{\infty\})))$ since $\textsc{dl}(\mathbb{l}'' \mathit{lck}) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}'' \mathit{lck}))$ and $\textsc{rel}(\mathbb{l}'' \mathit{lck}) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}'' \mathit{lck}))$.

ii. If $0 \in \gamma_t(\textsc{absTime}(\tilde{c}',\mathrm{T}))$ and also $\tilde{\textsc{dl}}(\tilde{\mathbb{l}}'' \mathit{lck}) \not\tilde{<}_t \tilde{t}_{\mathrm{T}}^{a\prime} \mathbin{\tilde{+}_t} \textsc{absTime}(\tilde{c}',\mathrm{T})$, then it must be that $\tilde{t} \mathbin{\tilde{+}_t} \textsc{absTime}(\tilde{c}'',\mathrm{T})$, where $\tilde{t} = (\tilde{t}_{\mathrm{T}}^{a\prime} \mathbin{\tilde{\sqcup}_t} \alpha_t(\{\infty\})) \mathbin{\tilde{\sqcap}_t} \tilde{\textsc{rel}}(\tilde{\mathbb{l}}\, \mathit{lck})$ and $\tilde{c}'' = \langle [\mathrm{T}', pc_{\mathrm{T}'}, \tilde{\mathbb{r}}_{\mathrm{T}'}, (\mathrm{T} = \mathrm{T}' \;?\; \tilde{t} : \tilde{t}_{\mathrm{T}'}^{a})]_{\mathrm{T}' \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}'' \rangle$, is a safe approximation of the last point in time when T can (or rather, will) acquire *lck* (c.f., Assumption 5.50) since $\textsc{rel}(\mathbb{l}'' \mathit{lck}) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}'' \mathit{lck}))$. Thus, it must be that $t_{\mathrm{T}}^{a\prime} \in \gamma_t((\tilde{t}_{\mathrm{T}}^{a\prime\prime} \mathbin{\tilde{\sqcup}_t} \tilde{t}_{\mathrm{T}}^{a\prime}) \mathbin{\tilde{\sqcap}_t} \tilde{\textsc{dl}}(\tilde{\mathbb{l}}'' \mathit{lck}) \mathbin{\tilde{\sqcap}_t} (\tilde{\textsc{rel}}(\tilde{\mathbb{l}}'' \mathit{lck}) \mathbin{\tilde{\sqcup}_t} \alpha_t(\{\infty\})))$ since $\textsc{rel}(\mathbb{l}'' \mathit{lck}) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}'' \mathit{lck}))$ and $\textsc{dl}(\mathbb{l}'' \mathit{lck}) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}'' \mathit{lck}))$.

iii. If $0 \notin \gamma_t(\textsc{absTime}(\tilde{c}',\mathrm{T}))$ and also $\tilde{\textsc{dl}}(\tilde{\mathbb{l}}'' \mathit{lck}) \not\tilde{<}_t \tilde{t}_{\mathrm{T}}^{a\prime} \mathbin{\tilde{+}_t} \textsc{absTime}(\tilde{c}',\mathrm{T})$, then it must be that, at some point, $\tilde{\textsc{rel}}(\tilde{\mathbb{l}}'' \mathit{lck}) \not\tilde{<}_t \tilde{t}_{\mathrm{T}}^{a\prime}$. Since $\textsc{rel}(\mathbb{l}'' \mathit{lck}) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}'' \mathit{lck}))$ and $\textsc{dl}(\mathbb{l}'' \mathit{lck}) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}'' \mathit{lck}))$, it is thus easy to see that $t_{\mathrm{T}}^{a\prime} \in \gamma_t((\tilde{t}_{\mathrm{T}}^{a\prime\prime} \mathbin{\tilde{\sqcup}_t} \tilde{t}_{\mathrm{T}}^{a\prime}) \mathbin{\tilde{\sqcap}_t} \tilde{\textsc{dl}}(\tilde{\mathbb{l}}'' \mathit{lck}) \mathbin{\tilde{\sqcap}_t} (\tilde{\textsc{rel}}(\tilde{\mathbb{l}}'' \mathit{lck}) \mathbin{\tilde{\sqcup}_t} \alpha_t(\{\infty\})))$.

This concludes the proof. ∎

It is important to notice that all the possible orders in which threads can acquire a lock in the concrete case are covered by the abstract transition rela-

tions, even though $\widetilde{\mathbf{Time}} = \mathbf{Intv}$. Since $\widetilde{\mathbf{Time}} = \mathbf{Intv}$, $\mathbf{Thrd}_{exe}$ might differ for concrete and abstract cases as discussed above. This means that even if some thread is the first in a set of threads to issue a `lock`-statement acting on some lock, $lck \in \mathbf{Lck}$, some other thread could issue its corresponding `lock` lck-statement first in the abstract case. Lemma 5.55 states that even if this happens, the first thread will be assigned, and eventually acquire, lck anyway for some transition sequence(s).

Lemma 5.55 (Properties of owner assignment for `lock`-transitions):
Given the valid concrete configurations (c.f., Definition 4.4), abstract configurations, lock and threads

$$
\begin{aligned}
&c^0 @ \langle [\mathrm{T}, pc^0_{\mathrm{T}}, \mathbb{r}^0_{\mathrm{T}}, t^{a^0}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^0, \mathbb{l}^0 \rangle \in \mathbf{Conf},\\
&c^i @ \langle [\mathrm{T}, pc^i_{\mathrm{T}}, \mathbb{r}^i_{\mathrm{T}}, t^{a^i}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^i, \mathbb{l}^i \rangle \in \mathbf{Conf},\\
&c^n @ \langle [\mathrm{T}, pc^n_{\mathrm{T}}, \mathbb{r}^n_{\mathrm{T}}, t^{a^n}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^n, \mathbb{l}^n \rangle \in \mathbf{Conf},\\
&\tilde{c}^0 @ \langle [\mathrm{T}, pc^{\tilde{c}^0}_{\mathrm{T}}, \tilde{\mathbb{r}}^0_{\mathrm{T}}, \tilde{t}^{a^0}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^0}}, \tilde{\mathbb{x}}^0, \tilde{\mathbb{l}}^0 \rangle \in \tilde{\mathbf{Conf}},\\
&\tilde{c}^j @ \langle [\mathrm{T}, pc^{\tilde{c}^j}_{\mathrm{T}}, \tilde{\mathbb{r}}^j_{\mathrm{T}}, \tilde{t}^{a^j}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^j}}, \tilde{\mathbb{x}}^j, \tilde{\mathbb{l}}^j \rangle \in \tilde{\mathbf{Conf}},\\
&\tilde{c}^k @ \langle [\mathrm{T}, pc^{\tilde{c}^k}_{\mathrm{T}}, \tilde{\mathbb{r}}^k_{\mathrm{T}}, \tilde{t}^{a^k}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^k}}, \tilde{\mathbb{x}}^k, \tilde{\mathbb{l}}^k \rangle \in \tilde{\mathbf{Conf}},\\
&lck' \in \mathbf{Lck},\\
&\mathrm{T}' \in \mathbf{Thrd}_{\tilde{c}^k} \textit{ and}\\
&\mathrm{T}'' \in \mathbf{Thrd}_{\tilde{c}^k},
\end{aligned}
$$

such that

$$
\begin{gathered}
0 \leq i < n, \\
c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^i \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^n, \\
0 \leq j < k, \\
\tilde{c}^0 \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}^j \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}^k, \\
\mathbf{Thrd}_{\tilde{c}^k} \subseteq \mathbf{Thrd}_{\tilde{c}^j} \subseteq \mathbf{Thrd}_{\tilde{c}^0} \subseteq \mathbf{Thrd}, \\
\textsc{stm}(\mathrm{T}'', pc^n_{\mathrm{T}''}) = [\texttt{lock}\ lck']^{pc^n_{\mathrm{T}''}}, \\
\mathrm{T}'' \in \mathbf{Thrd}^{c^n}_{exe}, \\
\textsc{stm}(\mathrm{T}', pc^{\tilde{c}^k}_{\mathrm{T}'}) = [\texttt{lock}\ lck']^{pc^{\tilde{c}^k}_{\mathrm{T}'}}, \\
\mathrm{T}' \in \mathbf{Thrd}^{\tilde{c}^k}_{exe}, \\
\forall h \in \{0, \ldots, k-1\} : (\mathrm{T}' \in \mathbf{Thrd}^{\tilde{c}^h}_{exe} \Rightarrow \textsc{stm}(\mathrm{T}', pc^{\tilde{c}^h}_{\mathrm{T}'}) \neq [\texttt{lock}\ lck']^{pc^{\tilde{c}^h}_{\mathrm{T}'}}), \\
\textsc{rel}(\mathbb{l}^i\ lck') \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^j\ lck')), \\
(pc^i_{\mathrm{T}'} = pc^{\tilde{c}^k}_{\mathrm{T}'} \wedge \\
t^{a^i}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a^k}_{\mathrm{T}'}) \wedge \\
\mathrm{T}' \in \mathbf{Thrd}^{c^i}_{exe} \wedge \\
\textsc{own}(\mathbb{l}^i\ lck') = \bot_{thrd} \wedge \\
\textsc{own}(\mathbb{l}^{i+1}\ lck') = \mathrm{T}') \ \textit{and} \\
(pc^n_{\mathrm{T}''} = pc^{\tilde{c}^j}_{\mathrm{T}''} \wedge \\
t^{a^n}_{\mathrm{T}''} \in \gamma_t(\tilde{t}^{a^j}_{\mathrm{T}''}) \wedge \\
\mathrm{T}'' \in \mathbf{Thrd}^{all^{\tilde{c}^j}}_{exe} \ \wedge \\
\tilde{\textsc{own}}(\tilde{\mathbb{l}}^j\ lck') = \bot_{thrd} \wedge \\
\tilde{\textsc{own}}(\tilde{\mathbb{l}}^{j+1}\ lck') = \mathrm{T}'),
\end{gathered}
$$

where the trace for T' *in* $\tilde{c}^0 \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}^k$ *is the same as in* $c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^i$, *the trace for* T'' *in* $\tilde{c}^0 \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}^j$ *is the same as in* $c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^n$, $\mathbf{Thrd}^{c^n}_{exe}$ *is as defined in Table 4.3, and* $\mathbf{Thrd}^{all^{\tilde{c}^j}}_{exe}$ *and* $\mathbf{Thrd}^{\tilde{c}^k}_{exe}$ *are as defined in Table 5.6,* $\xrightarrow[prg]{\sim}$ *satisfies:*

$$
\begin{gathered}
\tilde{\textsc{own}}(\tilde{\mathbb{l}}^k\ lck') = \tilde{\textsc{own}}(\tilde{\mathbb{l}}^{j+1}\ lck') = \textsc{own}(\mathbb{l}^{i+1}\ lck') = \mathrm{T}' \wedge \\
\textsc{stt}(\mathbb{l}^i\ lck') = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^j\ lck') = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^{j+1}\ lck') = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^k\ lck') = \textit{unlocked} \wedge \\
\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^k\ lck') = \tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{j+1}\ lck') \wedge \\
\min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^k\ lck'))) = -\infty \wedge \\
t^{a^i}_{\mathrm{T}'} + \textsc{time}(c^i, \mathrm{T}') \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^k\ lck'))
\end{gathered}
$$

□

PROOF. Assume that the valid concrete configurations, abstract configurations, lock and threads

$$
\begin{array}{l}
c^0 @ \langle [\mathrm{T}, pc^0_{\mathrm{T}}, \mathbb{r}^0_{\mathrm{T}}, t^{a^0}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^0, \mathbb{l}^0 \rangle \in \mathbf{Conf},\\
c^i @ \langle [\mathrm{T}, pc^i_{\mathrm{T}}, \mathbb{r}^i_{\mathrm{T}}, t^{a^i}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^i, \mathbb{l}^i \rangle \in \mathbf{Conf},\\
c^n @ \langle [\mathrm{T}, pc^n_{\mathrm{T}}, \mathbb{r}^n_{\mathrm{T}}, t^{a^n}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^n, \mathbb{l}^n \rangle \in \mathbf{Conf},\\
\tilde{c}^0 @ \langle [\mathrm{T}, pc^{\tilde{c}^0}_{\mathrm{T}}, \tilde{\mathbb{r}}^0_{\mathrm{T}}, \tilde{t}^{a^0}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^0}}, \tilde{\mathbb{x}}^0, \tilde{\mathbb{l}}^0 \rangle \in \mathbf{C\tilde{o}nf},\\
\tilde{c}^j @ \langle [\mathrm{T}, pc^{\tilde{c}^j}_{\mathrm{T}}, \tilde{\mathbb{r}}^j_{\mathrm{T}}, \tilde{t}^{a^j}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^j}}, \tilde{\mathbb{x}}^j, \tilde{\mathbb{l}}^j \rangle \in \mathbf{C\tilde{o}nf},\\
\tilde{c}^k @ \langle [\mathrm{T}, pc^{\tilde{c}^k}_{\mathrm{T}}, \tilde{\mathbb{r}}^k_{\mathrm{T}}, \tilde{t}^{a^k}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^k}}, \tilde{\mathbb{x}}^k, \tilde{\mathbb{l}}^k \rangle \in \mathbf{C\tilde{o}nf},\\
lck' \in \mathbf{Lck},\\
\mathrm{T}' \in \mathbf{Thrd}_{\tilde{c}^k} \text{ and}\\
\mathrm{T}'' \in \mathbf{Thrd}_{\tilde{c}^k},
\end{array}
$$

are such that

$$
\begin{array}{c}
0 \le i < n,\\
c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^i \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^n,\\
0 \le j < k,\\
\tilde{c}^0 \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}^j \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}^k,\\
\mathbf{Thrd}_{\tilde{c}^k} \subseteq \mathbf{Thrd}_{\tilde{c}^j} \subseteq \mathbf{Thrd}_{\tilde{c}^0} \subseteq \mathbf{Thrd},\\
\mathrm{STM}(\mathrm{T}'', pc^n_{\mathrm{T}''}) = [\texttt{lock}\ lck']^{pc^n_{\mathrm{T}''}},\\
\mathrm{T}'' \in \mathbf{Thrd}^{c^n}_{exe},\\
\mathrm{STM}(\mathrm{T}', pc^{\tilde{c}^k}_{\mathrm{T}'}) = [\texttt{lock}\ lck']^{pc^{\tilde{c}^k}_{\mathrm{T}'}},\\
\mathrm{T}' \in \mathbf{Thrd}^{\tilde{c}^k}_{exe},\\
\forall h \in \{0,\ldots,k-1\} : (\mathrm{T}' \in \mathbf{Thrd}^{\tilde{c}^h}_{exe} \Rightarrow \mathrm{STM}(\mathrm{T}', pc^{\tilde{c}^h}_{\mathrm{T}'}) \neq [\texttt{lock}\ lck']^{pc^{\tilde{c}^h}_{\mathrm{T}'}}),\\
\mathrm{REL}(\mathbb{l}^i\ lck') \in \gamma_t(\tilde{\mathrm{REL}}(\tilde{\mathbb{l}}^j\ lck')),\\
(pc^i_{\mathrm{T}'} = pc^{\tilde{c}^k}_{\mathrm{T}'} \wedge\\
t^{a^i}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a^k}_{\mathrm{T}'}) \wedge\\
\mathrm{T}' \in \mathbf{Thrd}^{c^i}_{exe} \wedge\\
\mathrm{OWN}(\mathbb{l}^i\ lck') = \bot_{thrd} \wedge\\
\mathrm{OWN}(\mathbb{l}^{i+1}\ lck') = \mathrm{T}') \text{ and}\\
(pc^n_{\mathrm{T}''} = pc^{\tilde{c}^j}_{\mathrm{T}''} \wedge\\
t^{a^n}_{\mathrm{T}''} \in \gamma_t(\tilde{t}^{a^j}_{\mathrm{T}''}) \wedge\\
\mathrm{T}'' \in \mathbf{Thrd}^{all^{\tilde{c}^j}}_{exe} \wedge\\
\mathrm{O\tilde{W}N}(\tilde{\mathbb{l}}^j\ lck') = \bot_{thrd} \wedge\\
\mathrm{O\tilde{W}N}(\tilde{\mathbb{l}}^{j+1}\ lck') = \mathrm{T}'),
\end{array}
$$

where the trace for T' in $\tilde{c}^0 \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}^k$ is the same as in $c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^i$, the trace for T'' in $\tilde{c}^0 \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}^j$ is the same as in $c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^n$, $\mathbf{Thrd}_{exe}^{c^n}$ is as defined in Table 4.3, and $\mathbf{Thrd}_{exe}^{all^{\tilde{c}^j}}$ and $\mathbf{Thrd}_{exe}^{\tilde{c}^k}$ are as defined in Table 5.6.

First note that since the trace for T' in $\tilde{c}^0 \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}^k$ is the same as in $c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^i$, the trace for T'' in $\tilde{c}^0 \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}^j$ is the same as in $c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^n$, $\textsc{stm}(\mathrm{T}'', pc^n_{\mathrm{T}''}) = [\texttt{lock}\ lck']^{pc^n_{\mathrm{T}''}}$, $\mathrm{T}'' \in \mathbf{Thrd}_{exe}^{c^n}$, $\textsc{stm}(\mathrm{T}', pc^{\tilde{c}^k}_{\mathrm{T}'}) = [\texttt{lock}\ lck']^{pc^{\tilde{c}^k}_{\mathrm{T}'}}$, $\mathrm{T}' \in \mathbf{Thrd}_{exe}^{\tilde{c}^k}$, $pc^i_{\mathrm{T}'} = pc^{\tilde{c}^k}_{\mathrm{T}'}$, $t^{a^i}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a^k}_{\mathrm{T}'})$, $\mathrm{T}' \in \mathbf{Thrd}_{exe}^{c^i}$, $\textsc{own}(\mathbb{l}^i\ lck') = \bot_{thrd}$, $\textsc{own}(\mathbb{l}^{i+1}\ lck') = \mathrm{T}'$, $pc^n_{\mathrm{T}''} = pc^{\tilde{c}^j}_{\mathrm{T}''}$, $t^{a^n}_{\mathrm{T}''} \in \gamma_t(\tilde{t}^{a^j}_{\mathrm{T}''})$, $\tilde{\textsc{own}}(\tilde{\mathbb{l}}^j\ lck') = \bot_{thrd}$ and $\tilde{\textsc{own}}(\tilde{\mathbb{l}}^{j+1}\ lck') = \mathrm{T}'$, it must be that T' acquires lck' in the transition between c^i and c^{i+1} and T'' wants to acquire lck' in a transition from c^n, while the abstract trace represents a situation (that can occur due to that $\tilde{\mathbf{Time}} = \mathbf{Intv}$) where T'' reaches the $\texttt{lock}\ lck'$-statement (i.e., it reaches $pc^n_{\mathrm{T}''}$) before T' (i.e., before T' reaches $pc^i_{\mathrm{T}'}$), but lck' is assigned to T' as shown below.

Since $\forall h \in \{0,\ldots,k-1\} : (\mathrm{T}' \in \mathbf{Thrd}_{exe}^{\tilde{c}^h} \Rightarrow \textsc{stm}(\mathrm{T}', pc^{\tilde{c}^h}_{\mathrm{T}'}) \neq [\texttt{lock}\ lck']^{pc^{\tilde{c}^h}_{\mathrm{T}'}})$, $\textsc{own}(\mathbb{l}^i\ lck') = \bot_{thrd}$, $\tilde{\textsc{own}}(\tilde{\mathbb{l}}^j\ lck') = \bot_{thrd}$ and $\tilde{\textsc{own}}(\tilde{\mathbb{l}}^{j+1}\ lck') = \mathrm{T}'$, it is easy to see that $\tilde{\textsc{own}}(\tilde{\mathbb{l}}^k\ lck') = \tilde{\textsc{own}}(\tilde{\mathbb{l}}^{j+1}\ lck') = \textsc{own}(\mathbb{l}^{i+1}\ lck') = \mathrm{T}'$, $\textsc{stt}(\mathbb{l}^i\ lck') = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^j\ lck') = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^{j+1}\ lck') = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^k\ lck') = \mathit{unlocked}$ and $\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^k\ lck') = \tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{j+1}\ lck')$ (c.f., Table 5.6).

Since $\textsc{dlock}$ is used to determine $\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{j+1}\ lck')$ and $\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^k\ lck') = \tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{j+1}\ lck')$, it is easy to see that $\min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^k\ lck'))) = -\infty$ since $\textsc{dlock}$ is used only if $\exists \mathrm{T} \in \mathbf{Thrd}_{exe}^{all^{\tilde{c}^j}} : \textsc{stm}(\mathrm{T}, pc^{\tilde{c}^j}_{\mathrm{T}}) = [\texttt{lock}\ lck']^{pc^{\tilde{c}^j}_{\mathrm{T}}}$ (c.f., Table 5.6) which is the case since $pc^n_{\mathrm{T}''} = pc^{\tilde{c}^j}_{\mathrm{T}''}$, $\tilde{\textsc{own}}(\tilde{\mathbb{l}}^j\ lck') = \bot_{thrd}$ and $\tilde{\textsc{own}}(\tilde{\mathbb{l}}^{j+1}\ lck') = \mathrm{T}'$ (c.f., Algorithm 5.11).

Since $\mathrm{T}' \in \mathbf{Thrd}_{exe}^{c^i}$, it must be that $t^{a^i}_{\mathrm{T}'} + \textsc{time}(c^i, \mathrm{T}') = \min(\{t^{a^i}_{\mathrm{T}} + \textsc{time}(c^i, \mathrm{T}) \mid \mathrm{T} \in \mathbf{Thrd}\})$, and since $\mathrm{T}'' \in \mathbf{Thrd}_{exe}^{c^n}$, it must be that $t^{a^n}_{\mathrm{T}''} + \textsc{time}(c^n, \mathrm{T}'') = \min(\{t^{a^n}_{\mathrm{T}} + \textsc{time}(c^n, \mathrm{T}) \mid \mathrm{T} \in \mathbf{Thrd}\})$. But since $c^i \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^n$, it must be that $t^{a^i}_{\mathrm{T}'} + \textsc{time}(c^i, \mathrm{T}') \leq t^{a^n}_{\mathrm{T}''} + \textsc{time}(c^n, \mathrm{T}'')$ (Lemma 4.2). Note that by choosing c^0, c^m, c^n, $\tilde{c}^0$ and $\tilde{c}^j$ (defined by Lemma 5.53) to be c^n, c^n, c^n, $\tilde{c}^j$ and $\tilde{c}^j$ (defined by this proof), respectively, and assuming that $\textsc{own}(\mathbb{l}^n\ lck') = \bot_{thrd}$ and $\textsc{rel}(\mathbb{l}^n\ lck') \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^j\ lck'))$ (which is actually not necessarily the case since T' acquires lck' in the transition between c^i and c^{i+1}; however, note that this assumption is okay since if T' would not

acquire lck', then $\text{OWN}(\mathbb{l}^n\ lck') = \bot_{thrd}$ and $\text{REL}(\mathbb{l}^n\ lck') \in \gamma_t(\text{R}\tilde{\text{E}}\text{L}(\tilde{\mathbb{l}}^j\ lck'))$ would hold since $\text{OWN}(\mathbb{l}^i\ lck') = \bot_{thrd}$ and $\text{REL}(\mathbb{l}^i\ lck') \in \gamma_t(\text{R}\tilde{\text{E}}\text{L}(\tilde{\mathbb{l}}^j\ lck'))$), it is easy to see that $t^{a^n}_{\text{T}''} + \text{TIME}(c^n, \text{T}'') \in \gamma_t(\text{DLLOCK}(\tilde{c}^j, lck'))$ since $\mathbf{Thrd}_{\tilde{c}^k} \subseteq \mathbf{Thrd}_{\tilde{c}^j} \subseteq \mathbf{Thrd}_{\tilde{c}^0} \subseteq \mathbf{Thrd}$, $\text{STM}(\text{T}'', pc^n_{\text{T}''}) = [\texttt{lock}\ lck']^{pc^n_{\text{T}''}}$, $\text{T}'' \in \mathbf{Thrd}^{c^n}_{exe}$ and $t^{a^n}_{\text{T}''} \in \gamma_t(\tilde{t}^{a^j}_{\text{T}''})$ (Lemma 5.53). But then, since $\min(\gamma_t(\tilde{\text{DL}}(\tilde{\mathbb{l}}^k\ lck'))) = -\infty$, $\tilde{\text{DL}}(\tilde{\mathbb{l}}^k\ lck') = \text{DLLOCK}(\tilde{c}^j, lck')$ and $t^{a^i}_{\text{T}'} + \text{TIME}(c^i, \text{T}') \leq t^{a^n}_{\text{T}''} + \text{TIME}(c^n, \text{T}'')$, it must be that $t^{a^i}_{\text{T}'} + \text{TIME}(c^i, \text{T}') \in \gamma_t(\tilde{\text{DL}}(\tilde{\mathbb{l}}^k\ lck'))$ which concludes the proof. ■

Three lemmas will be presented in order to prove that the abstract transitions described by $\xrightarrow[prg]{\sim}$ safely approximate the concrete transitions described by $\xrightarrow[prg]{}$. The lemmas hold given that the concrete transition sequences are finite in length (i.e., given that they terminate) and that either no thread issues a `load`-statement on a global variable or that the thread issuing the `load`-statement is the sole thread in $\mathbf{Thrd}_{exe}$ in any step of the transition sequence. The first lemma shows that the `halt`-, `skip`-, `:=`-, `if`-, `load`-, `store`- and `unlock`-statements, and also the `lock`-statement if the issuing thread immediately is assigned the lock, are safely approximated (Lemma 5.56). Note that a variable is considered global if it could transfer data between two or more threads (c.f., Algorithm 6.9).

Lemma 5.56 (Soundness of $\xrightarrow[prg]{\sim}$, no frozen thread):

Given the valid concrete configurations (c.f., Definition 4.4), abstract configurations and thread

$$
\begin{aligned}
&c^0 @ \langle [\text{T}, pc^0_{\text{T}}, \mathbb{r}^0_{\text{T}}, t^{a^0}_{\text{T}}]_{\text{T}\in\mathbf{Thrd}}, \mathbb{x}^0, \mathbb{l}^0 \rangle \in \mathbf{Conf},\\
&c^n @ \langle [\text{T}, pc^n_{\text{T}}, \mathbb{r}^n_{\text{T}}, t^{a^n}_{\text{T}}]_{\text{T}\in\mathbf{Thrd}}, \mathbb{x}^n, \mathbb{l}^n \rangle \in \mathbf{Conf},\\
&\tilde{c}^0 @ \langle [\text{T}, pc^{\tilde{c}^0}_{\text{T}}, \tilde{\mathbb{r}}^0_{\text{T}}, \tilde{t}^{a^0}_{\text{T}}]_{\text{T}\in\mathbf{Thrd}_{\tilde{c}^0}}, \tilde{\mathbb{x}}^0, \tilde{\mathbb{l}}^0 \rangle \in \tilde{\mathbf{Conf}},\\
&\tilde{c}^k @ \langle [\text{T}, pc^{\tilde{c}^k}_{\text{T}}, \tilde{\mathbb{r}}^k_{\text{T}}, \tilde{t}^{a^k}_{\text{T}}]_{\text{T}\in\mathbf{Thrd}_{\tilde{c}^k}}, \tilde{\mathbb{x}}^k, \tilde{\mathbb{l}}^k \rangle \in \tilde{\mathbf{Conf}},\ \textit{and}\\
&\text{T}' \in \mathbf{Thrd}_{\tilde{c}^k},
\end{aligned}
$$

such that

$$
\begin{gathered}
0 \leq n, \\
c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^n, \\
0 \leq k, \\
\tilde{c}^0 \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}^k, \\
\mathbf{Thrd}_{\tilde{c}^k} \subseteq \mathbf{Thrd}_{\tilde{c}^0} \subseteq \mathbf{Thrd}, \\
pc^0_{\mathrm{T}'} = pc^{\tilde{c}^0}_{\mathrm{T}'}, \\
\mathbb{I}^0_{\mathrm{T}'} \in \gamma_{reg}(\tilde{\mathbb{I}}^0_{\mathrm{T}'}), \\
t^{a^0}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a^0}_{\mathrm{T}'}), \\
\exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}^0) : \forall x \in \mathbf{Var} : \forall \mathrm{T} \in \mathbf{Thrd} : ((\mathbb{x}^0\ x)\ \mathrm{T}) \subseteq ((\mathbb{x}'\ x)\ \mathrm{T}), \\
\forall lck \in \mathbf{Lck} : ((\textsc{own}(\mathbb{l}^0\ lck) \neq \bot_{thrd} \Rightarrow (\textsc{stt}(\mathbb{l}^0\ lck) = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^0\ lck) \wedge \\
\textsc{own}(\mathbb{l}^0\ lck) = \tilde{\textsc{own}}(\tilde{\mathbb{l}}^0\ lck) \wedge \\
\textsc{dl}(\mathbb{l}^0\ lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck)) \wedge \\
\textsc{powned}(\mathbb{l}^0\ lck) = \tilde{\textsc{powned}}(\tilde{\mathbb{l}}^0\ lck) \wedge \\
\textsc{rel}(\mathbb{l}^0\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^0\ lck)) \wedge \\
\min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck))) = -\infty)) \wedge \\
(\textsc{own}(\mathbb{l}^0\ lck) = \bot_{thrd} \Rightarrow ((\textsc{own}(\mathbb{l}^0\ lck) = \tilde{\textsc{own}}(\tilde{\mathbb{l}}^0\ lck) \vee \\
(\tilde{\textsc{own}}(\tilde{\mathbb{l}}^0\ lck) = \mathrm{T}' \wedge \\
\tilde{\textsc{stt}}(\tilde{\mathbb{l}}^0\ lck) = \mathit{unlocked} \wedge \\
t^{a^n}_{\mathrm{T}'} + \textsc{time}(c^n, \mathrm{T}') \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck)) \wedge \\
\min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck))) = -\infty)) \wedge \\
\textsc{powned}(\mathbb{l}^0\ lck) = \tilde{\textsc{powned}}(\tilde{\mathbb{l}}^0\ lck) \wedge \\
\textsc{rel}(\mathbb{l}^0\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^0\ lck)) \wedge \\
\textsc{stm}(\mathrm{T}', pc^0_{\mathrm{T}'}) = [\texttt{lock}\ lck]^{pc^0_{\mathrm{T}'}} \Rightarrow \\
(\mathbb{l}^n\ lck = \mathbb{l}^0\ lck \wedge \\
\tilde{\mathbb{l}}^k\ lck = \tilde{\mathbb{l}}^0\ lck)))), \\
\forall i \in \{0, \ldots, n-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{c^i}_{exe}, \\
\textsc{stm}(\mathrm{T}', pc^n_{\mathrm{T}'}) \neq [\texttt{halt}]^{pc^n_{\mathrm{T}'}} \Rightarrow \mathrm{T}' \in \mathbf{Thrd}^{c^n}_{exe}, \\
\forall i \in \{0, \ldots, k-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{\tilde{c}^i}_{exe}, \\
\textsc{stm}(\mathrm{T}', pc^{\tilde{c}^k}_{\mathrm{T}'}) \neq [\texttt{halt}]^{pc^{\tilde{c}^k}_{\mathrm{T}'}} \Rightarrow \mathrm{T}' \in \mathbf{Thrd}^{\tilde{c}^k}_{exe}, \text{ and} \\
\forall i \in \{0, \ldots, k\} : (|\mathbf{Thrd}^{\tilde{c}^i}_{exe}| \not> 1 \vee \\
\{\mathrm{T} \in \mathbf{Thrd}^{\tilde{c}^i}_{exe} \mid \exists r \in \mathbf{Reg}_{\mathrm{T}} : \exists x \in \mathbf{Var_g} : \\
\textsc{stm}(\mathrm{T}, pc^{\tilde{c}^i}_{\mathrm{T}}) = [\texttt{load}\ r\ \texttt{from}\ x]^{pc^{\tilde{c}^i}_{\mathrm{T}}}\} = \emptyset),
\end{gathered}
$$

where for all $i \in \{0, \ldots, n\}$, $\mathbf{Thrd}^{c^i}_{exe}$ *is as defined in Table 4.3, for all* $i \in \{0, \ldots, k\}$, $\mathbf{Thrd}^{\tilde{c}^i}_{exe}$ *is as defined in Table 5.6, and* $\mathbf{Var_g}$ *contains all* $x \in \mathbf{Var}$

such that x can be written to by one thread and read from by another thread (i.e., there might be a data dependency between the threads; note that $\mathbf{Var_g}$ *can be derived using Algorithm 6.9),* $\xrightarrow[prg]{\sim}$ *satisfies:*

$$
\begin{array}{l}
\forall c@\langle[\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle \in \mathbf{Conf}: \\
\quad (c^n \xrightarrow[prg]{} c \Rightarrow \exists \tilde{c}@\langle[\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^k}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \mathbf{C\tilde{o}nf}: \\
\qquad (\tilde{c}^k \xrightarrow[prg]{\sim} \tilde{c} \wedge \\
\qquad pc_{\mathrm{T}'} = pc^{\tilde{c}}_{\mathrm{T}'} \wedge \\
\qquad \mathbb{r}_{\mathrm{T}'} \in \gamma_{reg}(\tilde{\mathbb{r}}_{\mathrm{T}'}) \wedge \\
\qquad t^a_{\mathrm{T}'} \in \gamma_t(\tilde{t}^a_{\mathrm{T}'}) \wedge \\
\qquad \exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}) : (\forall x \in \mathbf{Var} : ((\mathbb{x}\ x)\ \mathrm{T}') \subseteq ((\mathbb{x}'\ x)\ \mathrm{T}')) \wedge \\
\qquad \forall lck \in \mathbf{Lck} : ((\textsc{own}(\mathbb{l}^0\ lck) = \mathrm{T}' \vee \textsc{own}(\mathbb{l}\ lck) = \mathrm{T}') \Rightarrow \\
\qquad\qquad (\textsc{stt}(\mathbb{l}\ lck) = \textsc{s}\tilde{\textsc{t}}\textsc{t}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\qquad \textsc{own}(\mathbb{l}\ lck) = \textsc{o}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\qquad \textsc{dl}(\mathbb{l}\ lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}\ lck)) \wedge \\
\qquad\qquad \textsc{pown}(\mathbb{l}\ lck) = \textsc{po}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\qquad \textsc{rel}(\mathbb{l}\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}\ lck)) \wedge \\
\qquad\qquad \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}\ lck))) = -\infty))))
\end{array}
$$

□

PROOF. Assume that the valid concrete configurations (c.f., Definition 4.4), abstract configurations and thread

$$
\begin{array}{l}
c^0@\langle[\mathrm{T}, pc^0_{\mathrm{T}}, \mathbb{r}^0_{\mathrm{T}}, t^{a^0}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^0, \mathbb{l}^0\rangle \in \mathbf{Conf}, \\
c^n@\langle[\mathrm{T}, pc^n_{\mathrm{T}}, \mathbb{r}^n_{\mathrm{T}}, t^{a^n}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^n, \mathbb{l}^n\rangle \in \mathbf{Conf}, \\
\tilde{c}^0@\langle[\mathrm{T}, pc^{\tilde{c}^0}_{\mathrm{T}}, \tilde{\mathbb{r}}^0_{\mathrm{T}}, \tilde{t}^{a^0}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^0}}, \tilde{\mathbb{x}}^0, \tilde{\mathbb{l}}^0\rangle \in \mathbf{C\tilde{o}nf}, \\
\tilde{c}^k@\langle[\mathrm{T}, pc^{\tilde{c}^k}_{\mathrm{T}}, \tilde{\mathbb{r}}^k_{\mathrm{T}}, \tilde{t}^{a^k}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^k}}, \tilde{\mathbb{x}}^k, \tilde{\mathbb{l}}^k\rangle \in \mathbf{C\tilde{o}nf}, \text{ and} \\
\mathrm{T}' \in \mathbf{Thrd}_{\tilde{c}^k},
\end{array}
$$

are such that

$$
\begin{gathered}
0 \leq n, \\
c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^n, \\
0 \leq k, \\
\tilde{c}^0 \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}^k, \\
\mathbf{Thrd}_{\tilde{c}^k} \subseteq \mathbf{Thrd}_{\tilde{c}^0} \subseteq \mathbf{Thrd}, \\
pc^0_{\mathrm{T}'} = pc^{\tilde{c}^0}_{\mathrm{T}'}, \\
\mathbb{I}^0_{\mathrm{T}'} \in \gamma_{reg}(\tilde{\mathbb{I}}^0_{\mathrm{T}'}), \\
t^{a^0}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a^0}_{\mathrm{T}'}), \\
\exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}^0) : \forall x \in \mathbf{Var} : \forall \mathrm{T} \in \mathbf{Thrd} : ((\mathbb{x}^0\ x)\ \mathrm{T}) \subseteq ((\mathbb{x}'\ x)\ \mathrm{T}), \\
\forall lck \in \mathbf{Lck} : ((\textsc{own}(\mathbb{l}^0\ lck) \neq \bot_{thrd} \Rightarrow (\textsc{stt}(\mathbb{l}^0\ lck) = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^0\ lck) \wedge \\
\textsc{own}(\mathbb{l}^0\ lck) = \tilde{\textsc{own}}(\tilde{\mathbb{l}}^0\ lck) \wedge \\
\textsc{dl}(\mathbb{l}^0\ lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck)) \wedge \\
\textsc{powN}(\mathbb{l}^0\ lck) = \tilde{\textsc{powN}}(\tilde{\mathbb{l}}^0\ lck) \wedge \\
\textsc{rel}(\mathbb{l}^0\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^0\ lck)) \wedge \\
\min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck))) = -\infty)) \wedge \\
(\textsc{own}(\mathbb{l}^0\ lck) = \bot_{thrd} \Rightarrow ((\textsc{own}(\mathbb{l}^0\ lck) = \tilde{\textsc{own}}(\tilde{\mathbb{l}}^0\ lck) \vee \\
(\tilde{\textsc{own}}(\tilde{\mathbb{l}}^0\ lck) = \mathrm{T}' \wedge \\
\tilde{\textsc{stt}}(\tilde{\mathbb{l}}^0\ lck) = \mathit{unlocked} \wedge \\
t^{a^n}_{\mathrm{T}'} + \textsc{time}(c^n, \mathrm{T}') \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck)) \wedge \\
\min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck))) = -\infty)) \wedge \\
\textsc{powN}(\mathbb{l}^0\ lck) = \tilde{\textsc{powN}}(\tilde{\mathbb{l}}^0\ lck) \wedge \\
\textsc{rel}(\mathbb{l}^0\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^0\ lck)) \wedge \\
\textsc{stm}(\mathrm{T}', pc^0_{\mathrm{T}'}) = [\texttt{lock}\ lck]^{pc^0_{\mathrm{T}'}} \Rightarrow \\
(\mathbb{l}^n\ lck = \mathbb{l}^0\ lck \wedge \\
\tilde{\mathbb{l}}^k\ lck = \tilde{\mathbb{l}}^0\ lck)))), \\
\forall i \in \{0, \ldots, n-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{c^i}_{exe}, \\
\textsc{stm}(\mathrm{T}', pc^n_{\mathrm{T}'}) \neq [\texttt{halt}]^{pc^n_{\mathrm{T}'}} \Rightarrow \mathrm{T}' \in \mathbf{Thrd}^{c^n}_{exe}, \\
\forall i \in \{0, \ldots, k-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{\tilde{c}^i}_{exe}, \\
\textsc{stm}(\mathrm{T}', pc^{\tilde{c}^k}_{\mathrm{T}'}) \neq [\texttt{halt}]^{pc^{\tilde{c}^k}_{\mathrm{T}'}} \Rightarrow \mathrm{T}' \in \mathbf{Thrd}^{\tilde{c}^k}_{exe}, \text{ and} \\
\forall i \in \{0, \ldots, k\} : (|\mathbf{Thrd}^{\tilde{c}^i}_{exe}| \not> 1 \vee \\
\{\mathrm{T} \in \mathbf{Thrd}^{\tilde{c}^i}_{exe} \mid \exists r \in \mathbf{Reg}_{\mathrm{T}} : \exists x \in \mathbf{Var_g} : \\
\textsc{stm}(\mathrm{T}, pc^{\tilde{c}^i}_{\mathrm{T}}) = [\texttt{load}\ r\ \texttt{from}\ x]^{pc^{\tilde{c}^i}_{\mathrm{T}}}\} = \emptyset),
\end{gathered}
$$

where for all $i \in \{0, \ldots, n\}$, $\mathbf{Thrd}^{c^i}_{exe}$ is as defined in Table 4.3, for all $i \in \{0, \ldots, k\}$, $\mathbf{Thrd}^{\tilde{c}^i}_{exe}$ is as defined in Table 5.6, and $\mathbf{Var_g}$ contains all $x \in \mathbf{Var}$

such that x can be written to by one thread and read from by another thread (i.e., there might be a data dependency between the threads).

First note that:

- Since $\forall i \in \{0,\ldots,n-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{c^i}_{exe}$, it must be that $pc^n_{\mathrm{T}'} = pc^0_{\mathrm{T}'}$, $\mathbb{r}^n_{\mathrm{T}'} = \mathbb{r}^0_{\mathrm{T}'}$, $t^{a^n}_{\mathrm{T}'} = t^{a^0}_{\mathrm{T}'}$ and $\forall lck \in \mathbf{Lck} : (\textsc{own}(\mathbb{l}^0\ lck) = \mathrm{T}' \Rightarrow \mathbb{l}^n\ lck = \mathbb{l}^0\ lck)$ (c.f., Table 4.3).

- Since $\forall i \in \{0,\ldots,k-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{\tilde{c}^i}_{exe}$, it must be that $pc^{\tilde{c}^k}_{\mathrm{T}'} = pc^{\tilde{c}^0}_{\mathrm{T}'}$, $\tilde{\mathbb{r}}^k_{\mathrm{T}'} = \tilde{\mathbb{r}}^0_{\mathrm{T}'}$, $\tilde{t}^{a^k}_{\mathrm{T}'} = \tilde{t}^{a^0}_{\mathrm{T}'}$ and $\forall lck \in \mathbf{Lck} : (\tilde{\textsc{own}}(\tilde{\mathbb{l}}^0\ lck) = \mathrm{T}' \Rightarrow (\tilde{\mathbb{l}}^k\ lck = \tilde{\mathbb{l}}^0\ lck \wedge \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^k\ lck))) = -\infty))$.

- Since $pc^n_{\mathrm{T}'} = pc^0_{\mathrm{T}'}$, $\mathbb{r}^n_{\mathrm{T}'} = \mathbb{r}^0_{\mathrm{T}'}$, $t^{a^n}_{\mathrm{T}'} = t^{a^0}_{\mathrm{T}'}$, $\forall lck \in \mathbf{Lck} : (\textsc{own}(\mathbb{l}^0\ lck) = \mathrm{T}' \Rightarrow \mathbb{l}^n\ lck = \mathbb{l}^0\ lck)$, $pc^{\tilde{c}^k}_{\mathrm{T}'} = pc^{\tilde{c}^0}_{\mathrm{T}'}$, $\tilde{\mathbb{r}}^k_{\mathrm{T}'} = \tilde{\mathbb{r}}^0_{\mathrm{T}'}$, $\tilde{t}^{a^k}_{\mathrm{T}'} = \tilde{t}^{a^0}_{\mathrm{T}'}$, $\forall lck \in \mathbf{Lck} : (\tilde{\textsc{own}}(\tilde{\mathbb{l}}^0\ lck) = \mathrm{T}' \Rightarrow (\tilde{\mathbb{l}}^k\ lck = \tilde{\mathbb{l}}^0\ lck \wedge \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^k\ lck))) = -\infty))$, $pc^0_{\mathrm{T}'} = pc^{\tilde{c}^0}_{\mathrm{T}'}$, $\mathbb{r}^0_{\mathrm{T}'} \in \gamma_{reg}(\tilde{\mathbb{r}}^0_{\mathrm{T}'})$, $t^{a^0}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a^0}_{\mathrm{T}'})$ and $\forall lck \in \mathbf{Lck} : (\textsc{own}(\mathbb{l}^0\ lck) = \mathrm{T}' \Rightarrow (\textsc{stt}(\mathbb{l}^0\ lck) = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^0\ lck) \wedge \textsc{own}(\mathbb{l}^0\ lck) = \tilde{\textsc{own}}(\tilde{\mathbb{l}}^0\ lck) \wedge \textsc{dl}(\mathbb{l}^0\ lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck)) \wedge \textsc{pown}(\mathbb{l}^0\ lck) = \tilde{\textsc{pown}}(\tilde{\mathbb{l}}^0\ lck) \wedge \textsc{rel}(\mathbb{l}^0\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^0\ lck)) \wedge \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck))) = -\infty))$, it must be that:

$$
\begin{array}{l}
pc^n_{\mathrm{T}'} = pc^{\tilde{c}^k}_{\mathrm{T}'} \wedge \\
\mathbb{r}^n_{\mathrm{T}'} \in \gamma_{reg}(\tilde{\mathbb{r}}^k_{\mathrm{T}'}) \wedge \\
t^{a^n}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a^k}_{\mathrm{T}'}) \wedge \\
\forall lck \in \mathbf{Lck} : (\textsc{own}(\mathbb{l}^n\ lck) = \mathrm{T}' \Rightarrow (\textsc{stt}(\mathbb{l}^n\ lck) = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^k\ lck) \wedge \\
\qquad\qquad \textsc{own}(\mathbb{l}^n\ lck) = \tilde{\textsc{own}}(\tilde{\mathbb{l}}^k\ lck) \wedge \\
\qquad\qquad \textsc{dl}(\mathbb{l}^n\ lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^k\ lck)) \wedge \\
\qquad\qquad \textsc{pown}(\mathbb{l}^n\ lck) = \tilde{\textsc{pown}}(\tilde{\mathbb{l}}^k\ lck) \wedge \\
\qquad\qquad \textsc{rel}(\mathbb{l}^n\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^k\ lck)) \wedge \\
\qquad\qquad \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^k\ lck))) = -\infty))
\end{array}
$$

- Since $c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^n$ and $\forall i \in \{0,\ldots,n-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{c^i}_{exe}$, it must be that for all $x \in \mathbf{Var}$, $((\mathbb{x}^n\ x)\ \mathrm{T}') = ((\mathbb{x}^0\ x)\ \mathrm{T}')$ if no thread writes to x in the sequence $c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^n$, or $((\mathbb{x}^n\ x)\ \mathrm{T}') = \emptyset$ if some other thread has written to x in the given sequence (c.f., Table 4.3). Thus, $\forall x \in \mathbf{Var} : ((\mathbb{x}^n\ x)\ \mathrm{T}') \subseteq ((\mathbb{x}^0\ x)\ \mathrm{T}')$.

- Since $\exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}^0) : \forall x \in \mathbf{Var} : ((\mathbb{x}^0\ x)\ \mathrm{T}') \subseteq ((\mathbb{x}'\ x)\ \mathrm{T}')$, $c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^n$, $\tilde{c}^0 \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}^k$, $\forall i \in \{0,\ldots,n-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{c^i}_{exe}$, $\forall i \in \{0,\ldots,k-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{\tilde{c}^i}_{exe}$ and TRIM is safe (Lemma 5.27), it must be that $\exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}^k) : \forall x \in \mathbf{Var} : ((\mathbb{x}^n\ x)\ \mathrm{T}') \subseteq ((\mathbb{x}'\ x)\ \mathrm{T}')$.

- Since $\forall i \in \{0,\ldots,k\} : (|\mathbf{Thrd}^{\tilde{c}^i}_{exe}| \not> 1 \vee \{\mathrm{T} \in \mathbf{Thrd}^{\tilde{c}^i}_{exe} \mid \exists r \in \mathbf{Reg}_{\mathrm{T}} : \exists x \in \mathbf{Var_g} : \textsc{stm}(\mathrm{T}, pc^{\tilde{c}^i}_{\mathrm{T}}) = [\texttt{load}\ r\ \texttt{from}\ x]^{pc^{\tilde{c}^i}_{\mathrm{T}}}\} = \emptyset)$, it must be that $\forall i \in \{0,\ldots,k\} : (\{\mathrm{T} \in \mathbf{Thrd}^{\tilde{c}^i}_{exe} \mid \exists r \in \mathbf{Reg}_{\mathrm{T}} : \exists x \in \mathbf{Var_g} : \textsc{stm}(\mathrm{T}, pc^{\tilde{c}^i}_{\mathrm{T}}) = [\texttt{load}\ r\ \texttt{from}\ x]^{pc^{\tilde{c}^i}_{\mathrm{T}}}\} \neq \emptyset \Rightarrow |\mathbf{Thrd}^{\tilde{c}^i}_{exe}| = 1)$. This means that if some thread in $\mathbf{Thrd}^{\tilde{c}^i}_{exe}$, where $i \in \{0,\ldots,k\}$, performs a `load`-statement, there is only one single thread in $\mathbf{Thrd}^{\tilde{c}^i}_{exe}$; thus that thread performs the `load`-statement. It is then easy to see, from the definition of $\mathbf{Thrd}^{\tilde{c}^i}_{exe}$, that there cannot occur any other write than those represented by $\tilde{\mathbb{x}}^i$ such that it could affect the `load`-statement of the thread in $\mathbf{Thrd}^{\tilde{c}^i}_{exe}$ (c.f., Assumption 5.50) – thus, it must be that $\tilde{\mathbb{x}}^k$ (and also all $\tilde{\mathbb{x}}^i$, where $i \in \{0,\ldots,k\}$) contains safe write history (c.f., Definition 5.18).

- Since, trivially, $\forall lck \in \mathbf{Lck} : \{\mathrm{T} \in \mathbf{Thrd}^{c^n}_{exe} \cap \mathbf{Thrd}_{\tilde{c}^k} \mid \textsc{stm}(\mathrm{T}, pc^n_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc^n_{\mathrm{T}}}\} \subseteq \{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}^k} \mid \exists l \in \mathbf{Lbl}_{\mathrm{T}} : \textsc{stm}(\mathrm{T}, l) = [\texttt{lock}\ lck]^l\}$, it must be that if T' can be assigned a lock in the concrete case, it can also be assigned the lock in the corresponding abstract case.

- If, for some $lck \in \mathbf{Lck}$, $\textsc{stm}(\mathrm{T}', pc^{\tilde{c}^k}_{\mathrm{T}'}) = [\texttt{lock}\ lck]^{pc^{\tilde{c}^k}_{\mathrm{T}'}}$, it must be that $\textsc{own}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}^{k\prime\prime}\ lck) = \mathrm{T}'$, since $\forall i \in \{0,\ldots k-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{\tilde{c}^i}_{exe}$ and $\mathrm{T}' \in \mathbf{Thrd}^{\tilde{c}^k}_{exe}$.

- Since $\mathrm{T}' \in \mathbf{Thrd}_{\tilde{c}^k}$, $\mathbf{Thrd}_{\tilde{c}^k} \subseteq \mathbf{Thrd}$, $pc^{\tilde{c}^k}_{\mathrm{T}'} = pc^n_{\mathrm{T}'}$, $t^{a^n}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a^k}_{\mathrm{T}'})$, $\forall lck \in \mathbf{Lck} : (\textsc{own}(\mathbb{l}^0\ lck) = \mathrm{T}' \Rightarrow (\textsc{stt}(\mathbb{l}^n\ lck) = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^k\ lck) \wedge \textsc{own}(\mathbb{l}^n\ lck) = \textsc{own}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}^k\ lck) \wedge \textsc{dl}(\mathbb{l}^n\ lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^k\ lck)) \wedge \textsc{pown}(\mathbb{l}^n\ lck) = \textsc{po}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}^k\ lck) \wedge \textsc{rel}(\mathbb{l}^n\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^k\ lck))))$, $\mathrm{T}' \in \mathbf{Thrd}^{\tilde{c}^k}_{exe}$, $\mathrm{T}' \in \mathbf{Thrd}^{c^n}_{exe}$, $\forall lck \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}', pc^{\tilde{c}^k}_{\mathrm{T}'}) = [\texttt{lock}\ lck]^{pc^{\tilde{c}^k}_{\mathrm{T}'}} \Rightarrow \textsc{own}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}^{k\prime\prime}\ lck) = \mathrm{T}')$, it must be that $t^a_{\mathrm{T}'} \in \gamma_t(\textsc{accTime}(\langle[\mathrm{T}, pc^{\tilde{c}^k}_{\mathrm{T}}, \tilde{\mathbb{r}}^k_{\mathrm{T}}, \tilde{t}^{a^k}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^k}}, \tilde{\mathbb{x}}^k, \tilde{\mathbb{l}}^{k\prime\prime}\rangle, \mathbf{Thrd}^{\tilde{c}^k}_{exe}, \mathrm{T}'))$ whenever $\forall lck \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}', pc^n_{\mathrm{T}'}) = [\texttt{lock}\ lck]^{pc^n_{\mathrm{T}'}} \Rightarrow (\textsc{own}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}^{k\prime\prime}\ lck) = \mathrm{T}' \wedge \textsc{own}(\mathbb{l}^{n\prime\prime}\ lck) = \mathrm{T}'))$ (Lemma 5.54), where $t^a_{\mathrm{T}'}$

is derived from $c^n \xrightarrow[prg]{} \langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle$ and $\mathbb{l}^{n\prime\prime}$ and $\tilde{\mathbb{l}}^{k\prime\prime}$ are defined as in Tables 4.3 and 5.6, respectively.

- Since $\forall lck \in \mathbf{Lck} : (\textsc{own}(\mathbb{l}^0\ lck) = \bot_{thrd} \Rightarrow (\textsc{stm}(\mathrm{T}', pc^0_{\mathrm{T}'}) = [\mathtt{lock}\ lck]^{pc^0_{\mathrm{T}'}} \Rightarrow (\mathbb{l}^n\ lck = \mathbb{l}^0\ lck \wedge \tilde{\mathbb{l}}^k\ lck = \tilde{\mathbb{l}}^0\ lck)))$, $\forall i \in \{0,\ldots,n-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{c^i}_{exe}$, $\textsc{stm}(\mathrm{T}', pc^n_{\mathrm{T}'}) \neq [\mathtt{halt}]^{pc^n_{\mathrm{T}'}} \Rightarrow \mathrm{T}' \in \mathbf{Thrd}^{c^n}_{exe}$ and $\tilde{\textsc{own}}(\tilde{\mathbb{l}}^{k\prime\prime}\ lck) = \mathrm{T}'$, it must be that T' immediately acquires lck (i.e., without any other thread acquiring and possibly releasing lck in the sequence $c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^n$) if $\textsc{stm}(\mathrm{T}', pc^{\tilde{c}^k}_{\mathrm{T}'}) = [\mathtt{lock}\ lck]^{pc^{\tilde{c}^k}_{\mathrm{T}'}}$, both in the concrete and abstract cases (based on c^n and $\tilde{c}^k$).

- If, for some lock, $lck' \in \mathbf{Lck}$, $\textsc{stm}(\mathrm{T}', pc^{\tilde{c}^0}_{\mathrm{T}'}) = [\mathtt{lock}\ lck']^{pc^{\tilde{c}^0}_{\mathrm{T}'}}$ and $\textsc{own}(\mathbb{l}^0\ lck') = \bot_{thrd}$, it must be that $\min(\{t^{a^n}_{\mathrm{T}} + \textsc{time}(c^n, \mathrm{T}) \mid \mathrm{T} \in \mathbf{Thrd}\}) \in \gamma_t(\textsc{dllock}(\tilde{c}^j, lck'))$, since $\mathrm{T}' \in \mathbf{Thrd}_{\tilde{c}^k}$, $\mathbf{Thrd}_{\tilde{c}^k} \subseteq \mathbf{Thrd}_{\tilde{c}^0} \subseteq \mathbf{Thrd}$, m, n and j (c.f., Lemma 5.53) can be chosen to be 0, n and k (given by this proof), respectively, $t^{a^0}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a^0}_{\mathrm{T}'})$, $t^{a^n}_{\mathrm{T}'} = t^{a^0}_{\mathrm{T}'}$, $\tilde{t}^{a^k}_{\mathrm{T}'} = \tilde{t}^{a^0}_{\mathrm{T}'}$, $pc^{\tilde{c}^k}_{\mathrm{T}'} = pc^{\tilde{c}^0}_{\mathrm{T}'} = pc^n_{\mathrm{T}'} = pc^0_{\mathrm{T}'}$, $\mathrm{T}' \in \mathbf{Thrd}^{c^n}_{exe}$ and (since $\forall lck \in \mathbf{Lck} : (\textsc{own}(\mathbb{l}^0\ lck) = \bot_{thrd} \Rightarrow (\textsc{stm}(\mathrm{T}', pc^0_{\mathrm{T}'}) = [\mathtt{lock}\ lck]^{pc^0_{\mathrm{T}'}} \Rightarrow (\mathbb{l}^n\ lck = \mathbb{l}^0\ lck \wedge \tilde{\mathbb{l}}^k\ lck = \tilde{\mathbb{l}}^0\ lck)))$ and $\textsc{rel}(\mathbb{l}^0\ lck') \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^0\ lck')))\ \textsc{rel}(\mathbb{l}^0\ lck') \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^k\ lck'))$ (Lemma 5.53). Thus, $t^{a^n}_{\mathrm{T}'} + \textsc{time}(c^n, \mathrm{T}') \in \gamma_t(\textsc{dllock}(\tilde{c}^k, lck'))$, since $\mathrm{T}' \in \mathbf{Thrd}^{c^n}_{exe}$ which means that $t^{a^n}_{\mathrm{T}'} + \textsc{time}(c^n, \mathrm{T}') = \min(\{t^{a^n}_{\mathrm{T}} + \textsc{time}(c^n, \mathrm{T}) \mid \mathrm{T} \in \mathbf{Thrd}\})$ (c.f., Table 4.3).

The proof will now be conducted by considering the different statements that T' could issue in c^0 (i.e., in c^n).

1. If $\textsc{stm}(\mathrm{T}', pc^0_{\mathrm{T}'}) = [\mathtt{halt}]^{pc^0_{\mathrm{T}'}}$, then it must be that $\mathrm{T}' \notin \mathbf{Thrd}^{c^n}_{exe}$. Thus, it must be that $c^n \xrightarrow[prg]{} c$, where $c@\langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle$ is such that $pc_{\mathrm{T}'} = pc^n_{\mathrm{T}'}$, $\mathbb{r}_{\mathrm{T}'} = \mathbb{r}^n_{\mathrm{T}'}$, $t^a_{\mathrm{T}'} = t^{a^n}_{\mathrm{T}'}$, $\forall lck \in \mathbf{Lck} : (\textsc{own}(\mathbb{l}^0\ lck) = \mathrm{T}' \Rightarrow \mathbb{l}\ lck = \mathbb{l}^n\ lck)$ and $\forall x \in \mathbf{Var} : ((\mathbb{x}\ x)\ \mathrm{T}') \subseteq ((\mathbb{x}^n\ x)\ \mathrm{T}')$, provided that $\exists \mathrm{T} \in \mathbf{Thrd} : \textsc{stm}(\mathrm{T}, pc^0_{\mathrm{T}}) \neq [\mathtt{halt}]^{pc^0_{\mathrm{T}}}$ (otherwise $\xrightarrow[prg]{}$ is not applicable; c.f., Table 4.3).

   Note that $\mathrm{T}' \notin \mathbf{Thrd}^{\tilde{c}^k}_{exe}$ and choose $\tilde{c}@\langle [\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^k}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle$ such that $\tilde{c}^k \xrightarrow[prg]{\sim} \tilde{c}$, i.e., $pc^{\tilde{c}}_{\mathrm{T}'} = pc^{\tilde{c}^k}_{\mathrm{T}'}$, $\tilde{\mathbb{r}}_{\mathrm{T}'} = \tilde{\mathbb{r}}^k_{\mathrm{T}'}$, $\tilde{t}^a_{\mathrm{T}'} = \tilde{t}^{a^k}_{\mathrm{T}'}$, $\forall lck \in \mathbf{Lck} :$

$(\textsc{own}(\mathbb{l}^0\ lck) = \mathrm{T}' \Rightarrow (\tilde{\mathbb{l}}\ lck = \tilde{\mathbb{l}}^k\ lck \wedge \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}\ lck))) = -\infty))$. Note that $\tilde{\mathbb{x}}$ must still be such that for all $x \in \mathbf{Var}$, $((\tilde{\mathbb{x}}\ x)\ \mathrm{T}')$ is a safe approximation of the writes performed on x by T' since TRIM is safe (Lemma 5.27). Thus, it must be that:

$$
\begin{array}{l}
pc_{\mathrm{T}'} = pc^{\tilde{c}}_{\mathrm{T}'} \wedge \\
\mathbb{r}_{\mathrm{T}'} \in \gamma_{reg}(\tilde{\mathbb{r}}_{\mathrm{T}'}) \wedge \\
t^a_{\mathrm{T}'} \in \gamma_t(\tilde{t}^a_{\mathrm{T}'}) \wedge \\
\exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}) : (\forall x \in \mathbf{Var} : ((\mathbb{x}\ x)\ \mathrm{T}') \subseteq ((\mathbb{x}'\ x)\ \mathrm{T}')) \wedge \\
\forall lck \in \mathbf{Lck} : ((\textsc{own}(\mathbb{l}^0\ lck) = \mathrm{T}' \vee \textsc{own}(\mathbb{l}\ lck) = \mathrm{T}') \Rightarrow \\
\qquad\qquad (\textsc{stt}(\mathbb{l}\ lck) = \textsc{s}\tilde{\textsc{t}}\textsc{t}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\qquad\ \textsc{own}(\mathbb{l}\ lck) = \textsc{o}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\qquad\ \textsc{dl}(\mathbb{l}\ lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}\ lck)) \wedge \\
\qquad\qquad\ \textsc{pown}(\mathbb{l}\ lck) = \textsc{po}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\qquad\ \textsc{rel}(\mathbb{l}\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}\ lck)) \wedge \\
\qquad\qquad\ \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}\ lck))) = -\infty))
\end{array}
$$

2. If, for some $a \in \mathbf{Aexp}$, $b \in \mathbf{Bexp}$, $l \in \mathbf{Lbl}_{\mathrm{T}'}$, $r \in \mathbf{Reg}_{\mathrm{T}'}$, $x \in \mathbf{Var}$ and $lck \in \mathbf{Lck}$, $\textsc{stm}(\mathrm{T}', pc^0_{\mathrm{T}'}) \in \{[\texttt{skip}]^{pc^0_{\mathrm{T}'}}, [r := a]^{pc^0_{\mathrm{T}'}}, [\texttt{if}\ b\ \texttt{goto}\ l]^{pc^0_{\mathrm{T}'}}, [\texttt{store}\ r\ \texttt{to}\ x]^{pc^0_{\mathrm{T}'}}, [\texttt{unlock}\ lck]^{pc^0_{\mathrm{T}'}}\}$, then let the configuration $c@\langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle$ be such that $c^n \xrightarrow[prg]{} c$ and choose $\tilde{c}@\langle [\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^a_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}^k}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle$ such that $\tilde{c}^k \xrightarrow[prg]{\sim} \tilde{c}$. Thus, since $\forall i \in \{0, \ldots, n-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{c^i}_{exe}$, $\mathrm{T}' \in \mathbf{Thrd}^{c^n}_{exe}$, $\forall i \in \{0, \ldots, k-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{\tilde{c}^i}_{exe}$, $\mathrm{T}' \in \mathbf{Thrd}^{\tilde{c}^k}_{exe}$, $\xrightarrow[ax]{\sim}$ is a safe approximation of $\xrightarrow[ax]{}$ (Lemma 5.49), TRIM is safe (Lemma 5.27), $\mathbf{Thrd}_{\tilde{c}^k} \subseteq \mathbf{Thrd}$ and ACCTIME is safe (Lemma 5.54), it must be that:

$$
\begin{array}{l}
pc_{\mathrm{T}'} = pc^{\tilde{c}}_{\mathrm{T}'} \wedge \\
\mathbb{r}_{\mathrm{T}'} \in \gamma_{reg}(\tilde{\mathbb{r}}_{\mathrm{T}'}) \wedge \\
t^a_{\mathrm{T}'} \in \gamma_t(\tilde{t}^a_{\mathrm{T}'}) \wedge \\
\exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}) : (\forall x \in \mathbf{Var} : ((\mathbb{x}\ x)\ \mathrm{T}') \subseteq ((\mathbb{x}'\ x)\ \mathrm{T}')) \wedge \\
\forall lck \in \mathbf{Lck} : ((\textsc{own}(\mathbb{l}^0\ lck) = \mathrm{T}' \vee \textsc{own}(\mathbb{l}\ lck) = \mathrm{T}') \Rightarrow \\
\qquad\qquad (\textsc{stt}(\mathbb{l}\ lck) = \textsc{s}\tilde{\textsc{t}}\textsc{t}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\qquad\ \textsc{own}(\mathbb{l}\ lck) = \textsc{o}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\qquad\ \textsc{dl}(\mathbb{l}\ lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}\ lck)) \wedge \\
\qquad\qquad\ \textsc{pown}(\mathbb{l}\ lck) = \textsc{po}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\qquad\ \textsc{rel}(\mathbb{l}\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}\ lck)) \wedge \\
\qquad\qquad\ \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}\ lck))) = -\infty))
\end{array}
$$

Note that in the case $\textsc{stm}(\mathrm{T}', pc^0_{\mathrm{T}'}) = [\texttt{if } b \texttt{ goto } l]^{pc^0_{\mathrm{T}'}}$, $\tilde{c}$ can be chosen so that the corresponding branch to that taken in c is taken since $\mathbb{r}^0_{\mathrm{T}} \in \gamma_{reg}(\tilde{\mathbb{r}}^0_{\mathrm{T}})$ (c.f., Table 5.5 and Definition 5.7).

3. If, for some $r \in \mathbf{Reg}_{\mathrm{T}'}$ and $x \in \mathbf{Var}$, $\textsc{stm}(\mathrm{T}', pc^0_{\mathrm{T}'}) = [\texttt{load } r \texttt{ from } x]^{pc^0_{\mathrm{T}'}}$, then let $c@\langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle$ be such that $c^n \xrightarrow[prg]{} c$ and choose $\tilde{c}@\langle [\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^k}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle$ such that $\tilde{c}^k \xrightarrow[prg]{\sim} \tilde{c}$. Since $\forall i \in \{0,\ldots,n-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{c^i}_{exe}$, $\mathrm{T}' \in \mathbf{Thrd}^{c^n}_{exe}$, $\forall i \in \{0,\ldots,k-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{\tilde{c}^i}_{exe}$, $\mathrm{T}' \in \mathbf{Thrd}^{\tilde{c}^k}_{exe}$, $\xrightarrow[ax]{\sim}$ is a safe approximation of $\xrightarrow[ax]{}$ (Lemma 5.49), $\tilde{\mathbb{x}}^k$ contains safe write history, TRIM is safe (Lemma 5.27), $\mathbf{Thrd}_{\tilde{c}^k} \subseteq \mathbf{Thrd}$ and ACCTIME is safe (Lemma 5.54), it must be that:

$$
\begin{array}{l}
pc_{\mathrm{T}'} = pc^{\tilde{c}}_{\mathrm{T}'} \wedge \\
\mathbb{r}_{\mathrm{T}'} \in \gamma_{reg}(\tilde{\mathbb{r}}_{\mathrm{T}'}) \wedge \\
t^a_{\mathrm{T}'} \in \gamma_t(\tilde{t}^a_{\mathrm{T}'}) \wedge \\
\exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}) : (\forall x \in \mathbf{Var} : ((\mathbb{x}\ x)\ \mathrm{T}') \subseteq ((\mathbb{x}'\ x)\ \mathrm{T}')) \wedge \\
\forall lck \in \mathbf{Lck} : ((\textsc{own}(\mathbb{l}^0\ lck) = \mathrm{T}' \vee \textsc{own}(\mathbb{l}\ lck) = \mathrm{T}') \Rightarrow \\
\qquad\qquad (\textsc{stt}(\mathbb{l}\ lck) = \textsc{s}\tilde{\textsc{t}}\textsc{t}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\qquad\ \textsc{own}(\mathbb{l}\ lck) = \textsc{o}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\qquad\ \textsc{dl}(\mathbb{l}\ lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}\ lck)) \wedge \\
\qquad\qquad\ \textsc{pown}(\mathbb{l}\ lck) = \textsc{po}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\qquad\ \textsc{rel}(\mathbb{l}\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}\ lck)) \wedge \\
\qquad\qquad\ \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}\ lck))) = -\infty))
\end{array}
$$

4. If, for some $lck' \in \mathbf{Lck}$, $\textsc{stm}(\mathrm{T}', pc^0_{\mathrm{T}'}) = [\texttt{lock } lck']^{pc^0_{\mathrm{T}'}}$, only the case that T' successfully and immediately acquires lck' needs to be considered. (Note that the remaining cases will be considered in the proofs of Lemmas 5.57 and 5.58.) Hence, ACCTIME is safe since it must be that $\textsc{o}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}^{k''}\ lck') = \mathrm{T}'$ and $\textsc{own}(\mathbb{l}^{n''}\ lck') = \mathrm{T}'$ (Lemma 5.54).

   Since $\textsc{own}(\mathbb{l}^0\ lck') = \mathrm{T}' \Rightarrow \textsc{o}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}^0\ lck') = \mathrm{T}'$ and $\textsc{own}(\mathbb{l}^0\ lck') = \bot_{thrd} \Rightarrow (\textsc{o}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}^0\ lck') = \bot_{thrd} \vee \textsc{o}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}^0\ lck') = \mathrm{T}')$, there are three cases to consider.

   (a) Assume that $\textsc{own}(\mathbb{l}^0\ lck') = \mathrm{T}'$ (and thus, $\textsc{o}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}^0\ lck') = \mathrm{T}'$) and let $c@\langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle$ be such that $c^n \xrightarrow[prg]{} c$. Then choose $\tilde{c}@\langle [\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^k}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle$ such that $\tilde{c}^k \xrightarrow[prg]{\sim} \tilde{c}$. It is

trivially the case that $\text{OWN}(\mathbb{l}\ lck') = \text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck') = \text{T}'$ and thus

$$
\begin{aligned}
&pc_{\text{T}'} = pc^{\tilde{c}}_{\text{T}'} \wedge \\
&\mathbb{r}_{\text{T}'} \in \gamma_{reg}(\tilde{\mathbb{r}}_{\text{T}'}) \wedge \\
&t^a_{\text{T}'} \in \gamma_t(\tilde{t}^a_{\text{T}'}) \wedge \\
&\exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}) : (\forall x \in \mathbf{Var} : ((\mathbb{x}\ x)\ \text{T}') \subseteq ((\mathbb{x}'\ x)\ \text{T}')) \wedge \\
&\forall lck \in \mathbf{Lck} : ((\text{OWN}(\mathbb{l}^0\ lck) = \text{T}' \vee \text{OWN}(\mathbb{l}\ lck) = \text{T}') \Rightarrow \\
&\qquad (\text{STT}(\mathbb{l}\ lck) = \text{S}\tilde{\text{T}}\text{T}(\tilde{\mathbb{l}}\ lck) \wedge \\
&\qquad\ \text{OWN}(\mathbb{l}\ lck) = \text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck) \wedge \\
&\qquad\ \text{DL}(\mathbb{l}\ lck) \in \gamma_t(\tilde{\text{DL}}(\tilde{\mathbb{l}}\ lck)) \wedge \\
&\qquad\ \text{POWN}(\mathbb{l}\ lck) = \text{PO}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck) \wedge \\
&\qquad\ \text{REL}(\mathbb{l}\ lck) \in \gamma_t(\tilde{\text{REL}}(\tilde{\mathbb{l}}\ lck)) \wedge \\
&\qquad\ \min(\gamma_t(\tilde{\text{DL}}(\tilde{\mathbb{l}}\ lck))) = -\infty))
\end{aligned}
$$

since $\forall i \in \{0,\ldots,n-1\} : \text{T}' \notin \mathbf{Thrd}^{c^i}_{exe}$, $\text{T}' \in \mathbf{Thrd}^{c^n}_{exe}$, $\forall i \in \{0,\ldots,k-1\} : \text{T}' \notin \mathbf{Thrd}^{\tilde{c}^i}_{exe}$, $\text{T}' \in \mathbf{Thrd}^{\tilde{c}^k}_{exe}$, $\xrightarrow[ax]{\sim}$ is a safe approximation of $\xrightarrow[ax]{}$ (Lemma 5.49) and TRIM is safe (Lemma 5.27).

(b) Assume that $\text{OWN}(\mathbb{l}^0\ lck') = \text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}^0\ lck') = \bot_{thrd}$ and let $c@\langle[\text{T}, pc_{\text{T}}, \mathbb{r}_{\text{T}}, t^a_{\text{T}}]_{\text{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle$ be such that $c^n \xrightarrow[prg]{} c \wedge \text{OWN}(\mathbb{l}\ lck') = \text{T}'$ and choose $\tilde{c}@\langle[\text{T}, pc^{\tilde{c}}_{\text{T}}, \tilde{\mathbb{r}}_{\text{T}}, \tilde{t}^a_{\text{T}}]_{\text{T}\in\mathbf{Thrd}_{\tilde{c}^k}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle$ such that $\tilde{c}^k \xrightarrow[prg]{\sim} \tilde{c}$. Then it must be that $\text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck') = \text{T}'$ (since $\text{T}' \in \mathbf{Thrd}^{\tilde{c}^k}_{exe}$ and thus $\text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}^{k\prime\prime}\ lck') = \text{T}'$) and

$$
\begin{aligned}
&pc_{\text{T}'} = pc^{\tilde{c}}_{\text{T}'} \wedge \\
&\mathbb{r}_{\text{T}'} \in \gamma_{reg}(\tilde{\mathbb{r}}_{\text{T}'}) \wedge \\
&t^a_{\text{T}'} \in \gamma_t(\tilde{t}^a_{\text{T}'}) \wedge \\
&\exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}) : (\forall x \in \mathbf{Var} : ((\mathbb{x}\ x)\ \text{T}') \subseteq ((\mathbb{x}'\ x)\ \text{T}')) \wedge \\
&\forall lck \in \mathbf{Lck} : ((\text{OWN}(\mathbb{l}^0\ lck) = \text{T}' \vee \text{OWN}(\mathbb{l}\ lck) = \text{T}') \Rightarrow \\
&\qquad (\text{STT}(\mathbb{l}\ lck) = \text{S}\tilde{\text{T}}\text{T}(\tilde{\mathbb{l}}\ lck) \wedge \\
&\qquad\ \text{OWN}(\mathbb{l}\ lck) = \text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck) \wedge \\
&\qquad\ \text{DL}(\mathbb{l}\ lck) \in \gamma_t(\tilde{\text{DL}}(\tilde{\mathbb{l}}\ lck)) \wedge \\
&\qquad\ \text{POWN}(\mathbb{l}\ lck) = \text{PO}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck) \wedge \\
&\qquad\ \text{REL}(\mathbb{l}\ lck) \in \gamma_t(\tilde{\text{REL}}(\tilde{\mathbb{l}}\ lck)) \wedge \\
&\qquad\ \min(\gamma_t(\tilde{\text{DL}}(\tilde{\mathbb{l}}\ lck))) = -\infty))
\end{aligned}
$$

since $\forall i \in \{0,\ldots,n-1\} : \text{T}' \notin \mathbf{Thrd}^{c^i}_{exe}$, $\text{T}' \in \mathbf{Thrd}^{c^n}_{exe}$, $\forall i \in \{0,\ldots,k-1\} : \text{T}' \notin \mathbf{Thrd}^{\tilde{c}^i}_{exe}$, $\text{T}' \in \mathbf{Thrd}^{\tilde{c}^k}_{exe}$, $\xrightarrow[ax]{\sim}$ is a safe approxi-

mation of $\xrightarrow[ax]{}$ (Lemma 5.49), TRIM is safe (Lemma 5.27), $\mathbf{Thrd}_{\tilde{c}^k} \subseteq \mathbf{Thrd}$, $\forall \tilde{c}' \in \tilde{\mathbf{Conf}} : \forall lck \in \mathbf{Lck} : \min(\text{DLLOCK}(\tilde{c}', lck)) = -\infty$ (Algorithm 5.11), $\text{DL}(\mathbb{l}\ lck') = t^a_{\mathrm{T}'} = t^{a^n}_{\mathrm{T}'} + \text{TIME}(c^n, \mathrm{T}')$ (Table 4.3) and $t^a_{\mathrm{T}'} \in \gamma_t(\text{DLLOCK}(\tilde{c}^k, lck'))$ (Lemma 5.53).

(c) Assume that $\text{OWN}(\mathbb{l}^0\ lck') = \bot_{thrd}$ and $\text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}^0\ lck') = \mathrm{T}'$ and let $c@\langle[\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle$ be such that $c^n \xrightarrow[prg]{} c \wedge \text{OWN}(\mathbb{l}\ lck') = \mathrm{T}'$ and choose $\tilde{c}@\langle[\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^k}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle$ such that $\tilde{c}^k \xrightarrow[prg]{\sim} \tilde{c}$. Then it is easy to see that $\text{OWN}(\mathbb{l}\ lck') = \text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck') = \mathrm{T}'$.

First note that since $\text{OWN}(\mathbb{l}^n\ lck') = \text{OWN}(\mathbb{l}^0\ lck') = \bot_{thrd} \wedge \text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}^k\ lck') = \text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}^0\ lck') = \mathrm{T}'$, it must be that $\text{STT}(\mathbb{l}^n\ lck') = \tilde{\text{STT}}(\tilde{\mathbb{l}}^k\ lck') = \tilde{\text{STT}}(\tilde{\mathbb{l}}^0\ lck') = \mathit{unlocked}$, $\tilde{\text{DL}}(\tilde{\mathbb{l}}\ lck') = \tilde{\text{DL}}(\tilde{\mathbb{l}}^{k''}\ lck') = \tilde{\text{DL}}(\tilde{\mathbb{l}}^k\ lck') = \tilde{\text{DL}}(\tilde{\mathbb{l}}^0\ lck')$, and thus $\min(\gamma_t(\tilde{\text{DL}}(\tilde{\mathbb{l}}\ lck'))) = -\infty$ and $t^{a^n}_{\mathrm{T}'} + \text{TIME}(c^n, \mathrm{T}') \in \gamma_t(\tilde{\text{DL}}(\tilde{\mathbb{l}}\ lck'))$, which means that $\text{DL}(\tilde{\mathbb{l}}\ lck') \in \gamma_t(\tilde{\text{DL}}(\tilde{\mathbb{l}}\ lck'))$ since $\text{DL}(\tilde{\mathbb{l}}\ lck') = t^{a^n}_{\mathrm{T}'} + \text{TIME}(c^n, \mathrm{T}')$.

Note that since $t^{a^n}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a^k}_{\mathrm{T}'})$, it must be that $\tilde{\text{DL}}(\tilde{\mathbb{l}}^{k''}\ lck') \not\preceq_t \tilde{t}^{a^k}_{\mathrm{T}'} \dot{+}_t \text{ABSTIME}(\tilde{c}^k, \mathrm{T}')$. It is thus easy to see that

$$
\begin{array}{l}
pc_{\mathrm{T}'} = pc^{\tilde{c}}_{\mathrm{T}'} \wedge \\
\mathbb{r}_{\mathrm{T}'} \in \gamma_{reg}(\tilde{\mathbb{r}}_{\mathrm{T}'}) \wedge \\
t^a_{\mathrm{T}'} \in \gamma_t(\tilde{t}^a_{\mathrm{T}'}) \wedge \\
\exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}) : (\forall x \in \mathbf{Var} : ((\mathbb{x}\ x)\ \mathrm{T}') \subseteq ((\mathbb{x}'\ x)\ \mathrm{T}')) \wedge \\
\forall lck \in \mathbf{Lck} : ((\text{OWN}(\mathbb{l}^0\ lck) = \mathrm{T}' \vee \text{OWN}(\mathbb{l}\ lck) = \mathrm{T}') \Rightarrow \\
\qquad (\text{STT}(\mathbb{l}\ lck) = \tilde{\text{STT}}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\ \text{OWN}(\mathbb{l}\ lck) = \text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\ \text{DL}(\mathbb{l}\ lck) \in \gamma_t(\tilde{\text{DL}}(\tilde{\mathbb{l}}\ lck)) \wedge \\
\qquad\ \text{POWN}(\mathbb{l}\ lck) = \text{PO}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\ \text{REL}(\mathbb{l}\ lck) \in \gamma_t(\text{R}\tilde{\text{EL}}(\tilde{\mathbb{l}}\ lck)) \wedge \\
\qquad\ \min(\gamma_t(\tilde{\text{DL}}(\tilde{\mathbb{l}}\ lck))) = -\infty))
\end{array}
$$

since $\forall i \in \{0, \ldots, n-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{c^i}_{exe}$, $\mathrm{T}' \in \mathbf{Thrd}^{c^n}_{exe}$, $\forall i \in \{0, \ldots, k-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{\tilde{c}^i}_{exe}$, $\mathrm{T}' \in \mathbf{Thrd}^{\tilde{c}^k}_{exe}$, $\xrightarrow[ax]{\sim}$ is a safe approximation of $\xrightarrow[ax]{}$ (Lemma 5.49), TRIM is safe (Lemma 5.27) and $\mathbf{Thrd}_{\tilde{c}^k} \subseteq \mathbf{Thrd}$.

This concludes the proof. ∎

Lemma 5.57 shows that $\xrightarrow[prg]{\sim}$ safely approximates the case that a thread issuing `lock` *lck* for some lock, $lck \in \mathbf{Lck}$, has to wait for an arbitrary number of owner switches on *lck* before it acquires *lck*. Note that the lemma holds if all threads wanting to acquire some lock eventually will be able to do so (which obviously is the case if the concrete transition sequences are finite in length) and if either no thread issues a `load`-statement on a global variable or that the thread issuing the `load`-statement is the sole thread in $\mathbf{Thrd}_{exe}$ in any step of the transition sequence.

Lemma 5.57 (Soundness of $\xrightarrow[prg]{\sim}$, frozen thread):

Given the valid concrete configurations (c.f., Definition 4.4), abstract configurations, lock and thread

$$
\begin{aligned}
&c^0 \,@\, \langle [\mathrm{T}, pc_{\mathrm{T}}^0, \mathbb{r}_{\mathrm{T}}^0, t_{\mathrm{T}}^{a^0}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^0, \mathbb{l}^0 \rangle \in \mathbf{Conf},\\
&c^{n_1} \,@\, \langle [\mathrm{T}, pc_{\mathrm{T}}^{n_1}, \mathbb{r}_{\mathrm{T}}^{n_1}, t_{\mathrm{T}}^{a^{n_1}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^{n_1}, \mathbb{l}^{n_1} \rangle \in \mathbf{Conf},\\
&c^{n_2} \,@\, \langle [\mathrm{T}, pc_{\mathrm{T}}^{n_2}, \mathbb{r}_{\mathrm{T}}^{n_2}, t_{\mathrm{T}}^{a^{n_2}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^{n_2}, \mathbb{l}^{n_2} \rangle \in \mathbf{Conf},\\
&\quad\vdots\\
&c^{n_m} \,@\, \langle [\mathrm{T}, pc_{\mathrm{T}}^{n_m}, \mathbb{r}_{\mathrm{T}}^{n_m}, t_{\mathrm{T}}^{a^{n_m}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^{n_m}, \mathbb{l}^{n_m} \rangle \in \mathbf{Conf},\\
&c^{n} \,@\, \langle [\mathrm{T}, pc_{\mathrm{T}}^{n}, \mathbb{r}_{\mathrm{T}}^{n}, t_{\mathrm{T}}^{a^{n}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^{n}, \mathbb{l}^{n} \rangle \in \mathbf{Conf},\\
&\tilde{c}^0 \,@\, \langle [\mathrm{T}, pc_{\mathrm{T}}^{\tilde{c}^0}, \tilde{\mathbb{r}}_{\mathrm{T}}^0, \tilde{t}_{\mathrm{T}}^{a^0}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^0}}, \tilde{\mathbb{x}}^0, \tilde{\mathbb{l}}^0 \rangle \in \tilde{\mathbf{Conf}},\\
&\tilde{c}^k \,@\, \langle [\mathrm{T}, pc_{\mathrm{T}}^{\tilde{c}^k}, \tilde{\mathbb{r}}_{\mathrm{T}}^k, \tilde{t}_{\mathrm{T}}^{a^k}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^k}}, \tilde{\mathbb{x}}^k, \tilde{\mathbb{l}}^k \rangle \in \tilde{\mathbf{Conf}},\\
&lck' \in \mathbf{Lck}, \text{ and}\\
&\mathrm{T}' \in \mathbf{Thrd}_{\tilde{c}^k},
\end{aligned}
$$

such that

$$\begin{gathered}
\textsc{stm}(\mathrm{T}', pc^0_{\mathrm{T}'}) = [\texttt{lock } lck']^{pc^0_{\mathrm{T}'}},\\
0 \le n_1 \le n_2 \le n_m \le n,\\
c^0 \xrightarrow[prg]{} \dots \xrightarrow[prg]{} c^{n_1} \xrightarrow[prg]{} \dots \xrightarrow[prg]{} c^{n_2} \xrightarrow[prg]{} \dots \xrightarrow[prg]{} c^{n_m} \xrightarrow[prg]{} \dots \xrightarrow[prg]{} c^n,\\
0 \le k,\\
\tilde{c}^0 \xrightarrow[prg]{\sim} \dots \xrightarrow[prg]{\sim} \tilde{c}^k,\\
\mathbf{Thrd}_{\tilde{c}^k} \subseteq \mathbf{Thrd}_{\tilde{c}^0} \subseteq \mathbf{Thrd},\\
pc^0_{\mathrm{T}'} = pc^{\tilde{c}^0}_{\mathrm{T}'},\\
\mathbb{r}^0_{\mathrm{T}'} \in \gamma_{reg}(\tilde{\mathbb{r}}^0_{\mathrm{T}'}),\\
t^{a^0}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a^0}_{\mathrm{T}'}),\\
\exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}^0) : \forall x \in \mathbf{Var} : \forall \mathrm{T} \in \mathbf{Thrd} : ((\mathbb{x}^0\ x)\ \mathrm{T}) \subseteq ((\mathbb{x}'\ x)\ \mathrm{T}),\\
\forall lck \in \mathbf{Lck} : ((\textsc{own}(\mathbb{l}^0\ lck) \neq \bot_{thrd} \Rightarrow (\textsc{stt}(\mathbb{l}^0\ lck) = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^0\ lck) \wedge\\
\qquad \textsc{own}(\mathbb{l}^0\ lck) = \tilde{\textsc{own}}(\tilde{\mathbb{l}}^0\ lck) \wedge\\
\qquad \textsc{dl}(\mathbb{l}^0\ lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck)) \wedge\\
\qquad \textsc{pown}(\mathbb{l}^0\ lck) = \tilde{\textsc{pown}}(\tilde{\mathbb{l}}^0\ lck) \wedge\\
\qquad \textsc{rel}(\mathbb{l}^0\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^0\ lck)) \wedge\\
\qquad \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck))) = -\infty)) \wedge\\
\quad (\textsc{own}(\mathbb{l}^0\ lck) = \bot_{thrd} \Rightarrow ((\textsc{own}(\mathbb{l}^0\ lck) = \tilde{\textsc{own}}(\tilde{\mathbb{l}}^0\ lck) \vee\\
\qquad (\tilde{\textsc{own}}(\tilde{\mathbb{l}}^0\ lck) = \mathrm{T}' \wedge\\
\qquad\ \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^0\ lck) = \mathit{unlocked} \wedge\\
\qquad\ t^{a^n}_{\mathrm{T}'} + \textsc{time}(c^n, \mathrm{T}') \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck)) \wedge\\
\qquad\ \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck))) = -\infty)) \wedge\\
\qquad \textsc{pown}(\mathbb{l}^0\ lck) = \tilde{\textsc{pown}}(\tilde{\mathbb{l}}^0\ lck) \wedge\\
\qquad \textsc{rel}(\mathbb{l}^0\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^0\ lck))))),\\
\forall i \in \{0,\dots,n-1\} \setminus \{n_1, n_2, \dots, n_m\} : \mathrm{T}' \notin \mathbf{Thrd}^{c^i}_{exe},\\
\forall i \in \{n_1, n_2, \dots, n_m\} : (\mathrm{T}' \in \mathbf{Thrd}^{c^i}_{exe} \wedge \textsc{own}(\mathbb{l}^{i''}\ lck') \neq \mathrm{T}'),\\
\mathrm{T}' \in \mathbf{Thrd}^{c^n}_{exe},\\
\forall i \in \{0,\dots,k-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{\tilde{c}^i}_{exe},\\
\mathrm{T}' \in \mathbf{Thrd}^{\tilde{c}^k}_{exe},\ \textit{and}\\
\forall i \in \{0,\dots,k\} : (|\mathbf{Thrd}^{\tilde{c}^i}_{exe}| \not> 1 \vee\\
\qquad \{\mathrm{T} \in \mathbf{Thrd}^{\tilde{c}^i}_{exe} \mid \exists r \in \mathbf{Reg}_{\mathrm{T}} : \exists x \in \mathbf{Var_g} :\\
\qquad\qquad \textsc{stm}(\mathrm{T}, pc^{\tilde{c}^i}_{\mathrm{T}}) = [\texttt{load } r \texttt{ from } x]^{pc^{\tilde{c}^i}_{\mathrm{T}}}\} = \emptyset),
\end{gathered}$$

where for all $i \in \{0,\dots,n\}$, $\mathbf{Thrd}^{c^i}_{exe}$ *is as defined in Table 4.3, for all* $i \in \{0,\dots,k\}$, $\mathbf{Thrd}^{\tilde{c}^i}_{exe}$ *is as defined in Table 5.6, and* $\mathbf{Var_g}$ *contains all* $x \in \mathbf{Var}$ *such that* x *can be written to by one thread and read from by another thread*

(i.e., there might be a data dependency between the threads; note that $\mathbf{Var_g}$ *can be derived using Algorithm 6.9),* $\xrightarrow[prg]{\sim}$ *satisfies:*

$$
\begin{array}{l}
\forall c@\langle[\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle \in \mathbf{Conf}: \\
\quad (c^n \xrightarrow[prg]{} c \Rightarrow \exists \tilde{c}@\langle[\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^k}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \tilde{\mathbf{Conf}}: \\
\qquad (\tilde{c}^k \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c} \wedge \\
\qquad pc_{\mathrm{T}'} = pc^{\tilde{c}}_{\mathrm{T}'} \wedge \\
\qquad \mathbb{r}_{\mathrm{T}'} \in \gamma_{reg}(\tilde{\mathbb{r}}_{\mathrm{T}'}) \wedge \\
\qquad t^a_{\mathrm{T}'} \in \gamma_t(\tilde{t}^a_{\mathrm{T}'}) \wedge \\
\qquad \exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}) : (\forall x \in \mathbf{Var} : ((\mathbb{x}\ x)\ \mathrm{T}') \subseteq ((\mathbb{x}'\ x)\ \mathrm{T}')) \wedge \\
\qquad \forall lck \in \mathbf{Lck} : (\textsc{own}(\mathbb{l}\ lck) = \mathrm{T}' \Rightarrow (\textsc{stt}(\mathbb{l}\ lck) = \textsc{s}\tilde{\textsc{t}}\textsc{t}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\qquad\qquad\qquad \textsc{own}(\mathbb{l}\ lck) = \textsc{o}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\qquad\qquad\qquad \textsc{dl}(\mathbb{l}\ lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}\ lck)) \wedge \\
\qquad\qquad\qquad\qquad \textsc{pown}(\mathbb{l}\ lck) = \textsc{po}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\qquad\qquad\qquad \textsc{rel}(\mathbb{l}\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}\ lck)) \wedge \\
\qquad\qquad\qquad\qquad \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}\ lck))) = -\infty))))
\end{array}
$$

□

PROOF. Assume that the valid concrete configurations (c.f., Definition 4.4), abstract configurations, lock and thread

$$
\begin{array}{l}
c^0@\langle[\mathrm{T}, pc^0_{\mathrm{T}}, \mathbb{r}^0_{\mathrm{T}}, t^{a^0}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^0, \mathbb{l}^0\rangle \in \mathbf{Conf}, \\
c^{n_1}@\langle[\mathrm{T}, pc^{n_1}_{\mathrm{T}}, \mathbb{r}^{n_1}_{\mathrm{T}}, t^{a^{n_1}}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^{n_1}, \mathbb{l}^{n_1}\rangle \in \mathbf{Conf}, \\
c^{n_2}@\langle[\mathrm{T}, pc^{n_2}_{\mathrm{T}}, \mathbb{r}^{n_2}_{\mathrm{T}}, t^{a^{n_2}}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^{n_2}, \mathbb{l}^{n_2}\rangle \in \mathbf{Conf}, \\
\quad \vdots \\
c^{n_m}@\langle[\mathrm{T}, pc^{n_m}_{\mathrm{T}}, \mathbb{r}^{n_m}_{\mathrm{T}}, t^{a^{n_m}}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^{n_m}, \mathbb{l}^{n_m}\rangle \in \mathbf{Conf}, \\
c^{n}@\langle[\mathrm{T}, pc^{n}_{\mathrm{T}}, \mathbb{r}^{n}_{\mathrm{T}}, t^{a^{n}}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}^{n}, \mathbb{l}^{n}\rangle \in \mathbf{Conf}, \\
\tilde{c}^0@\langle[\mathrm{T}, pc^{\tilde{c}^0}_{\mathrm{T}}, \tilde{\mathbb{r}}^0_{\mathrm{T}}, \tilde{t}^{a^0}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^0}}, \tilde{\mathbb{x}}^0, \tilde{\mathbb{l}}^0\rangle \in \tilde{\mathbf{Conf}}, \\
\tilde{c}^k@\langle[\mathrm{T}, pc^{\tilde{c}^k}_{\mathrm{T}}, \tilde{\mathbb{r}}^k_{\mathrm{T}}, \tilde{t}^{a^k}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^k}}, \tilde{\mathbb{x}}^k, \tilde{\mathbb{l}}^k\rangle \in \tilde{\mathbf{Conf}}, \\
lck' \in \mathbf{Lck}, \text{ and} \\
\mathrm{T}' \in \mathbf{Thrd}_{\tilde{c}^k},
\end{array}
$$

are such that

$$\begin{gathered}
\textsc{stm}(\mathrm{T}', pc^0_{\mathrm{T}'}) = [\texttt{lock } lck']^{pc^0_{\mathrm{T}'}},\\
0 \le n_1 \le n_2 \le n_m \le n,\\
c^0 \xrightarrow[prg]{} \dots \xrightarrow[prg]{} c^{n_1} \xrightarrow[prg]{} \dots \xrightarrow[prg]{} c^{n_2} \xrightarrow[prg]{} \dots \xrightarrow[prg]{} c^{n_m} \xrightarrow[prg]{} \dots \xrightarrow[prg]{} c^n,\\
0 \le k,\\
\tilde{c}^0 \xrightarrow[prg]{\sim} \dots \xrightarrow[prg]{\sim} \tilde{c}^k,\\
\mathbf{Thrd}_{\tilde{c}^k} \subseteq \mathbf{Thrd}_{\tilde{c}^0} \subseteq \mathbf{Thrd},\\
pc^0_{\mathrm{T}'} = pc^{\tilde{c}^0}_{\mathrm{T}'},\\
\mathbb{r}^0_{\mathrm{T}'} \in \gamma_{reg}(\tilde{\mathbb{r}}^0_{\mathrm{T}'}),\\
t^{a^0}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a^0}_{\mathrm{T}'}),\\
\exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}^0) : \forall x \in \mathbf{Var} : \forall \mathrm{T} \in \mathbf{Thrd} : ((\mathbb{x}^0\ x)\ \mathrm{T}) \subseteq ((\mathbb{x}'\ x)\ \mathrm{T}),\\
\forall lck \in \mathbf{Lck} : ((\textsc{own}(\mathbb{l}^0\ lck) \neq \bot_{thrd} \Rightarrow (\textsc{stt}(\mathbb{l}^0\ lck) = \textsc{s}\tilde{\textsc{t}}\textsc{t}(\tilde{\mathbb{l}}^0\ lck) \wedge\\
\qquad \textsc{own}(\mathbb{l}^0\ lck) = \textsc{o}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}^0\ lck) \wedge\\
\qquad \textsc{dl}(\mathbb{l}^0\ lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck)) \wedge\\
\qquad \textsc{pown}(\mathbb{l}^0\ lck) = \textsc{po}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}^0\ lck) \wedge\\
\qquad \textsc{rel}(\mathbb{l}^0\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^0\ lck)) \wedge\\
\qquad \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck))) = -\infty)) \wedge\\
(\textsc{own}(\mathbb{l}^0\ lck) = \bot_{thrd} \Rightarrow ((\textsc{own}(\mathbb{l}^0\ lck) = \textsc{o}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}^0\ lck) \vee\\
\qquad (\textsc{o}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}^0\ lck) = \mathrm{T}' \wedge\\
\qquad\ \textsc{s}\tilde{\textsc{t}}\textsc{t}(\tilde{\mathbb{l}}^0\ lck) = \mathit{unlocked} \wedge\\
\qquad\ t^{a^n}_{\mathrm{T}'} + \textsc{time}(c^n, \mathrm{T}') \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck)) \wedge\\
\qquad\ \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck))) = -\infty)) \wedge\\
\qquad \textsc{pown}(\mathbb{l}^0\ lck) = \textsc{po}\tilde{\textsc{w}}\textsc{n}(\tilde{\mathbb{l}}^0\ lck) \wedge\\
\qquad \textsc{rel}(\mathbb{l}^0\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^0\ lck))))),\\
\forall i \in \{0,\dots,n-1\} \setminus \{n_1, n_2, \dots, n_m\} : \mathrm{T}' \notin \mathbf{Thrd}^{c^i}_{exe},\\
\forall i \in \{n_1, n_2, \dots, n_m\} : (\mathrm{T}' \in \mathbf{Thrd}^{c^i}_{exe} \wedge \textsc{own}(\mathbb{l}^{i''}\ lck') \neq \mathrm{T}'),\\
\mathrm{T}' \in \mathbf{Thrd}^{c^n}_{exe},\\
\forall i \in \{0,\dots,k-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{\tilde{c}^i}_{exe},\\
\mathrm{T}' \in \mathbf{Thrd}^{\tilde{c}^k}_{exe}, \text{ and}\\
\forall i \in \{0,\dots,k\} : (|\mathbf{Thrd}^{\tilde{c}^i}_{exe}| \not> 1 \vee\\
\qquad \{\mathrm{T} \in \mathbf{Thrd}^{\tilde{c}^i}_{exe} \mid \exists r \in \mathbf{Reg}_{\mathrm{T}} : \exists x \in \mathbf{Var_g} :\\
\qquad\qquad \textsc{stm}(\mathrm{T}, pc^{\tilde{c}^i}_{\mathrm{T}}) = [\texttt{load } r \texttt{ from } x]^{pc^{\tilde{c}^i}_{\mathrm{T}}}\} = \emptyset),
\end{gathered}$$

where for all $i \in \{0,\dots,n\}$, $\mathbf{Thrd}^{c^i}_{exe}$ is as defined in Table 4.3, for all $i \in \{0,\dots,k\}$, $\mathbf{Thrd}^{\tilde{c}^i}_{exe}$ is as defined in Table 5.6, and $\mathbf{Var_g}$ contains all $x \in \mathbf{Var}$ such that x can be written to by one thread and read from by another thread

(i.e., there might be a data dependency between the threads; note that $\mathbf{Var_g}$ can be derived using Algorithm 6.9).

First note that:

- Since $\forall i \in \{0,\ldots,n-1\} \setminus \{n_1,n_2,\ldots,n_m\} : \mathrm{T}' \notin \mathbf{Thrd}^{c^i}_{exe}$, $\forall i \in \{n_1,n_2,\ldots,n_m\} : (\mathrm{T}' \in \mathbf{Thrd}^{c^i}_{exe} \wedge \textsc{own}(\mathbb{l}''' \; lck') \neq \mathrm{T}')$ and $\textsc{stm}(\mathrm{T}', pc^0_{\mathrm{T}'}) = [\texttt{lock}\; lck']^{pc^0_{\mathrm{T}'}}$, it must be that $pc^n_{\mathrm{T}'} = pc^0_{\mathrm{T}'}$, $\mathbb{r}^n_{\mathrm{T}'} = \mathbb{r}^0_{\mathrm{T}'}$, $t^{a^n}_{\mathrm{T}'} = t^{a^0}_{\mathrm{T}'} + \textsc{time}(c^{n_1},\mathrm{T}') + \textsc{time}(c^{n_2},\mathrm{T}') + \ldots + \textsc{time}(c^{n_m},\mathrm{T}')$ and $\forall lck \in \mathbf{Lck} : (\textsc{own}(\mathbb{l}^0 \; lck) = \mathrm{T}' \Rightarrow \mathbb{l}^n \; lck = \mathbb{l}^0 \; lck)$ (c.f., Table 4.3).

- Since $\forall i \in \{0,\ldots,k-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{\tilde{c}^i}_{exe}$, it must be that $pc^{\tilde{c}^k}_{\mathrm{T}'} = pc^{\tilde{c}^0}_{\mathrm{T}'}$, $\tilde{\mathbb{r}}^k_{\mathrm{T}'} = \tilde{\mathbb{r}}^0_{\mathrm{T}'}$, $\tilde{t}^{a^k}_{\mathrm{T}'} = \tilde{t}^{a^0}_{\mathrm{T}'}$ and $\forall lck \in \mathbf{Lck} : (\tilde{\textsc{own}}(\tilde{\mathbb{l}}^0 \; lck) = \mathrm{T}' \Rightarrow (\tilde{\mathbb{l}}^k \; lck = \tilde{\mathbb{l}}^0 \; lck \wedge \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^k \; lck))) = -\infty))$.

- Since $pc^n_{\mathrm{T}'} = pc^0_{\mathrm{T}'}$, $\mathbb{r}^n_{\mathrm{T}'} = \mathbb{r}^0_{\mathrm{T}'}$, $\forall lck \in \mathbf{Lck} : (\textsc{own}(\mathbb{l}^0 \; lck) = \mathrm{T}' \Rightarrow \mathbb{l}^n \; lck = \mathbb{l}^0 \; lck)$, $pc^{\tilde{c}^k}_{\mathrm{T}'} = pc^{\tilde{c}^0}_{\mathrm{T}'}$, $\tilde{\mathbb{r}}^k_{\mathrm{T}'} = \tilde{\mathbb{r}}^0_{\mathrm{T}'}$, $\forall lck \in \mathbf{Lck} : (\tilde{\textsc{own}}(\tilde{\mathbb{l}}^0 \; lck) = \mathrm{T}' \Rightarrow (\tilde{\mathbb{l}}^k \; lck = \tilde{\mathbb{l}}^0 \; lck \wedge \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^k \; lck))) = -\infty))$, $pc^0_{\mathrm{T}'} = pc^{\tilde{c}^0}_{\mathrm{T}'}$, $\mathbb{r}^0_{\mathrm{T}'} \in \gamma_{reg}(\tilde{\mathbb{r}}^0_{\mathrm{T}'})$, $t^{a^0}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a^0}_{\mathrm{T}'})$ and $\forall lck \in \mathbf{Lck} : (\textsc{own}(\mathbb{l}^0 \; lck) = \mathrm{T}' \Rightarrow (\textsc{stt}(\mathbb{l}^0 \; lck) = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^0 \; lck) \wedge \textsc{own}(\mathbb{l}^0 \; lck) = \tilde{\textsc{own}}(\tilde{\mathbb{l}}^0 \; lck) \wedge \textsc{dl}(\mathbb{l}^0 \; lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0 \; lck)) \wedge \textsc{pown}(\mathbb{l}^0 \; lck) = \tilde{\textsc{pown}}(\tilde{\mathbb{l}}^0 \; lck) \wedge \textsc{rel}(\mathbb{l}^0 \; lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^0 \; lck)) \wedge \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0 \; lck))) = -\infty))$, it must be that:

  $pc^n_{\mathrm{T}'} = pc^{\tilde{c}^k}_{\mathrm{T}'}$,
  $\mathbb{r}^n_{\mathrm{T}'} \in \gamma_{reg}(\tilde{\mathbb{r}}^k_{\mathrm{T}'})$ and
  $\forall lck \in \mathbf{Lck} : (\textsc{own}(\mathbb{l}^n \; lck) = \mathrm{T}' \Rightarrow (\textsc{stt}(\mathbb{l}^n \; lck) = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^k \; lck) \wedge$
  $\textsc{own}(\mathbb{l}^n \; lck) = \tilde{\textsc{own}}(\tilde{\mathbb{l}}^k \; lck) \wedge$
  $\textsc{dl}(\mathbb{l}^n \; lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^k \; lck)) \wedge$
  $\textsc{pown}(\mathbb{l}^n \; lck) = \tilde{\textsc{pown}}(\tilde{\mathbb{l}}^k \; lck) \wedge$
  $\textsc{rel}(\mathbb{l}^n \; lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^k \; lck)) \wedge$
  $\min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^k \; lck))) = -\infty))$

- Since $\exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}^0) : \forall x \in \mathbf{Var} : ((\mathbb{x}^0 \; x) \; \mathrm{T}') \subseteq ((\mathbb{x}' \; x) \; \mathrm{T}')$, $c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^{n_1} \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^{n_2} \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^{n_m} \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^n$, $\tilde{c}^0 \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}^k$, $\forall i \in \{0,\ldots,n-1\} \setminus \{n_1,n_2,\ldots,n_m\} : \mathrm{T}' \notin \mathbf{Thrd}^{c^i}_{exe}$, $\forall i \in \{n_1,n_2,\ldots,n_m\} : \textsc{own}(\mathbb{l}''' \; lck') \neq \mathrm{T}'$, $\forall i \in \{0,\ldots,k-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{\tilde{c}^i}_{exe}$, $\textsc{stm}(\mathrm{T}', pc^0_{\mathrm{T}'}) = [\texttt{lock}\; lck']^{pc^0_{\mathrm{T}'}}$ and TRIM is safe (Lemma 5.27), it must be that $\exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}^k) : \forall x \in \mathbf{Var} : ((\mathbb{x}^n \; x) \; \mathrm{T}') \subseteq ((\mathbb{x}' \; x) \; \mathrm{T}')$.

- Since $\forall i \in \{0,\ldots,k\} : (|\mathbf{Thrd}^{\tilde{c}^i}_{exe}| \not> 1 \vee \{\mathrm{T} \in \mathbf{Thrd}^{\tilde{c}^i}_{exe} \mid \exists r \in \mathbf{Reg}_{\mathrm{T}} : \exists x \in \mathbf{Var_g} : \textsc{stm}(\mathrm{T}, pc^{\tilde{c}^i}_{\mathrm{T}}) = [\texttt{load}\ r\ \texttt{from}\ x]^{pc^{\tilde{c}^i}_{\mathrm{T}}}\} = \emptyset)$, it must be that $\forall i \in \{0,\ldots,k\} : (\{\mathrm{T} \in \mathbf{Thrd}^{\tilde{c}^i}_{exe} \mid \exists r \in \mathbf{Reg}_{\mathrm{T}} : \exists x \in \mathbf{Var_g} : \textsc{stm}(\mathrm{T}, pc^{\tilde{c}^i}_{\mathrm{T}}) = [\texttt{load}\ r\ \texttt{from}\ x]^{pc^{\tilde{c}^i}_{\mathrm{T}}}\} \neq \emptyset \Rightarrow |\mathbf{Thrd}^{\tilde{c}^i}_{exe}| = 1)$. This means that if some thread in $\mathbf{Thrd}^{\tilde{c}^i}_{exe}$, where $i \in \{0,\ldots,k\}$, performs a `load`-statement, there is only one single thread in $\mathbf{Thrd}^{\tilde{c}^i}_{exe}$; thus that thread performs the `load`-statement. It is then easy to see, from the definition of $\mathbf{Thrd}^{\tilde{c}^i}_{exe}$, that there cannot occur any other write than those represented by $\tilde{\mathbb{x}}^i$ such that it could affect the `load`-statement of the thread in $\mathbf{Thrd}^{\tilde{c}^i}_{exe}$ (c.f., Assumption 5.50) – thus, it must be that $\tilde{\mathbb{x}}^k$ (and also all $\tilde{\mathbb{x}}^i$, where $i \in \{0,\ldots,k\}$) contains safe write history (c.f., Definition 5.18).

- Since, trivially, $\forall lck \in \mathbf{Lck} : \{\mathrm{T} \in \mathbf{Thrd}^{c^n}_{exe} \cap \mathbf{Thrd}_{\tilde{c}^k} \mid \textsc{stm}(\mathrm{T}, pc^n_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc^n_{\mathrm{T}}}\} \subseteq \{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}^k} \mid \exists l \in \mathbf{Lbl}_{\mathrm{T}} : \textsc{stm}(\mathrm{T}, l) = [\texttt{lock}\ lck]^l\}$, it must be that if T' can be assigned a lock in the concrete case, it can also be assigned the lock in the corresponding abstract case.

- Since $\xrightarrow[prg]{\sim}$ over-approximates the lock-owner assignment possible for $\xrightarrow[prg]{}$ and $\mathrm{T}' \in \mathbf{Thrd}_{\tilde{c}^k}$, it must be that $\textsc{own}(\tilde{\mathbb{l}}^0\ lck') = \mathrm{T}'$ is possible even if $\textsc{own}(\mathbb{l}^0\ lck') = \bot_{thrd}$, given that some other thread (i.e., not T') does `lock` lck' before T' in the abstract case and that the abstract transition sequence safely represents the concrete transition sequence; the situation is possible since $\mathbf{Ti\tilde{m}e} = \mathbf{Intv}$ (c.f., Lemma 5.55). However, if $\textsc{stm}(\mathrm{T}', pc^0_{\mathrm{T}'}) = [\texttt{lock}\ lck']^{pc^0_{\mathrm{T}'}}$, $\textsc{own}(\mathbb{l}^0\ lck') = \bot_{thrd}$ and $\textsc{own}(\tilde{\mathbb{l}}^0\ lck') = \mathrm{T}'$, it must be that $\textsc{st}(\tilde{\mathbb{l}}^0\ lck') = \mathit{unlocked}$, $t^{a^n}_{\mathrm{T}'} + \textsc{time}(c^n, \mathrm{T}') \in \gamma_t(\textsc{dl}(\tilde{\mathbb{l}}^0\ lck'))$ and $\min(\gamma_t(\textsc{dl}(\tilde{\mathbb{l}}^0\ lck'))) = -\infty$ (Table 5.6, Algorithm 5.11 and Lemmas 5.53 and 5.55).

- Since $\textsc{stm}(\mathrm{T}', pc^{\tilde{c}^k}_{\mathrm{T}'}) = [\texttt{lock}\ lck']^{pc^{\tilde{c}^k}_{\mathrm{T}'}}$ and $\mathrm{T}' \in \mathbf{Thrd}^{\tilde{c}^k}_{exe}$, it must be that $\textsc{own}(\tilde{\mathbb{l}}^{k\prime\prime}\ lck') = \mathrm{T}'$.

The proof will be conducted using induction based on T' having to wait for j, where $j \geq 0$, threads to first (acquire and) release lck' before it can successfully acquire lck'.

First consider the base case. Therefore, assume that T' is the first thread in a set of competing threads to successfully acquire lck'; i.e., $j = 0$. Then it must be that $\{n_1, n_2, \ldots, n_m\} = \emptyset$, and thus, $\forall i \in \{0,\ldots,n-1\} : \mathrm{T}' \notin \mathbf{Thrd}^{c^i}_{exe}$. (Note

that c^0 can be chosen to be the first configuration satisfying $\text{OWN}(\mathbb{l}^0\ lck') = \bot_{thrd} \wedge \text{STM}(\mathrm{T}', pc^0_{\mathrm{T}'}) = [\texttt{lock}\ lck']^{pc^0_{\mathrm{T}'}}$ and the rest of the assumptions of the lemma.) It must also be that the case $\text{OWN}(\mathbb{l}^n\ lck') = \bot_{thrd}$, $\text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}^k\ lck') \in \{\bot_{thrd}, \mathrm{T}'\}$ and $\text{OWN}(\mathbb{l}^{n''}\ lck') = \text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}^{k''}\ lck') = \mathrm{T}'$ must be considered.

Since T' is the first thread to acquire lck' it must be that $\mathbb{l}^n\ lck' = \mathbb{l}^0\ lck'$ and $\tilde{\mathbb{l}}^k\ lck' = \tilde{\mathbb{l}}^0\ lck'$. Thus, since $\text{OWN}(\mathbb{l}^0\ lck') = \bot_{thrd}$ and $\text{STM}(\mathrm{T}', pc^0_{\mathrm{T}'}) = [\texttt{lock}\ lck']^{pc^0_{\mathrm{T}'}}$, it must be that $\text{OWN}(\mathbb{l}^0\ lck') = \bot_{thrd} \Rightarrow (\text{STM}(\mathrm{T}', pc^0_{\mathrm{T}'}) = [\texttt{lock}\ lck']^{pc^0_{\mathrm{T}'}} \Rightarrow (\mathbb{l}^n\ lck' = \mathbb{l}^0\ lck' \wedge \tilde{\mathbb{l}}^k\ lck' = \tilde{\mathbb{l}}^0\ lck'))$. But, then all the assumptions of Lemma 5.56 are fulfilled, and thus, it must be that:

$$
\begin{array}{l}
\forall c @ \langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle \in \mathbf{Conf}: \\
\quad (c^n \xrightarrow[prg]{} c \Rightarrow \exists \tilde{c} @ \langle [\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^k}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \tilde{\mathbf{Conf}}: \\
\qquad (\tilde{c}^k \xrightarrow[prg]{} \tilde{c} \wedge \\
\qquad\quad pc_{\mathrm{T}'} = pc^{\tilde{c}}_{\mathrm{T}'} \wedge \\
\qquad\quad \mathbb{r}_{\mathrm{T}'} \in \gamma_{reg}(\tilde{\mathbb{r}}_{\mathrm{T}'}) \wedge \\
\qquad\quad t^a_{\mathrm{T}'} \in \gamma_t(\tilde{t}^a_{\mathrm{T}'}) \wedge \\
\qquad\quad \exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}) : (\forall x \in \mathbf{Var} : ((\mathbb{x}\ x)\ \mathrm{T}') \subseteq ((\mathbb{x}'\ x)\ \mathrm{T}')) \wedge \\
\qquad\quad \forall lck \in \mathbf{Lck} : (\text{OWN}(\mathbb{l}\ lck) = \mathrm{T}' \Rightarrow \\
\qquad\qquad\qquad (\text{STT}(\mathbb{l}\ lck) = \text{S}\tilde{\text{T}}\text{T}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\qquad\qquad\ \text{OWN}(\mathbb{l}\ lck) = \text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\qquad\qquad\ \text{DL}(\mathbb{l}\ lck) \in \gamma_t(\tilde{\text{DL}}(\tilde{\mathbb{l}}\ lck)) \wedge \\
\qquad\qquad\qquad\ \text{POWN}(\mathbb{l}\ lck) = \text{PO}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck) \wedge \\
\qquad\qquad\qquad\ \text{REL}(\mathbb{l}\ lck) \in \gamma_t(\tilde{\text{REL}}(\tilde{\mathbb{l}}\ lck)) \wedge \\
\qquad\qquad\qquad\ \min(\gamma_t(\tilde{\text{DL}}(\tilde{\mathbb{l}}\ lck))) = -\infty))))
\end{array}
$$

This concludes the proof of the base case.

Now consider the case that T' must wait for j owner switches (i.e., `lock`:s and `unlock`:s) on lck' before it can acquire lck' itself; i.e., T' is owner number $j+1$ among a set of competing threads to successfully acquire lck' (note that a thread could successfully acquire and release lck' several times while T' is waiting to acquire lck'; each time then counts as an owner switch). The induction assumption is that the lemma holds for all j owners that acquire lck' while T' is waiting (i.e., frozen in the abstract case) and for all cases involving other locks.

Assume that T' must wait for j owner switches on lck' before it successfully acquires lck' itself and that the lemma holds for all j owners that acquire lck' while T' is waiting. Then it must be that $\{n_1, n_2, \ldots, n_m\} \neq \emptyset$, and thus $t^{a^0}_{\mathrm{T}'} \leq t^{a^n}_{\mathrm{T}'} = t^{a^0}_{\mathrm{T}'} + \text{TIME}(c^{n_1}, \mathrm{T}') + \text{TIME}(c^{n_2}, \mathrm{T}') + \ldots + \text{TIME}(c^{n_m}, \mathrm{T}') =$

$t_{\mathrm{T}'}^{a^{n_m}} + \textsc{time}(c^{n_m}, \mathrm{T}')$ (c.f., Assumption 4.1 and Table 4.3).

Since the lemma holds for all j owners that acquire lck' while T' is waiting, and all other cases involving other locks, and $\xrightarrow[prg]{\sim}$ safely over-approximates the transitions described by $\xrightarrow[prg]{}$ for all other cases (Lemma 5.56), including lock owner assignments (Lemma 5.55), it must be that there exists an abstract transition trace (starting at $\tilde{c}^0$ and ending at $\tilde{c}^k$) that safely represents the concrete trace from c^0 to c^n for all j owners of lck', at least until the point in which they release lck' and do not acquire it again (which is the important part of the trace to consider here), the order in which threads acquire lck' and all states, including the accumulated execution times (c.f., Lemmas 5.51 and 5.56 and the induction assumption). Thus, since $\mathrm{T}' \in \mathbf{Thrd}_{exe}^{c^{n_m}} \wedge \textsc{own}(\mathbb{l}^{n_m\prime\prime}\ lck') \neq \mathrm{T}'$, $\forall i \in \{n_m+1,\ldots,n-1\} : \mathrm{T}' \notin \mathbf{Thrd}_{exe}^{c^i}$ and $\mathrm{T}' \in \mathbf{Thrd}_{exe}^{c^n} \wedge \textsc{own}(\mathbb{l}^n\ lck') = \bot_{thrd} \wedge \textsc{own}(\mathbb{l}^{n\prime\prime}\ lck') = \mathrm{T}'$ (since it is assumed that T' acquires lck' in the transition from c^n), it must be that lck' is released (by owner number j) in a transition to $c^{n'}$, where $c^{n_m} \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^{n'} \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^n$ and $n_m < n' \leq n$. Thus, it must be that $t_{\mathrm{T}'}^{a^{n_m}} + \textsc{time}(c^{n_m}, \mathrm{T}') = t_{\mathrm{T}'}^{a^{n'}} \leq \textsc{rel}(\mathbb{l}^n\ lck') \leq t_{\mathrm{T}'}^{a^n} + \textsc{time}(c^n, \mathrm{T}')$ (c.f., Assumption 4.1 and Table 4.3), where $\textsc{rel}(\mathbb{l}^n\ lck') = \textsc{rel}(\mathbb{l}^{n'}\ lck')$ and $\textsc{rel}(\mathbb{l}^n\ lck') \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^k\ lck'))$ (given the abstract trace from $\tilde{c}^0$ to $\tilde{c}^k$ that safely represents the trace from c^0 to $c^{n'}$ for the previous, i.e., j^{th}, owner of lck', it is easy to see that this is the result when the j^{th} owner issues `unlock` lck'). But, then it is trivially the case that $t_{\mathrm{T}'}^{a^n} + \textsc{time}(c^n, \mathrm{T}') \in \gamma_t(\textsc{dllock}(\tilde{c}^k, lck'))$ (Lemma 5.53).

To show that $\textsc{acctime}$ is safe for this case, first note that $\mathrm{T}' \in \mathbf{Thrd}_{exe}^{\tilde{c}^k}$, $\textsc{stm}(\mathrm{T}', pc_{\mathrm{T}'}^{\tilde{c}^k}) = [\texttt{lock}\ lck']^{pc_{\mathrm{T}'}^{\tilde{c}^k}}$ and $\tilde{\textsc{st}}(\tilde{\mathbb{l}}^{k\prime\prime}\ lck') = \mathit{unlocked}$. Also note that since $t_{\mathrm{T}'}^{a^n} + \textsc{time}(c^n, \mathrm{T}') \in \gamma_t(\textsc{dllock}(\tilde{c}^k, lck')) = \tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{k\prime\prime}\ lck')$, $\textsc{dl}(\mathbb{l}^{n\prime\prime}\ lck') = t_{\mathrm{T}'}^{a^n} + \textsc{time}(c^n, \mathrm{T}')$ (c.f., Tables 4.2 and 4.3 since T' acquires lck' in a transition from c^n) and $\min(\gamma_t(\tilde{t}_{\mathrm{T}'}^{a^0} \mathbin{\tilde{+}_t} \textsc{absTime}(\tilde{c}^0, \mathrm{T}'))) \leq t_{\mathrm{T}'}^{a^n} + \textsc{time}(c^n, \mathrm{T}') \leq \max(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{k\prime\prime}\ lck')))$ (c.f., Assumptions 4.1 and 5.50), it must be the case that $\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{k\prime\prime}\ lck') \not\tilde{<}_t\ \tilde{t}_{\mathrm{T}'}^{a^0} \mathbin{\tilde{+}_t} \textsc{absTime}(\tilde{c}^0, \mathrm{T}')$, which means that there are three branches of Algorithm 5.12 that must be considered here. Note that this also means that $\textsc{dl}(\mathbb{l}^{n\prime\prime}\ lck') \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{k\prime\prime}\ lck'))$. For the sake of readability, let $\tilde{c}^{k\prime\prime} = \langle [\mathrm{T}, pc_{\mathrm{T}}^{\tilde{c}^k}, \tilde{\mathbb{r}}_{\mathrm{T}}^k, \tilde{t}_{\mathrm{T}}^{a^k}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}^k}}, \tilde{\mathbb{X}}^k, \tilde{\mathbb{l}}^{k\prime\prime} \rangle$. Also let $\tilde{t}_{\mathrm{T}'}^{a\prime}$ be defined as $\tilde{t}_{\mathrm{T}}^{a\prime}$ in Algorithm 5.12.

1. Since T' has been frozen while waiting to acquire lck', it can be the case that $\tilde{t}_{\mathrm{T}'}^{a^k} \mathbin{\tilde{+}_t} \textsc{absTime}(\tilde{c}^{k\prime\prime}, \mathrm{T}') \tilde{<}_t \tilde{\textsc{rel}}(\tilde{\mathbb{l}}^{k\prime\prime}\ lck')$, where $\tilde{t}_{\mathrm{T}'}^{a^k} = \tilde{t}_{\mathrm{T}'}^{a^0}$. (Note

that this does not necessarily have to be the case, though.) Let $\tilde{c}'$ be any configuration derived before (i.e., $\tilde{c}' = \tilde{c}^{k\prime\prime}$) or inside the **while**-loop.

First note that it cannot be that $\textsc{absTime}(\tilde{c}', \mathrm{T}') = \alpha_t(\{0\})$ and $\tilde{t}^{a\prime}_{\mathrm{T}'} \dot{+}_t \textsc{absTime}(\tilde{c}', \mathrm{T}') \tilde{<}_t \textsc{rel}(\tilde{\mathbb{I}}^{k\prime\prime}\ lck')$ (c.f., Assumptions 4.3 and 5.50). This means that the **while**-loop will eventually terminate. It does so when $\tilde{t}^{a\prime}_{\mathrm{T}'}$ is the last point in time that safely represents the situation that T' has not yet acquired lck'; thus, at $\tilde{t}^{a\prime}_{\mathrm{T}'} \dot{+}_t \textsc{absTime}(\tilde{c}', \mathrm{T}')$, T' might have acquired lck' (i.e., $\tilde{t}^{a\prime}_{\mathrm{T}'} \tilde{<}_t \textsc{rel}(\tilde{\mathbb{I}}^{k\prime\prime}\ lck')$ and $\tilde{t}^{a\prime}_{\mathrm{T}'} \dot{+}_t \textsc{absTime}(\tilde{c}', \mathrm{T}') \not\tilde{<}_t \textsc{rel}(\tilde{\mathbb{I}}^{k\prime\prime}\ lck')$). In later references within this proof, the $\tilde{t}^{a\prime}_{\mathrm{T}'}$ obtained at the exit of the **while**-loop will be referred to as $\tilde{t}^{a^w}_{\mathrm{T}'}$.

Since $\tilde{t}^{a^w}_{\mathrm{T}'} \tilde{<}_t \textsc{rel}(\tilde{\mathbb{I}}^{k\prime\prime}\ lck')$ and $\tilde{t}^{a^w}_{\mathrm{T}'} \dot{+}_t \textsc{absTime}(\tilde{c}', \mathrm{T}') \not\tilde{<}_t \textsc{rel}(\tilde{\mathbb{I}}^{k\prime\prime}\ lck')$, it is easy to see that this branch will lead to an auxiliary configuration, $\tilde{c}^{k\prime}$, such that $\tilde{c}^k \xrightarrow[prg]{\sim} \tilde{c}^{k\prime}$, for which $\tilde{\mathbb{I}}^{k\prime}\ lck' = \tilde{\mathbb{I}}^{k\prime\prime}\ lck'$; i.e., T' has not yet acquired lck'. The only difference for T' between $\tilde{c}^k$ and $\tilde{c}^{k\prime}$ is that, in the latter, it has an advanced accumulated execution time (c.f., Table 5.5). Since $\tilde{t}^{a^w}_{\mathrm{T}'} \dot{+}_t \textsc{absTime}(\tilde{c}', \mathrm{T}') \not\tilde{<}_t \textsc{rel}(\tilde{\mathbb{I}}^{k\prime\prime}\ lck')$, it is also easy to see that this branch of Algorithm 5.12 will not be taken when $\textsc{accTime}$ is called based on $\tilde{c}^{k\prime}$. Note that it must be that $|\{n_1, n_2, \ldots, n_m\}|$ is greater than or equal to the number of iterations of the **while**-loop (Assumption 5.50 and Lemma 5.52). Thus, it is also easy to see that $\textsc{dl}(\tilde{\mathbb{I}}^{k\prime\prime}\ lck') \not\tilde{<}_t \tilde{t}^{a^w}_{\mathrm{T}'} \dot{+}_t \textsc{absTime}(\tilde{c}^{k\prime}, \mathrm{T}')$ since $\textsc{time}(c^i, \mathrm{T}') \in \gamma_t(\textsc{absTime}(\tilde{c}^{k\prime}, \mathrm{T}'))$, where $i \in \{n_1, n_2, \ldots, n_m\}$ is the corresponding concrete configuration for which the **while**-loop terminates (Assumption 5.50). This means that for $\tilde{c}^{k\prime}$, one of the two last branches (considered in the next two bullets) of Algorithm 5.12 will apply.

2. First note that it must be that $\textsc{own}(\tilde{\mathbb{I}}^{k\prime\prime}\ lck') \neq \mathrm{T}'$ since T' has been waiting for at least one other thread to release lck' before it is allowed to acquire it (c.f., Tables 5.5 and 5.6 and the induction assumption). If, on the other hand, $\textsc{rel}(\tilde{\mathbb{I}}^{k\prime\prime}\ lck') \tilde{<}_t \tilde{t}^{a\prime}_{\mathrm{T}'} \dot{+}_t \textsc{absTime}(\tilde{c}^{k\prime\prime}, \mathrm{T}')$, then the proof is equivalent to the corresponding part of the proof for Lemma 5.54 since $t^{a^n}_{\mathrm{T}'} + \textsc{time}(c^n, \mathrm{T}') = \textsc{dl}(\mathbb{I}^{n\prime\prime}\ lck')$, $\textsc{dl}(\mathbb{I}^{n\prime\prime}\ lck') \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{I}}^{k\prime\prime}\ lck'))$ and $\textsc{rel}(\mathbb{I}^{n\prime\prime}\ lck') \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{I}}^{k\prime\prime}\ lck'))$. Note that this also applies if $\tilde{t}^{a\prime}_{\mathrm{T}'} = \tilde{t}^{a^w}_{\mathrm{T}'}$ (i.e., if $\tilde{c}^{k\prime\prime} = \tilde{c}^{k\prime}$) since it must be that $t^{a^n}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a^w}_{\mathrm{T}'})$, which follows from Assumption 5.50 and Lemma 5.51 based on 1 above.

3. If $(\tilde{t}^{a\prime}_{\mathrm{T}'} \dot{+}_t \textsc{absTime}(\tilde{c}', \mathrm{T}')) \tilde{\sqcap}_t \textsc{rel}(\tilde{\mathbb{I}}^{k\prime\prime}\ lck') \neq \tilde{\bot}_t$, then let $\tilde{t}^{a\prime\prime}_{\mathrm{T}'} = \tilde{t}^{a\prime}_{\mathrm{T}'} \dot{+}_t$

$\textsc{absTime}(\tilde{c}', \mathrm{T})$ (where $\tilde{t}^{a''}_{\mathrm{T}'}$ is thus defined as in Algorithm 5.12 and $\tilde{t}^{a'}_{\mathrm{T}'}$ is either $\tilde{t}^{a^k}_{\mathrm{T}'}$ or $\tilde{t}^{a^w}_{\mathrm{T}'}$ and $\tilde{c}'$ is either $\tilde{c}^{k''}$ or $\tilde{c}^{k'}$), which is obviously a safe estimation of the first point in time when T' can acquire lck'.

Now, let $\tilde{c}'$ be any configuration derived before (i.e., $\tilde{c}' = \tilde{c}^{k''}$ or $\tilde{c}' = \tilde{c}^{k'}$) or inside the **repeat**-loop (and the corresponding for $\tilde{t}^{a'}_{\mathrm{T}'}$), which will now be considered. Note that $\tilde{t}^{r}_{\mathrm{T}'} = \tilde{\top}_t$ is used to exit the loop in case $\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{k''}\ lck') \tilde{<}_t \tilde{t}^{a'}_{\mathrm{T}'} \tilde{+}_t \textsc{absTime}(\tilde{c}', \mathrm{T}')$ or $0 \in \gamma_t(\textsc{absTime}(\tilde{c}', \mathrm{T}'))$, where the latter case means that a $\tilde{t}^{a'}_{\mathrm{T}'}$ such that $\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^{k''}\ lck') \tilde{<}_t \tilde{t}^{a'}_{\mathrm{T}'}$ cannot be derived (c.f., Assumption 5.50).

(a) If $\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{k''}\ lck') \tilde{<}_t \tilde{t}^{a'}_{\mathrm{T}'} \tilde{+}_t \textsc{absTime}(\tilde{c}', \mathrm{T}')$, then it must be that $\tilde{t}^{a'}_{\mathrm{T}'}$ is a safe estimation of the last point in time when T' can acquire lck' since $t^{a^n}_{\mathrm{T}'} + \textsc{time}(c^n, \mathrm{T}') = \textsc{dl}(\mathbb{l}^{n''}\ lck')$, $\textsc{dl}(\mathbb{l}^{n''}\ lck') \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{k''}\ lck'))$ and T' acquires lck' in a transition from c^n (c.f., Assumption 5.50 and Lemma 5.52 which means that the total number of iterations of the **repeat**-loop, and possibly the **while**-loop from 1, must be greater than or equal to $|\{n_1, n_2, \ldots, n_m, n\}|$). Thus, it must be that $t^{a^n}_{\mathrm{T}'} + \textsc{time}(c^n, \mathrm{T}') \in \gamma_t((\tilde{t}^{a''}_{\mathrm{T}'} \tilde{\sqcup}_t \tilde{t}^{a'}_{\mathrm{T}'}) \tilde{\sqcap}_t \tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{k''}\ lck') \tilde{\sqcap}_t (\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^{k''}\ lck') \tilde{\sqcup}_t \alpha_t(\{\infty\})))$ since $\textsc{rel}(\mathbb{l}^{n''}\ lck') \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^{k''}\ lck'))$ and $\min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{k''}\ lck'))) = -\infty$.

(b) If $0 \in \gamma_t(\textsc{absTime}(\tilde{c}', \mathrm{T}'))$, then it must obviously be that $\tilde{t} \tilde{+}_t \textsc{absTime}(\tilde{c}'', \mathrm{T}')$, where $\tilde{t} = (\tilde{t}^{a'}_{\mathrm{T}'} \tilde{\sqcup}_t \alpha_t(\{\infty\})) \tilde{\sqcap}_t \tilde{\textsc{rel}}(\tilde{\mathbb{l}}^{k''}\ lck')$, and $\tilde{c}'' = \langle [\mathrm{T}, pc^{\tilde{c}^k}_{\mathrm{T}}, \tilde{\mathbb{r}}^k_{\mathrm{T}}, (\mathrm{T} = \mathrm{T}' \ ?\ \tilde{t} : \tilde{t}^{a^k}_{\mathrm{T}})]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}^k}}, \tilde{\mathbb{x}}^k, \tilde{\mathbb{l}}^{k''} \rangle$, is a safe approximation of the last point in time when T' can (or rather, will) acquire lck' (c.f., Assumption 5.50 and Lemma 5.52). Thus, it must be that $t^{a^n}_{\mathrm{T}'} + \textsc{time}(c^n, \mathrm{T}') \in \gamma_t((\tilde{t}^{a''}_{\mathrm{T}'} \tilde{\sqcup}_t \tilde{t}^{a'}_{\mathrm{T}'}) \tilde{\sqcap}_t \tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{k''}\ lck') \tilde{\sqcap}_t (\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^{k''}\ lck') \tilde{\sqcup}_t \alpha_t(\{\infty\})))$ since $\textsc{rel}(\mathbb{l}^{n''}\ lck') \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^{k''}\ lck'))$, $\textsc{dl}(\mathbb{l}^{n''}\ lck') \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{k''}\ lck'))$, $t^{a^n}_{\mathrm{T}'} + \textsc{time}(c^n, \mathrm{T}') = \textsc{dl}(\mathbb{l}^{n''}\ lck')$, $\min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{k''}\ lck'))) = -\infty$ and T' acquires lck' in a transition from c^n.

(c) If $0 \notin \gamma_t(\textsc{absTime}(\tilde{c}', \mathrm{T}'))$ and also $\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{k''}\ lck') \not\tilde{<}_t \tilde{t}^{a'}_{\mathrm{T}'} \tilde{+}_t \textsc{absTime}(\tilde{c}', \mathrm{T}')$, then it must be that, at the end of some iteration of the **repeat**-loop, $\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^{k''}\ lck') \tilde{<}_t \tilde{t}^{a'}_{\mathrm{T}'}$. For such a $\tilde{t}^{a'}_{\mathrm{T}'}$, it is easy to see that $t^{a^n}_{\mathrm{T}'} + \textsc{time}(c^n, \mathrm{T}') \in \gamma_t((\tilde{t}^{a''}_{\mathrm{T}'} \tilde{\sqcup}_t \tilde{t}^{a'}_{\mathrm{T}'}) \tilde{\sqcap}_t \tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{k''}\ lck') \tilde{\sqcap}_t (\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^{k''}\ lck') \tilde{\sqcup}_t \alpha_t(\{\infty\})))$ since $\textsc{rel}(\mathbb{l}^{n''}\ lck') \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^{k''}\ lck'))$, $\textsc{dl}(\mathbb{l}^{n''}\ lck') \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{k''}\ lck'))$, $t^{a^n}_{\mathrm{T}'} +$

$\textsc{time}(c^n,\mathrm{T}') = \textsc{dl}(\mathbb{l}^{n\prime\prime}\ lck')$, $\min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{k\prime\prime}\ lck'))) = -\infty$ and T' acquires lck' in a transition from c^n (c.f., Assumption 5.50 and Lemma 5.52 which means that the total number of iterations of the **repeat**-loop, and possibly the **while**-loop from 1, must be greater than or equal to $|\{n_1,n_2,\ldots,n_m,n\}|$).

Thus, it has been shown that $t^{a^n}_{\mathrm{T}'} + \textsc{time}(c^n,\mathrm{T}') \in \gamma_t(\textsc{accTime}(\tilde{c}'@\langle[\mathrm{T},pc'_{\mathrm{T}},\tilde{\mathbb{r}}'_{\mathrm{T}},\tilde{t}^{a\prime}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^k}},\tilde{\mathbb{x}}',\tilde{\mathbb{l}}'\rangle,\mathbf{Thrd}^{\tilde{c}'}_{exe},\mathrm{T}'))$, for both the case that $\tilde{c}'$ is $\tilde{c}^{k\prime\prime}$ (if $\tilde{t}^{a^k}_{\mathrm{T}'} \tilde{+}_t \textsc{absTime}(\tilde{c}^{k\prime\prime},\mathrm{T}') \not\tilde{<}_t \tilde{\textsc{rel}}(\tilde{\mathbb{l}}^{k\prime\prime}\ lck')$) and $\tilde{c}'$ is $\tilde{c}^{k\prime}$ (if $\tilde{t}^{a^k}_{\mathrm{T}'} \tilde{+}_t \textsc{absTime}(\tilde{c}^{k\prime\prime},\mathrm{T}') \tilde{<}_t \tilde{\textsc{rel}}(\tilde{\mathbb{l}}^{k\prime\prime}\ lck')$), where $\tilde{c}^{k\prime\prime}$ and $\tilde{c}^{k\prime}$ are as defined above. If $\tilde{t}^{a^k}_{\mathrm{T}'} \tilde{+}_t \textsc{absTime}(\tilde{c}^{k\prime\prime},\mathrm{T}') \tilde{<}_t \tilde{\textsc{rel}}(\tilde{\mathbb{l}}^{k\prime\prime}\ lck')$, it is easy to see that

$$
\begin{array}{l}
pc^n_{\mathrm{T}'} = pc^{\tilde{c}^{k\prime}}_{\mathrm{T}'},\\
\mathbb{r}^n_{\mathrm{T}'} \in \gamma_{reg}(\tilde{\mathbb{r}}^{k\prime}_{\mathrm{T}'}),\\
\exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}^{k\prime}) : (\forall x \in \mathbf{Var} : ((\mathbb{x}^n\ x)\ \mathrm{T}') \subseteq ((\mathbb{x}'\ x)\ \mathrm{T}')) \text{ and}\\
\forall lck \in \mathbf{Lck} : (\textsc{own}(\mathbb{l}^n\ lck) = \mathrm{T}' \Rightarrow (\textsc{stt}(\mathbb{l}^n\ lck) = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^{k\prime}\ lck) \wedge\\
\qquad\qquad \textsc{own}(\mathbb{l}^n\ lck) = \tilde{\textsc{own}}(\tilde{\mathbb{l}}^{k\prime}\ lck) \wedge\\
\qquad\qquad \textsc{dl}(\mathbb{l}^n\ lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{k\prime}\ lck)) \wedge\\
\qquad\qquad \textsc{powN}(\mathbb{l}^n\ lck) = \tilde{\textsc{powN}}(\tilde{\mathbb{l}}^{k\prime}\ lck) \wedge\\
\qquad\qquad \textsc{rel}(\mathbb{l}^n\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^{k\prime}\ lck)) \wedge\\
\qquad\qquad \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^{k\prime}\ lck))) = -\infty))
\end{array}
$$

since, for T', the accumulated execution time is the only state affected by the transition $\tilde{c}^k \xrightarrow[prg]{\sim} \tilde{c}^{k\prime}$ (c.f., Table 5.5; this means that, e.g., $\tilde{\mathbb{l}}^{k\prime}\ lck' = \tilde{\mathbb{l}}^{k\prime\prime}\ lck'$) and TRIM is safe (Lemma 5.27). Thus, for both transition sequences described by $\tilde{c}^k \xrightarrow[prg]{\sim} \tilde{c}^{k\prime} \xrightarrow[prg]{\sim} \tilde{c}$ and $\tilde{c}^k \xrightarrow[prg]{\sim} \tilde{c}$, where $\tilde{c}@\langle[\mathrm{T},pc^{\tilde{c}}_{\mathrm{T}},\tilde{\mathbb{r}}_{\mathrm{T}},\tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}^k}},\tilde{\mathbb{x}},\tilde{\mathbb{l}}\rangle \in \tilde{\mathbf{Conf}}$, for the two different cases ($\tilde{t}^{a^k}_{\mathrm{T}'} \tilde{+}_t \textsc{absTime}(\tilde{c}^{k\prime\prime},\mathrm{T}') \tilde{<}_t \tilde{\textsc{rel}}(\tilde{\mathbb{l}}^{k\prime\prime}\ lck')$ and $\tilde{t}^{a^k}_{\mathrm{T}'} \tilde{+}_t$

$\textsc{absTime}(\tilde{c}^{k\prime\prime},\mathrm{T}') \not\preceq_t \tilde{\textsc{rel}}(\tilde{\mathbb{l}}^{k\prime\prime}\ lck')$, respectively), it must be that

$$
\begin{array}{l}
pc_{\mathrm{T}'} = pc^{\tilde{c}}_{\mathrm{T}'},\\
\mathbb{r}_{\mathrm{T}'} \in \gamma_{reg}(\tilde{\mathbb{r}}_{\mathrm{T}'}),\\
t^{a}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a}_{\mathrm{T}'}),\\
\exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}) : (\forall x \in \mathbf{Var} : ((\mathbb{x}\ x)\ \mathrm{T}') \subseteq ((\mathbb{x}'\ x)\ \mathrm{T}')) \text{ and}\\
\forall lck \in \mathbf{Lck} : (\textsc{own}(\mathbb{l}\ lck) = \mathrm{T}' \Rightarrow (\textsc{stt}(\mathbb{l}\ lck) = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}\ lck) \wedge\\
\qquad\qquad \textsc{own}(\mathbb{l}\ lck) = \tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck) \wedge\\
\qquad\qquad \textsc{dl}(\mathbb{l}\ lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}\ lck)) \wedge\\
\qquad\qquad \textsc{pown}(\mathbb{l}\ lck) = \tilde{\textsc{pown}}(\tilde{\mathbb{l}}\ lck) \wedge\\
\qquad\qquad \textsc{rel}(\mathbb{l}\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}\ lck)) \wedge\\
\qquad\qquad \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}\ lck))) = -\infty))
\end{array}
$$

where $c^n \xrightarrow[prg]{} c$ for some $c@\langle[\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}, \mathbb{l}\rangle \in \mathbf{Conf}$, since $\xrightarrow[ax]{\sim}$ is a safe approximation of $\xrightarrow[ax]{}$ (Lemma 5.49), $\textsc{dl}(\mathbb{l}\ lck') = t^{a^n}_{\mathrm{T}'} + \textsc{time}(c^n,\mathrm{T}')$ (Table 4.2) and TRIM is safe (Lemma 5.27). But then the lemma holds. ∎

Lemma 5.58 shows that $\xrightarrow[prg]{\sim}$ can be used to safely approximate any finite concrete transition sequence. Note that the approximation is safe if either no thread issues a `load`-statement on a global variable or that the thread issuing the `load`-statement is the sole thread in $\mathbf{Thrd}_{exe}$ in any step of the transition sequence.

Lemma 5.58 (Soundness of $\xrightarrow[prg]{\sim}$, final state):
Given the valid concrete configurations (c.f., Definition 4.4) $c^0@\langle[\mathrm{T}, pc^0_{\mathrm{T}}, \mathbb{r}^0_{\mathrm{T}}, t^{a^0}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \mathbb{x}^0, \mathbb{l}^0\rangle \in \mathbf{Conf}$ *and* $c^n@\langle[\mathrm{T}, pc^n_{\mathrm{T}}, \mathbb{r}^n_{\mathrm{T}}, t^{a^n}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \mathbb{x}^n, \mathbb{l}^n\rangle \in \mathbf{Conf}$ *and the abstract configuration* $\tilde{c}^0@\langle[\mathrm{T}, pc^{\tilde{c}^0}_{\mathrm{T}}, \tilde{\mathbb{r}}^0_{\mathrm{T}}, \tilde{t}^{a^0}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}^0, \tilde{\mathbb{l}}^0\rangle \in \tilde{\mathbf{Conf}}$, *such that* $c^0 \in \gamma_{conf}(\tilde{c}^0)$, $\forall lck \in \mathbf{Lck} : \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck))) =$

$-\infty, 0 < n$ *and* $c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^n$, $\xrightarrow[prg]{\sim}$ *satisfies*

$$
\begin{array}{l}
(\forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}^n) = [\texttt{halt}]^{pc_{\mathrm{T}}^n}) \\
\Rightarrow \\
(\exists \tilde{c}^k @ \langle [\mathrm{T}, pc_{\mathrm{T}}^{\tilde{c}^k}, \tilde{\mathbb{r}}_{\mathrm{T}}^k, \tilde{t}_{\mathrm{T}}^{a^k}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}^k, \tilde{\mathbb{l}}^k \rangle \in \tilde{\mathbf{Conf}} : \\
\quad (\tilde{c}^0 \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}^k \wedge \\
\quad\quad (\forall i \in \{0, \ldots, k-1\} : (|\mathbf{Thrd}_{exe}^{\tilde{c}^i}| \not> 1 \vee \\
\quad\quad\quad\quad \{\mathrm{T} \in \mathbf{Thrd}_{exe}^{\tilde{c}^i} \mid \exists r \in \mathbf{Reg}_{\mathrm{T}} : \exists x \in \mathbf{Var_g} : \\
\quad\quad\quad\quad\quad \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}^{\tilde{c}^i}) = [\texttt{load } r \texttt{ from } x]^{pc_{\mathrm{T}}^{\tilde{c}^i}}\} = \emptyset)) \\
\quad\quad \Rightarrow \\
\quad\quad (\forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : (pc_{\mathrm{T}}^n = pc_{\mathrm{T}}^{\tilde{c}^k} \wedge \\
\quad\quad\quad\quad \mathbb{r}_{\mathrm{T}}^n \in \gamma_{reg}(\tilde{\mathbb{r}}_{\mathrm{T}}^k) \wedge \\
\quad\quad\quad\quad t_{\mathrm{T}}^{a^n} \in \gamma_t(\tilde{t}_{\mathrm{T}}^{a^k}) \wedge \\
\quad\quad\quad\quad \exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}^k) : (\forall x \in \mathbf{Var} : ((\mathbb{x}^n\ x)\ \mathrm{T}) \subseteq ((\mathbb{x}'\ x)\ \mathrm{T}))) \wedge \\
\quad\quad\quad\quad \forall lck \in \mathbf{Lck} : (\textsc{stt}(\mathbb{l}^n\ lck) = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^k\ lck) \wedge \\
\quad\quad\quad\quad\quad\quad \textsc{own}(\mathbb{l}^n\ lck) = \tilde{\textsc{own}}(\tilde{\mathbb{l}}^k\ lck) \wedge \\
\quad\quad\quad\quad\quad\quad \textsc{dl}(\mathbb{l}^n\ lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^k\ lck)) \wedge \\
\quad\quad\quad\quad\quad\quad \textsc{pown}(\mathbb{l}^n\ lck) = \tilde{\textsc{pown}}(\tilde{\mathbb{l}}^k\ lck) \wedge \\
\quad\quad\quad\quad\quad\quad \textsc{rel}(\mathbb{l}^n\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^k\ lck)) \wedge \\
\quad\quad\quad\quad\quad\quad \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^k\ lck))) = -\infty))))
\end{array}
$$

where, for all $i \in \{0, \ldots, k-1\}$, $\mathbf{Thrd}_{exe}^{\tilde{c}^i}$ *is as defined in Table 5.6, and* $\mathbf{Var_g}$ *contains all* $x \in \mathbf{Var}$ *such that* x *can be written to by one thread and read from by another thread (i.e., there might be a data dependency between the threads; note that* $\mathbf{Var_g}$ *can be derived using Algorithm 6.9).* □

PROOF. Assume that the valid (c.f., Definition 4.4) concrete configurations $c^0 @ \langle [\mathrm{T}, pc_{\mathrm{T}}^0, \mathbb{r}_{\mathrm{T}}^0, t_{\mathrm{T}}^{a^0}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \mathbb{x}^0, \mathbb{l}^0 \rangle \in \mathbf{Conf}$ and $c^n @ \langle [\mathrm{T}, pc_{\mathrm{T}}^n, \mathbb{r}_{\mathrm{T}}^n, t_{\mathrm{T}}^{a^n}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \mathbb{x}^n, \mathbb{l}^n \rangle \in \mathbf{Conf}$ and the abstract configuration $\tilde{c}^0 @ \langle [\mathrm{T}, pc_{\mathrm{T}}^{\tilde{c}^0}, \tilde{\mathbb{r}}_{\mathrm{T}}^0, \tilde{t}_{\mathrm{T}}^{a^0}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}^0, \tilde{\mathbb{l}}^0 \rangle \in \tilde{\mathbf{Conf}}$ are such that $c^0 \in \gamma_{conf}(\tilde{c}^0)$, $c^0 \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^n$, $\forall lck \in \mathbf{Lck}$: $\min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck))) = -\infty$ and $\forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}^n) = [\texttt{halt}]^{pc_{\mathrm{T}}^n}$.

Note that since $\forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}^n) = [\texttt{halt}]^{pc_{\mathrm{T}}^n}$, it must be that all threads trying to acquire a lock at some point will eventually successfully do so (i.e., there are no deadlocks etc.) and there are no infinite loops. Also note that the possible abstract combinations of the owner and the state for some lock, $lck \in \mathbf{Lck}$, given a reference thread, $\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}$, in a lock state, $\tilde{\mathbb{l}}$, resulting from a transition using $\xrightarrow[prg]{\sim}$ are by definition as follows.

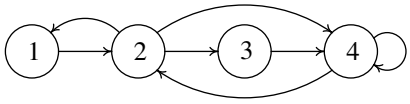


Figure 5.7: Abstract lock state transitions.

1. $\text{OW}\tilde{\text{N}}(\tilde{\mathbb{l}}\ lck) \notin \{\bot_{thrd}, \text{T}\}$ – This means that T will be frozen if it issues `lock` *lck* and occurs when $\text{OWN}(\mathbb{l}\ lck) \neq \text{T}$.

2. $\text{OW}\tilde{\text{N}}(\tilde{\mathbb{l}}\ lck) = \bot_{thrd}$ – This occurs when $\text{OWN}(\mathbb{l}\ lck) = \bot_{thrd}$. A safe (over-approximate) owner assignment will occur if T issues `lock` *lck*. (The soundness is given by that it is trivially the case that for all concrete and abstract configurations consisting of the threads in $\mathbf{Thrd}_{\tilde{c}}$, $\{\text{T}' \in \mathbf{Thrd}_{exe} \mid \text{STM}(\text{T}', pc_{\text{T}'}) = [\texttt{lock}\ lck]^{pc_{\text{T}'}}\} \subseteq \{\text{T}' \in \mathbf{Thrd}_{\tilde{c}} \mid \exists l \in \mathbf{Lbl}_{\text{T}'} : \text{STM}(\text{T}', l) = [\texttt{lock}\ lck]^{l}\}$; c.f., Table 4.3.)

3. $\text{OW}\tilde{\text{N}}(\tilde{\mathbb{l}}\ lck) = \text{T} \wedge \text{S}\tilde{\text{T}}\text{T}(\tilde{\mathbb{l}}\ lck) = unlocked$ – This means that T has not yet done `lock` *lck*, but some other thread has (with the result that T was assigned *lck*; c.f., the discussion for state 2). If T issues `lock` *lck* within the deadline, it will successfully acquire *lck*. If it does not, there is no corresponding concrete situation described by the owner assignment, given that $\text{DL}(\mathbb{l}\ lck) \in \tilde{\text{DL}}(\tilde{\mathbb{l}}\ lck)$, and thus, the configuration will be discontinued; c.f., Algorithms 6.1 and 6.6, which are discussed in Chapter 6. This occurs when $\text{OWN}(\mathbb{l}\ lck) = \bot_{thrd}$.

4. $\text{OW}\tilde{\text{N}}(\tilde{\mathbb{l}}\ lck) = \text{T} \wedge \text{S}\tilde{\text{T}}\text{T}(\tilde{\mathbb{l}}\ lck) = locked$ – This occurs when $\text{OWN}(\mathbb{l}\ lck) = \text{T}$.

The possible transitions between these abstract states (as defined by $\xrightarrow[ax]{\sim}$ and $\xrightarrow[prg]{\sim}$) are depicted in Figure 5.7. State 3 (a result from the over-approximate owner assignment performed by $\xrightarrow[prg]{\sim}$) is needed because $\tilde{\mathbf{Time}} = \mathbf{Intv}$, which means that even if a thread acquires a lock first in the abstract case, it could be that some other thread could actually acquire the lock first in the corresponding concrete case. Lemma 5.55 gives that $\xrightarrow[prg]{\sim}$ covers all the possible concrete situations for lock owner assignments, regardless of which thread issues `lock` *lck* first in the abstract case; c.f., a transition from state 2 to state 4, possibly via state 3.

This proof will partly be conducted using induction on how the states of a configuration are changed during transitions, based on one thread at

a time. Therefore, consider $c^f @ \langle [\mathrm{T}, pc^f_{\mathrm{T}}, \mathbb{r}^f_{\mathrm{T}}, t^{a^f}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \mathbb{x}^f, \mathbb{l}^f \rangle \in \mathbf{Conf}$, $c^g @ \langle [\mathrm{T}, pc^g_{\mathrm{T}}, \mathbb{r}^g_{\mathrm{T}}, t^{a^g}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \mathbb{x}^g, \mathbb{l}^g \rangle \in \mathbf{Conf}$, $\tilde{c}^i @ \langle [\mathrm{T}, pc^{\tilde{c}^i}_{\mathrm{T}}, \tilde{\mathbb{r}}^i_{\mathrm{T}}, \tilde{t}^{a^i}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}^i, \tilde{\mathbb{l}}^i \rangle \in \mathbf{C\tilde{o}nf}$ and $\mathrm{T}' \in \mathbf{Thrd}_{\tilde{c}}$ such that

$$
\begin{array}{l}
c^f \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c^g \wedge \\
0 \leq f < g \leq n \wedge \\
\forall h \in \{f, \ldots, g-2\} : (\mathrm{T}' \notin \mathbf{Thrd}^{c^h}_{exe} \vee \\
\qquad \exists lck \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}', pc^h_{\mathrm{T}'}) = [\texttt{lock}\ lck]^{pc^h_{\mathrm{T}'}} \wedge \\
\qquad\qquad \textsc{own}(\mathbb{l}^{h+1}\ lck) \neq \mathrm{T}')) \wedge \\
\mathrm{T}' \in \mathbf{Thrd}^{c^{g-1}}_{exe} \wedge \\
\forall lck \in \mathbf{Lck} : \textsc{stm}(\mathrm{T}', pc^{g-1}_{\mathrm{T}'}) = [\texttt{lock}\ lck]^{pc^{g-1}_{\mathrm{T}'}} \Rightarrow \textsc{own}(\mathbb{l}^g\ lck) = \mathrm{T}' \wedge \\
pc^f_{\mathrm{T}'} = pc^{\tilde{c}^i}_{\mathrm{T}'} \wedge \\
\mathbb{r}^f_{\mathrm{T}'} \in \gamma_{reg}(\tilde{\mathbb{r}}^i_{\mathrm{T}'}) \wedge \\
t^{a^f}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a^i}_{\mathrm{T}'}) \wedge \\
\exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}^i) : \forall x \in \mathbf{Var} : \forall \mathrm{T} \in \mathbf{Thrd} : ((\mathbb{x}^f\ x)\ \mathrm{T}) \subseteq ((\mathbb{x}'\ x)\ \mathrm{T}) \wedge \\
\forall lck \in \mathbf{Lck} : ((\textsc{own}(\mathbb{l}^f\ lck) \neq \bot_{thrd} \Rightarrow (\textsc{stt}(\mathbb{l}^f\ lck) = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^i\ lck) \wedge \\
\qquad\qquad \textsc{own}(\mathbb{l}^f\ lck) = \tilde{\textsc{own}}(\tilde{\mathbb{l}}^i\ lck) \wedge \\
\qquad\qquad \textsc{dl}(\mathbb{l}^f\ lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^i\ lck)) \wedge \\
\qquad\qquad \textsc{powned}(\mathbb{l}^f\ lck) = \tilde{\textsc{powned}}(\tilde{\mathbb{l}}^i\ lck) \wedge \\
\qquad\qquad \textsc{rel}(\mathbb{l}^f\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^i\ lck)) \wedge \\
\qquad\qquad \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^i\ lck))) = -\infty)) \wedge \\
\qquad (\textsc{own}(\mathbb{l}^f\ lck) = \bot_{thrd} \Rightarrow ((\textsc{own}(\mathbb{l}^f\ lck) = \tilde{\textsc{own}}(\tilde{\mathbb{l}}^i\ lck) \vee \\
\qquad\qquad (\tilde{\textsc{own}}(\tilde{\mathbb{l}}^i\ lck) = \mathrm{T}' \wedge \\
\qquad\qquad\ \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^i\ lck) = \mathit{unlocked} \wedge \\
\qquad\qquad\ t^{a^g}_{\mathrm{T}'} \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^i\ lck)) \wedge \\
\qquad\qquad\ \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^i\ lck))) = -\infty)) \wedge \\
\qquad\qquad \textsc{powned}(\mathbb{l}^f\ lck) = \tilde{\textsc{powned}}(\tilde{\mathbb{l}}^i\ lck) \wedge \\
\qquad\qquad \textsc{rel}(\mathbb{l}^f\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^i\ lck))))) \wedge \\
\forall h \in \{f, \ldots, g-1\} : (|\mathbf{Thrd}^{\tilde{c}^h}_{exe}| \not> 1 \vee \\
\qquad \{\mathrm{T} \in \mathbf{Thrd}^{\tilde{c}^h}_{exe} \mid \exists r \in \mathbf{Reg}_{\mathrm{T}} : \exists x \in \mathbf{Var_g} : \\
\qquad\qquad \textsc{stm}(\mathrm{T}, pc^{\tilde{c}^h}_{\mathrm{T}}) = [\texttt{load}\ r\ \texttt{from}\ x]^{pc^{\tilde{c}^h}_{\mathrm{T}}} \} = \emptyset)
\end{array}
$$

where $\mathbf{Thrd}_{exe}$ is as defined in Table 4.3. This is the induction assumption. Then it is easy to see that there exists a $\tilde{c}^j @ \langle [\mathrm{T}, pc^{\tilde{c}^j}_{\mathrm{T}}, \tilde{\mathbb{r}}^j_{\mathrm{T}}, \tilde{t}^{a^j}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}^j, \tilde{\mathbb{l}}^j \rangle \in$

Coñf, such that

$$
\begin{array}{l}
\tilde{c}^i \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}^j \wedge \\
pc^g_{\mathrm{T}'} = pc^{\tilde{c}^j}_{\mathrm{T}'} \wedge \\
\mathbb{r}^g_{\mathrm{T}'} \in \gamma_{reg}(\tilde{\mathbb{r}}^j_{\mathrm{T}'}) \wedge \\
t^{a^g}_{\mathrm{T}'} \in \gamma_t(\tilde{t}^{a^j}_{\mathrm{T}'}) \wedge \\
\exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}^j) : (\forall x \in \mathbf{Var} : ((\mathbb{x}^g\ x)\ \mathrm{T}') \subseteq ((\mathbb{x}'\ x)\ \mathrm{T}')) \wedge \\
\forall lck \in \mathbf{Lck} : ((\textsc{own}(\mathbb{l}^f\ lck) = \mathrm{T}' \vee \\
\qquad \textsc{own}(\mathbb{l}^g\ lck) = \mathrm{T}') \Rightarrow (\textsc{stt}(\mathbb{l}^g\ lck) = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^j\ lck) \wedge \\
\qquad\qquad \textsc{own}(\mathbb{l}^g\ lck) = \tilde{\textsc{own}}(\tilde{\mathbb{l}}^j\ lck) \wedge \\
\qquad\qquad \textsc{dl}(\mathbb{l}^g\ lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^j\ lck)) \wedge \\
\qquad\qquad \textsc{pown}(\mathbb{l}^g\ lck) = \tilde{\textsc{pown}}(\tilde{\mathbb{l}}^j\ lck) \wedge \\
\qquad\qquad \textsc{rel}(\mathbb{l}^g\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^j\ lck)) \wedge \\
\qquad\qquad \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^j\ lck))) = -\infty))
\end{array}
$$

(Lemmas 5.55, 5.56 and 5.57). Note that even if for some lock, $lck \in \mathbf{Lck}$, T' issues `lock` lck but lck is assigned to some other thread, T' will eventually be assigned lck so that it can acquire it (since all threads that want to acquire a lock eventually will be able to do so). For such cases, T' is the owner of lck in c^g and $\tilde{c}^j$ (c.f., Lemmas 5.56 and 5.57).

Now consider the base case for the induction part of the proof. Since $c^0 \in \gamma_{conf}(\tilde{c}^0)$, $\forall lck \in \mathbf{Lck} : \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck))) = -\infty$ and c^0 is valid, it is easy to see that

$$
\begin{array}{l}
\forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : (pc^0_{\mathrm{T}} = pc^{\tilde{c}^0}_{\mathrm{T}} \wedge \\
\qquad \mathbb{r}^0_{\mathrm{T}} \in \gamma_{reg}(\tilde{\mathbb{r}}^0_{\mathrm{T}}) \wedge \\
\qquad t^{a^0}_{\mathrm{T}} \in \gamma_t(\tilde{t}^{a^0}_{\mathrm{T}}) \wedge \\
\qquad \exists \mathbb{x}' \in \gamma_{var}(\tilde{\mathbb{x}}^0) : \forall x \in \mathbf{Var} : ((\mathbb{x}^0\ x)\ \mathrm{T}) \subseteq ((\mathbb{x}'\ x)\ \mathrm{T})) \wedge \\
\forall lck \in \mathbf{Lck} : (\textsc{stt}(\mathbb{l}^0\ lck) = \tilde{\textsc{stt}}(\tilde{\mathbb{l}}^0\ lck) \wedge \\
\qquad \textsc{own}(\mathbb{l}^0\ lck) = \tilde{\textsc{own}}(\tilde{\mathbb{l}}^0\ lck) \wedge \\
\qquad \textsc{dl}(\mathbb{l}^0\ lck) \in \gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck)) \wedge \\
\qquad \textsc{pown}(\mathbb{l}^0\ lck) = \tilde{\textsc{pown}}(\tilde{\mathbb{l}}^0\ lck) \wedge \\
\qquad \textsc{rel}(\mathbb{l}^0\ lck) \in \gamma_t(\tilde{\textsc{rel}}(\tilde{\mathbb{l}}^0\ lck)) \wedge \\
\qquad \min(\gamma_t(\tilde{\textsc{dl}}(\tilde{\mathbb{l}}^0\ lck))) = -\infty)
\end{array}
$$

which means that as long as $\forall h \in \{0,\ldots,k-1\} : (|\mathbf{Thrd}^{\tilde{c}^h}_{exe}| \not> 1 \vee \{\mathrm{T} \in \mathbf{Thrd}^{\tilde{c}^h}_{exe} \mid \exists r \in \mathbf{Reg}_{\mathrm{T}} : \exists x \in \mathbf{Var_g} : \textsc{stm}(\mathrm{T}, pc^{\tilde{c}^h}_{\mathrm{T}}) = [\texttt{load}\ r\ \texttt{from}\ x]^{pc^{\tilde{c}^h}_{\mathrm{T}}}\} = \emptyset)$, the induction holds for all threads in $\mathbf{Thrd}_{\tilde{c}}$.

Note that by definition, $\mathbb{l}^n\ lck = \mathbb{l}^0\ lck$ (c.f., Tables 4.2 and 4.3) and $\tilde{\mathbb{l}}^k\ lck = \tilde{\mathbb{l}}^0\ lck$ (c.f., Tables 5.5 and 5.6) if *lck* is never acquired by any thread, or never released by its initially owning thread (i.e., the owner of *lck* in c^0 and $\tilde{c}^0$, respectively).

This concludes the proof. ■

Because of the unsafe nature of $\xrightarrow[prg]{\sim}$ (i.e., it cannot safely approximate all concrete transition sequences), it cannot be directly used to derive a safe set of possible final configurations (i.e., configurations such that all threads are issuing `halt`). It must instead be encapsulated by an algorithm that uses it in a safe manner and handles the unsafe situations explicitly. Such an algorithm is defined in the next chapter.

Chapter 6

Safe Timing Analysis by Abstract Execution

In this chapter, an algorithm for deriving safe timing bounds of PPL programs will be defined. The analysis will be based on the abstraction of the PPL semantics presented in Chapter 5.

NOTE. *A summary of the notation and nomenclature used in this thesis can be found in Appendix A.*

6.1 Abstract Execution

ABSEXE, as given by Algorithm 6.1, can be used to derive safe approximations of the concrete final states resulting from a finite transition sequence (Lemma 6.8). ABSEXE will also safely approximate some infinite transition sequences but is not guaranteed to terminate for all possible inputs. The algorithm is a work-list algorithm and is defined based on the auxiliary functions CHOOSE, which returns an arbitrary element of a given set (which must be non-empty), ISFINAL, ISDEADLOCK, ISTIMEOUT, ISVALID, CYCLE, EXELOADTHRD, GLOBALVAR, EXETHRD, GETVARLOAD and GETREGLOAD, defined in Algorithms 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 6.10, 6.11 and 6.12,

Algorithm 6.1 Abstract Execution

function ABSEXE($\tilde{C}, \tilde{t}_{to}$)
  $\tilde{C}^w \leftarrow \tilde{C}$; $\tilde{C}^f \leftarrow \emptyset$; $\tilde{C}^d \leftarrow \emptyset$; $\tilde{C}^t \leftarrow \emptyset$
  while $\tilde{C}^w \neq \emptyset$ **do**
    $\tilde{c}@\langle [\mathrm{T}, pc_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{I}}\rangle \leftarrow$ CHOOSE($\tilde{C}^w$)
    $\tilde{C}^w \leftarrow \tilde{C}^w \setminus \{\tilde{c}\}$
    if ISFINAL($\tilde{c}$) **then**
      $\tilde{C}^f \leftarrow \tilde{C}^f \cup \{\tilde{c}\}$
    else if ISDEADLOCK($\tilde{c}$) **then**
      $\tilde{C}^d \leftarrow \tilde{C}^d \cup \{\tilde{c}\}$
    else if ISTIMEOUT($\tilde{c}, \tilde{t}_{to}$) **then**
      $\tilde{C}^t \leftarrow \tilde{C}^t \cup \{\tilde{c}\}$
    else if ISVALID($\tilde{c}, \tilde{t}_{to}$) **then**
      $\mathbf{Thrd}^{\mathtt{load}}_{exe} \leftarrow$ EXELOADTHRD($\tilde{c}$)
      if $\mathbf{Thrd}^{\mathtt{load}}_{exe} \neq \emptyset \wedge |\text{EXETHRD}(\tilde{c})| > 1$ **then**
        $\langle [\tilde{t}^{a\prime}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}^{\mathtt{load}}_{exe}} \rangle \leftarrow \langle [\tilde{t}^a_\mathrm{T} \mathbin{\tilde{+}}_t \text{ABSTIME}(\tilde{c}, \mathrm{T})]_{\mathrm{T}\in\mathbf{Thrd}^{\mathtt{load}}_{exe}} \rangle$
        for all $\mathrm{T} \in \mathbf{Thrd}^{\mathtt{load}}_{exe}$ **do**
          $x \leftarrow$ GETVARLOAD(STM($\mathrm{T}, pc_\mathrm{T}$))
          $r \leftarrow$ GETREGLOAD(STM($\mathrm{T}, pc_\mathrm{T}$))
          $\tilde{c}' \leftarrow \langle [\mathrm{T}', pc_{\mathrm{T}'}, \tilde{\mathbb{r}}_{\mathrm{T}'}, \tilde{t}^a_{\mathrm{T}'}]_{\mathrm{T}'\in\mathbf{Thrd}_{\tilde{c}}\setminus\{\mathrm{T}\}}, \tilde{\mathbb{x}}, \tilde{\mathbb{I}}\rangle$
          $(\tilde{C}^f_\mathrm{T}, \tilde{C}^d_\mathrm{T}, \tilde{C}^t_\mathrm{T}) \leftarrow$ ABSEXE($\{\tilde{c}'\}, \tilde{t}^{a\prime}_\mathrm{T} \mathbin{\tilde{\sqcap}}_t (\tilde{t}_{to} \mathbin{\tilde{\sqcup}}_t \alpha_t(\{-\infty\}))$)
          $\tilde{\mathbb{r}}_\mathrm{T} \leftarrow \tilde{\mathbb{r}}_\mathrm{T}[r \mapsto \tilde{\bot}_{val}]$
          for all $\langle \tilde{\mathbf{T}}, \tilde{\mathbb{x}}', \tilde{\mathbb{I}}' \rangle \in \tilde{C}^f_\mathrm{T} \cup \tilde{C}^d_\mathrm{T} \cup \tilde{C}^t_\mathrm{T} \cup \{\tilde{c}\}$ **do**
            $\tilde{\mathbb{r}}_\mathrm{T} \leftarrow \tilde{\mathbb{r}}_\mathrm{T}[r \mapsto (\tilde{\mathbb{r}}_\mathrm{T}\ r) \mathbin{\tilde{\sqcup}}_{val} \text{READ}(\tilde{\mathbb{x}}', x, \mathrm{T}, \tilde{t}^{a\prime}_\mathrm{T})]$
          end for
        end for
        $\langle [pc_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}^{\mathtt{load}}_{exe}} \rangle \leftarrow \langle [pc_\mathrm{T} + 1]_{\mathrm{T}\in\mathbf{Thrd}^{\mathtt{load}}_{exe}} \rangle$
        $\langle [\tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}^{\mathtt{load}}_{exe}} \rangle \leftarrow \langle [\tilde{t}^{a\prime}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}^{\mathtt{load}}_{exe}} \rangle$
        $\tilde{C}^w \leftarrow \tilde{C}^w \cup \{\langle [\mathrm{T}, pc_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{I}}\rangle\}$
      else
        $\tilde{C}^w \leftarrow \tilde{C}^w \cup \{\tilde{c}' \in \tilde{\mathbf{Conf}} \mid \tilde{c} \xrightarrow[prg]{\sim} \tilde{c}'\}$
      end if
    end if
  end while
  return $(\tilde{C}^f, \tilde{C}^d, \tilde{C}^t)$
end function

respectively.

A thread executing a `load`-statement on some global variable (i.e., a variable that could transfer data between threads) is extracted and handled separately in case it would not be the sole thread executed on a transition. This is done by recursively using ABSEXE for each such thread to simulate how the rest of the threads in the configuration can affect the read value. When the effects have been derived, they are merged and put in the target register for the thread that issues the `load`-statement. Next, a new configuration, in which the `load`-statements have been performed, is added to the work-list.

If no `load`-statement on some global variable is issued in any thread, or a thread issuing such a `load`-statement is the sole thread that will execute on a transition, $\xrightarrow[prg]{\sim}$ is used to derive a set of succeeding configurations, which are then added to the work-list.

Algorithm 6.2 Choose an Element

```
1: function CHOOSE(S)
Require: S ≠ ∅
2:     return one of the elements in S
3: end function
```

Given an abstract configuration, $\tilde{c} \in \tilde{\mathbf{Conf}}$, ISFINAL$(\tilde{c})$ means that $\tilde{c}$ is in the final state; i.e., all threads issue `halt`-statements.

Algorithm 6.3 Final Abstract Configuration

```
1: function ISFINAL(⟨[T, pc_T, r̃_T, t̃^a_T]_{T∈Thrd_c̃}, x̃, l̃⟩)
2:     return ∀T ∈ Thrd_c̃ : STM(T, pc_T) = [halt]^{pc_T}
3: end function
```

Given an abstract configuration, $\tilde{c} \in \tilde{\mathbf{Conf}}$, ISDEADLOCK$(\tilde{c})$ means that $\tilde{c}$ cannot reach a final state (Lemma 6.5). Note that ISDEADLOCK is not guaranteed to identify all such cases, though.

Given an abstract configuration, $\tilde{c} \in \tilde{\mathbf{Conf}}$, and a timeout, $\tilde{t}_{to} \in \tilde{\mathbf{Time}}$, ISTIMEOUT$(\tilde{c}, \tilde{t}_{to})$ means that $\tilde{c}$ cannot reach a final state before $\tilde{t}_{to}$ has passed (Lemma 6.6). Note that ISTIMEOUT might not identify all possible such cases, though.

Given an abstract configuration, $\tilde{c} \in \tilde{\mathbf{Conf}}$, and a timeout, $\tilde{t}_{to} \in \tilde{\mathbf{Time}}$, $\neg$ISVALID$(\tilde{c}, \tilde{t}_{to})$ means that $\tilde{c}$ cannot reach a configuration that could represent at least one valid (c.f., Definition 4.4) concrete configuration (Lemma 6.7). Note that ISVALID might not identify all possible such cases, though.

Algorithm 6.4 Deadlocked Abstract Configuration

1: **function** ISDEADLOCK($\tilde{c}@\langle[\mathrm{T}, pc_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle$)

Require: $\neg$ISFINAL($\tilde{c}$)

2: $\mathbf{Thrd}_{\mathtt{lock}} \leftarrow \{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \mid \exists lck \in \mathbf{Lck} : (\mathrm{STM}(\mathrm{T}, pc_\mathrm{T}) = [\mathtt{lock}\ lck]^{pc_\mathrm{T}} \wedge \mathrm{O\tilde{W}N}(\tilde{\mathbb{l}}\ lck) \notin \{\bot_{thrd}, \mathrm{T}\} \wedge \mathrm{S\tilde{T}T}(\tilde{\mathbb{l}}\ lck) = locked)\}$

3: $E \leftarrow \{(\mathrm{T}, \mathrm{T}') \in \mathbf{Thrd}_{\mathtt{lock}} \times \mathbf{Thrd}_{\mathtt{lock}} \mid \exists lck \in \mathbf{Lck} : (\mathrm{STM}(\mathrm{T}, pc_\mathrm{T}) = [\mathtt{lock}\ lck]^{pc_\mathrm{T}} \wedge \mathrm{O\tilde{W}N}(\tilde{\mathbb{l}}\ lck) = \mathrm{T}')\}$

4: **return** $\mathbf{Thrd}_{\tilde{c}} = \mathbf{Thrd} \wedge (\mathrm{CYCLE}(\mathbf{Thrd}_{\mathtt{lock}}, E) \vee \exists \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : \exists lck \in \mathbf{Lck} : (\mathrm{STM}(\mathrm{T}, pc_\mathrm{T}) = [\mathtt{lock}\ lck]^{pc_\mathrm{T}} \wedge \mathrm{S\tilde{T}T}(\tilde{\mathbb{l}}\ lck) = locked \wedge \mathrm{O\tilde{W}N}(\tilde{\mathbb{l}}\ lck) \neq \bot_{thrd} \wedge \mathrm{STM}(\mathrm{O\tilde{W}N}(\tilde{\mathbb{l}}\ lck), pc_{\mathrm{O\tilde{W}N}(\tilde{\mathbb{l}}\ lck)}) = [\mathtt{halt}]^{pc_{\mathrm{O\tilde{W}N}(\tilde{\mathbb{l}}\ lck)}}))$

5: **end function**

Algorithm 6.5 Timed-Out Abstract Configuration

1: **function** ISTIMEOUT($\tilde{c}@\langle[\mathrm{T}, pc_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle, \tilde{t}_{to}$)

Require: $\neg$ISFINAL($\tilde{c}$) $\wedge$ $\neg$ISDEADLOCK($\tilde{c}$)

2: **return** $\forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : (\mathrm{STM}(\mathrm{T}, pc_\mathrm{T}) \neq [\mathtt{halt}]^{pc_\mathrm{T}} \Rightarrow \tilde{t}_{to} \lesssim_t (\tilde{t}^a_\mathrm{T} \dot{+}_t \mathrm{ABSTIME}(\tilde{c}, \mathrm{T})) \vee (\mathbf{Thrd}_{\tilde{c}} \subset \mathbf{Thrd} \wedge \exists lck \in \mathbf{Lck} : (\mathrm{STM}(\mathrm{T}, pc_\mathrm{T}) = [\mathtt{lock}\ lck]^{pc_\mathrm{T}} \wedge \mathrm{O\tilde{W}N}(\tilde{\mathbb{l}}\ lck) \notin \{\bot_{thrd}, \mathrm{T}\})))$

3: **end function**

Algorithm 6.6 Valid Abstract Configuration

1: **function** ISVALID($\tilde{c}@\langle[\mathrm{T}, pc_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle, \tilde{t}_{to}$)

Require: $\neg$ISFINAL($\tilde{c}$) $\wedge$ $\neg$ISDEADLOCK($\tilde{c}$) $\wedge$ $\neg$ISTIMEOUT($\tilde{c}, \tilde{t}_{to}$)

2: $\mathbf{Thrd}_{\mathtt{lock}} \leftarrow \{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \mid \exists lck \in \mathbf{Lck} : (\mathrm{STM}(\mathrm{T}, pc_\mathrm{T}) = [\mathtt{lock}\ lck]^{pc_\mathrm{T}} \wedge \mathrm{O\tilde{W}N}(\tilde{\mathbb{l}}\ lck) \notin \{\bot_{thrd}, \mathrm{T}\})\}$

3: $E \leftarrow \{(\mathrm{T}, \mathrm{T}') \in \mathbf{Thrd}_{\mathtt{lock}} \times \mathbf{Thrd}_{\mathtt{lock}} \mid \exists lck \in \mathbf{Lck} : (\mathrm{STM}(\mathrm{T}, pc_\mathrm{T}) = [\mathtt{lock}\ lck]^{pc_\mathrm{T}} \wedge \mathrm{O\tilde{W}N}(\tilde{\mathbb{l}}\ lck) = \mathrm{T}')\}$

4: **return** $(\mathbf{Thrd}_{\tilde{c}} = \mathbf{Thrd} \Rightarrow \neg\mathrm{CYCLE}(\mathbf{Thrd}_{\mathtt{lock}}, E)) \wedge \forall lck \in \mathbf{Lck} : \forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : ((\mathrm{O\tilde{W}N}(\tilde{\mathbb{l}}\ lck) = \mathrm{T} \wedge \mathrm{S\tilde{T}T}(\tilde{\mathbb{l}}\ lck) = unlocked) \Rightarrow (\mathrm{STM}(\mathrm{T}, pc_\mathrm{T}) \neq [\mathtt{halt}]^{pc_\mathrm{T}} \wedge \mathrm{D\tilde{L}}(\tilde{\mathbb{l}}\ lck) \not\gtrsim_t (\tilde{t}^a_\mathrm{T} \dot{+}_t \mathrm{ABSTIME}(\tilde{c}, \mathrm{T}))))$

5: **end function**

Given a graph, (V,E), CYCLE(V,E) means that (V,E) contains at least one cycle (Lemma 6.1).

Algorithm 6.7 Determine if Graph Has Cycles

```
1:  function CYCLE(V, E)
2:      V' ← V
3:      E' ← E
4:      while V' ≠ ∅ do
5:          V'' ← {v ∈ V' | ¬∃v' ∈ V' : (v', v) ∈ E'}
6:          if V'' = ∅ then
7:              return true
8:          else
9:              E' ← E' \ {(v, v') ∈ E' | v ∈ V'' ∧ v' ∈ V'}
10:             V' ← V' \ V''
11:         end if
12:     end while
13:     return false
14: end function
```

Given a configuration, $\tilde{c} \in \tilde{\mathbf{Conf}}$, EXELOADTHRD$(\tilde{c})$ is a set of threads that might issue a `load`-statement on a global variable in a transition from $\tilde{c}$ (Lemma 6.4).

Algorithm 6.8 Threads Executing a Possibly Unsafe Load Statement

```
1: function EXELOADTHRD(c̃@⟨[T, pc_T, r̃_T, t̃^a_T]_{T∈Thrd_c̃}, x̃, Ĩ⟩)
2:     return {T ∈ EXETHRD(c̃) | ∃r ∈ Reg_T : ∃x ∈ GLOBALVAR(Thrd_c̃) :
                                              STM(T, pc_T) = [load r from x]^{pc_T}}
3: end function
```

Given a set of threads, $\mathbf{Thrd}_{\tilde{c}} \subseteq \mathbf{Thrd}$, GLOBALVAR$(\mathbf{Thrd}_{\tilde{c}})$ is the set of variables that could transfer data between some of the threads in $\mathbf{Thrd}_{\tilde{c}}$ (Lemma 6.3).

Given a configuration, $\tilde{c} \in \tilde{\mathbf{Conf}}$, EXETHRD$(\tilde{c})$ is an over-approximation of $\mathbf{Thrd}_{exe}$ as defined in Table 5.6 (Lemma 6.2).

Given a `load`-statement, GETVARLOAD is the variable, and GETREGLOAD is the register, defined by the statement.

Lemma 6.1 (Soundness of CYCLE):
Given the graph (V,E), where V is a set of vertices and E is a set of edges (i.e.,

Algorithm 6.9 Global Variables in an Abstract Configuration

function GLOBALVAR($\mathbf{Thrd}_{\tilde{c}}$)
  $\langle [\mathbf{Var}_{\mathrm{T}}^{\mathtt{load}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}} \rangle \leftarrow$
    $\langle [\{x \in \mathbf{Var} \mid \exists r \in \mathbf{Reg}_{\mathrm{T}} : \exists l \in \mathbf{Lbl}_{\mathrm{T}} : \mathrm{STM}(\mathrm{T},l) = [\mathtt{load}\ r\ \mathtt{from}\ x]^l\}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}} \rangle$
  $\langle [\mathbf{Var}_{\mathrm{T}}^{\mathtt{store}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}} \rangle \leftarrow$
    $\langle [\{x \in \mathbf{Var} \mid \exists r \in \mathbf{Reg}_{\mathrm{T}} : \exists l \in \mathbf{Lbl}_{\mathrm{T}} : \mathrm{STM}(\mathrm{T},l) = [\mathtt{store}\ r\ \mathtt{to}\ x]^l\}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}} \rangle$
  return $\{x \in \mathbf{Var} \mid \exists \mathrm{T},\mathrm{T}' \in \mathbf{Thrd}_{\tilde{c}} : (\mathrm{T} \neq \mathrm{T}' \wedge x \in \mathbf{Var}_{\mathrm{T}}^{\mathtt{load}} \wedge x \in \mathbf{Var}_{\mathrm{T}'}^{\mathtt{store}})\}$
end function

Algorithm 6.10 Threads to Execute in Abstract Configuration

function EXETHRD($\tilde{c}@\langle [\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}_{\mathrm{T}}^{a}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{I}} \rangle$)
  $\mathbf{Thrd}_{\mathbf{hold}} \leftarrow \{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \mid \mathrm{STM}(\mathrm{T}, pc_{\mathrm{T}}) = [\mathtt{halt}]^{pc_{\mathrm{T}}} \vee \exists lck \in \mathbf{Lck} :$
    $(\mathrm{STM}(\mathrm{T}, pc_{\mathrm{T}}) = [\mathtt{lock}\ lck]^{pc_{\mathrm{T}}} \wedge \mathrm{O\tilde{W}N}(\tilde{\mathbb{I}}\ lck) \neq \mathrm{T})\}$
  $\langle [\tilde{t}_{\mathrm{T}}^{r}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}\setminus\mathbf{Thrd}_{\mathbf{hold}}} \rangle \leftarrow \langle [\mathrm{ABSTIME}(\tilde{c}, \mathrm{T})]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}\setminus\mathbf{Thrd}_{\mathbf{hold}}} \rangle$
  $t_{min} \leftarrow \min\{\min(\gamma_t(\tilde{t}_{\mathrm{T}}^{a} \tilde{+}_t \tilde{t}_{\mathrm{T}}^{r})) \mid \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \setminus \mathbf{Thrd}_{\mathbf{hold}}\}$
  $t_{max} \leftarrow \min\{\max(\gamma_t(\tilde{t}_{\mathrm{T}}^{a} \tilde{+}_t \tilde{t}_{\mathrm{T}}^{r})) \mid \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \setminus \mathbf{Thrd}_{\mathbf{hold}}\}$
  $\tilde{t} \leftarrow \alpha_t(\{t_{min}, t_{max}\})$
  return $\{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \setminus \mathbf{Thrd}_{\mathbf{hold}} \mid \tilde{t} \tilde{\sqcap}_t (\tilde{t}_{\mathrm{T}}^{a} \tilde{+}_t \tilde{t}_{\mathrm{T}}^{r}) \neq \tilde{\bot}_t\} \cup$
    $\{\mathrm{T} \in \mathbf{Thrd}_{\mathbf{hold}} \mid \exists lck \in \mathbf{Lck} :$
      $(\mathrm{STM}(\mathrm{T}, pc_{\mathrm{T}}) = [\mathtt{lock}\ lck]^{pc_{\mathrm{T}}} \wedge \mathrm{O\tilde{W}N}(\tilde{\mathbb{I}}\ lck) = \bot_{thrd})\}$
end function

Algorithm 6.11 Get Variable in Load Statement

function GETVARLOAD($[\mathtt{load}\ r\ \mathtt{from}\ x]^l$)
  return x
end function

Algorithm 6.12 Get Register in Load Statement

function GETREGLOAD($[\mathtt{load}\ r\ \mathtt{from}\ x]^l$)
  return r
end function

pairs of vertices) connecting the vertices, then CYCLE(V,E) *iff* (V,E) *contains at least one cycle.* □

PROOF. Assume that (V,E) is a graph, where V is a set of vertices and E is a set of edges connecting the vertices. By definition, a cycle involving vertices $v_1, v_2, v_3, \ldots, v_n$ is described by the edges $(v_1,v_2),(v_2,v_3),\ldots,(v_n,v_1)$, where $n \geq 2$. Thus it is easy to see that a vertex, $v \in V$, cannot be part of a cycle in (V,E) if $\neg\exists v' \in V : (v',v) \in E$; i.e., if v has no incoming edges. It is also easy to see that the graph (V',E'), where $V' = V \setminus \{v \in V \mid \neg\exists v' \in V : (v',v) \in E\}$ (i.e., all vertices without incoming edges are removed) and $E' = E \setminus \{(v,v') \in E \mid v \in \{v'' \in V \mid \neg\exists v''' \in V : (v''',v'') \in E\} \wedge v' \in V\}$, (i.e., all edges going out from a vertex without incoming edges are removed) contains exactly as many cycles as (V,E).

Thus it must be that if this procedure can be repeated until an empty graph is reached, there are no cycles in the initial graph. Likewise, if it is not possible to reduce the initial graph to an empty graph, there must be at least one cycle in the initial graph. If there is no cycle in the initial graph, it is easy to see that the graph can be reduced to the empty graph by the above procedure. Likewise it is easy to see that if there is a cycle in the initial graph, the graph cannot be reduced to the empty graph by the above procedure. But then it must be that CYCLE(V,E) iff the graph (V,E) contains at least one cycle. ■

Lemma 6.2 (Soundness of EXETHRD):
Given $\tilde{c} \in \tilde{\mathbf{Conf}}$, $\mathbf{Thrd}^{\tilde{c}}_{exe} \subseteq$ EXETHRD$(\tilde{c})$, *where* $\mathbf{Thrd}^{\tilde{c}}_{exe}$ *is as defined in Table 5.6.* □

PROOF. Based on $\tilde{c}@\langle[\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathtt{r}}_{\mathrm{T}}, \tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathtt{x}}, \tilde{\mathbb{l}}\rangle \in \tilde{\mathbf{Conf}}$, assume that $\tilde{t}$ is as defined in Algorithm 6.10 and that $\tilde{t}'$ is defined as $\tilde{t}$ given by Table 5.6. It is easy to see that $\mathbf{Thrd}_{\mathbf{hold}}$ as given by Algorithm 6.10 is a superset of $\mathbf{Thrd}_{\mathbf{hold}}$ as given by Table 5.6 since in the latter case, a lock might have been assigned to some thread, which will exclude that thread from $\mathbf{Thrd}_{\mathbf{hold}}$. Thus it must be that $\min(\gamma_t(\tilde{t}')) \leq \min(\gamma_t(\tilde{t}))$ and $\max(\gamma_t(\tilde{t}')) \leq \max(\gamma_t(\tilde{t}))$ (since $\tilde{t}'$ is derived based on a superset of the threads used to derive $\tilde{t}$; note that if it is a true superset, it must be that all the extra threads issue `lock` *lck* for some locks, $lck \in \mathbf{Lck}$, and have been assigned the ownership of *lck*, and that for those locks $\tilde{\text{OWN}}(\tilde{\mathbb{l}}\ lck) = \bot_{thrd}$). Thus it must be that $\mathbf{Thrd}^{\tilde{c}}_{exe} \subseteq$ EXETHRD$(\tilde{c})$, where $\mathbf{Thrd}^{\tilde{c}}_{exe}$ is as defined in Table 5.6, since EXETHRD$(\tilde{c})$ is derived based on $\tilde{t}$ but also includes all threads issuing `lock` *lck* where $lck \in \mathbf{Lck}$ and $\tilde{\text{OWN}}(\tilde{\mathbb{l}}\ lck) = \bot_{thrd}$. ■

Lemma 6.3 (Soundness of GLOBALVAR):
GLOBALVAR($\mathbf{Thrd}_{\tilde{c}}$) *is the set of variables (called global variables) for which a data dependency between two or more threads can occur in the program described by* $\mathbf{Thrd}_{\tilde{c}}$. □

PROOF. Assume that $x \in \mathbf{Var}$. First note that

- if $\{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \mid \exists l \in \mathbf{Lbl}_{\mathrm{T}} : \exists r \in \mathbf{Reg}_{\mathrm{T}} : [\texttt{store}\ r\ \texttt{to}\ x]^l\} = \emptyset$ (i.e., no thread ever writes to x), then it must be that x can be considered a constant (since $x \in \mathbf{Var}$, there must be some thread reading from it),
- if $\{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \mid \exists l \in \mathbf{Lbl}_{\mathrm{T}} : \exists r \in \mathbf{Reg}_{\mathrm{T}} : [\texttt{load}\ r\ \texttt{from}\ x]^l\} = \emptyset$ (i.e., no thread ever reads from x), then it must be that x can be considered a trash variable (since $x \in \mathbf{Var}$, there must be some thread writing to it),
- if, for some thread, $\mathrm{T}' \in \mathbf{Thrd}_{\tilde{c}}$, $\{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \mid \exists l \in \mathbf{Lbl}_{\mathrm{T}} : \exists r \in \mathbf{Reg}_{\mathrm{T}} : [\texttt{store}\ r\ \texttt{to}\ x]^l\} = \{\mathrm{T}'\}$ and $\{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \mid \exists l \in \mathbf{Lbl}_{\mathrm{T}} : \exists r \in \mathbf{Reg}_{\mathrm{T}} : [\texttt{load}\ r\ \texttt{from}\ x]^l\} = \{\mathrm{T}'\}$, then it must be that x is only read from and written to by T' (thus there cannot be any data dependency on x between two threads), and
- for a data dependency to occur on x for two threads, $\mathrm{T}', \mathrm{T}'' \in \mathbf{Thrd}_{\tilde{c}}$, it must be that $\mathrm{T}' \in \{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \mid \exists l \in \mathbf{Lbl}_{\mathrm{T}} : \exists r \in \mathbf{Reg}_{\mathrm{T}} : [\texttt{store}\ r\ \texttt{to}\ x]^l\}$, $\mathrm{T}'' \in \{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \mid \exists l \in \mathbf{Lbl}_{\mathrm{T}} : \exists r \in \mathbf{Reg}_{\mathrm{T}} : [\texttt{load}\ r\ \texttt{from}\ x]^l\}$ and $\mathrm{T}' \neq \mathrm{T}''$.

Thus, since for each $\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}$, the set $\mathbf{Var}_{\mathrm{T}}^{\texttt{load}}$ contains all variables that T might read from and the set $\mathbf{Var}_{\mathrm{T}}^{\texttt{store}}$ contains all variables that T might write to, it must be that $\{x \in \mathbf{Var} \mid \exists \mathrm{T}', \mathrm{T}'' \in \mathbf{Thrd}_{\tilde{c}} : (\mathrm{T}' \neq \mathrm{T}'' \wedge x \in \mathbf{Var}_{\mathrm{T}'}^{\texttt{load}} \wedge x \in \mathbf{Var}_{\mathrm{T}''}^{\texttt{store}})\}$ is the set of variables for which data dependencies occur between at least two threads. ■

Lemma 6.4 (Soundness of EXELOADTHRD):
Given a configuration $\tilde{c}@\langle [\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathrm{r}}_{\mathrm{T}}, \tilde{t}_{\mathrm{T}}^{a}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle \in \mathbf{C\tilde{o}nf}$, $\{\mathrm{T} \in \mathbf{Thrd}_{exe}^{\tilde{c}} \mid \exists r \in \mathbf{Reg}_{\mathrm{T}} : \exists x \in \mathrm{GLOBALVAR}(\mathbf{Thrd}_{\tilde{c}}) : \mathrm{STM}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{load}\ r\ \texttt{from}\ x]^{pc_{\mathrm{T}}}\} \subseteq \mathrm{EXELOADTHRD}(\tilde{c})$, *where* $\mathbf{Thrd}_{exe}^{\tilde{c}}$ *is defined as in Table 5.6.* □

PROOF. Assume that $\tilde{c}@\langle [\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathrm{r}}_{\mathrm{T}}, \tilde{t}_{\mathrm{T}}^{a}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle \in \mathbf{C\tilde{o}nf}$ and that $\mathbf{Thrd}_{exe}^{\tilde{c}}$ is defined as in Table 5.6. The proof follows directly from the fact that $\mathbf{Thrd}_{exe}^{\tilde{c}} \subseteq \mathrm{EXETHRD}(\tilde{c})$ (Lemma 6.2). ■

Lemma 6.5 (Soundness of ISDEADLOCK):
Given a configuration $\tilde{c}@\langle[\mathrm{T},pc_\mathrm{T},\tilde{\mathbb{r}}_\mathrm{T},\tilde{t}^{a}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}},\tilde{\mathbb{x}},\tilde{\mathbb{l}}\rangle \in \tilde{\mathbf{Conf}}$*, such that* $\exists\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}} : \textsc{stm}(\mathrm{T},pc_\mathrm{T}) \neq [\texttt{halt}]^{pc_\mathrm{T}}$, $\textsc{isDeadlock}(\tilde{c}) \Rightarrow \forall c \in \gamma_{conf}(\tilde{c}) : \neg\exists c'@\langle[\mathrm{T},pc'_\mathrm{T},\mathbb{r}'_\mathrm{T},t^{a\prime}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}},\mathbb{x}',\mathbb{l}'\rangle \in \mathbf{Conf} : (c \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c' \wedge \forall\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}} : \textsc{stm}(\mathrm{T},pc'_\mathrm{T}) = [\texttt{halt}]^{pc'_\mathrm{T}})$*, where* c *and* c' *are valid concrete configurations (c.f., Definition 4.4); i.e., if* $\textsc{isDeadlock}(\tilde{c})$*, then* $\tilde{c}$ *does not represent any concrete configuration that can possibly reach a final state.* □

PROOF. Assume that $\tilde{c}@\langle[\mathrm{T},pc_\mathrm{T},\tilde{\mathbb{r}}_\mathrm{T},\tilde{t}^{a}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}},\tilde{\mathbb{x}},\tilde{\mathbb{l}}\rangle \in \tilde{\mathbf{Conf}}$, such that $\exists\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}} : \textsc{stm}(\mathrm{T},pc_\mathrm{T}) \neq [\texttt{halt}]^{pc_\mathrm{T}}$ (note that this assumption fulfills $\neg\textsc{isFinal}(\tilde{c})$) and that $\textsc{isDeadlock}(\tilde{c})$. Note that it must be that $\mathbf{Thrd}_{\tilde{c}} = \mathbf{Thrd}$ (otherwise, $\neg\textsc{isDeadlock}(\tilde{c})$).

Since $\mathbf{Thrd}_{\texttt{lock}} = \{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}} \mid \exists lck\in\mathbf{Lck} : (\textsc{stm}(\mathrm{T},pc_\mathrm{T}) = [\texttt{lock}\ lck]^{pc_\mathrm{T}} \wedge \tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck) \notin \{\bot_{thrd},\mathrm{T}\} \wedge \tilde{\textsc{st}}(\tilde{\mathbb{l}}\ lck) = locked)\}$ and $E = \{(\mathrm{T},\mathrm{T}') \mid \mathrm{T},\mathrm{T}' \in \mathbf{Thrd}_{\texttt{lock}} \wedge \exists lck\in\mathbf{Lck} : (\textsc{stm}(\mathrm{T},pc_\mathrm{T}) = [\texttt{lock}\ lck]^{pc_\mathrm{T}} \wedge \tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck) = \mathrm{T}')\}$, it is easy to see that $(\mathbf{Thrd}_{\texttt{lock}},E)$ is a graph where the vertices (in $\mathbf{Thrd}_{\texttt{lock}}$) represent threads that are waiting to acquire a lock that is currently acquired (i.e., owned and *locked*) by some other thread, and each edge, $(\mathrm{T},\mathrm{T}') \in E$, describes a dependency (i.e., T is waiting to acquire a lock that is currently acquired by T'). But then it must be that if $\textsc{cycle}(\mathbf{Thrd}_{\texttt{lock}},E)$, then for all $c \in \gamma_{conf}(\tilde{c})$ (such that c is valid) there exists a deadlock in c (since $(\mathbf{Thrd}_{\texttt{lock}},E)$ contains at least one cycle; Lemma 6.1), and thus, $\forall c \in \gamma_{conf}(\tilde{c}) : \neg\exists c'@\langle[\mathrm{T},pc'_\mathrm{T},\mathbb{r}'_\mathrm{T},t^{a\prime}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}},\mathbb{x}',\mathbb{l}'\rangle \in \mathbf{Conf} : (c \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c' \wedge \forall\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}} : \textsc{stm}(\mathrm{T},pc'_\mathrm{T}) = [\texttt{halt}]^{pc'_\mathrm{T}})$. (Note that if for some thread, $\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}$, and lock, $lck \in \mathbf{Lck}$, $\textsc{stm}(\mathrm{T},pc_\mathrm{T}) = [\texttt{lock}\ lck]^{pc_\mathrm{T}}$, $\tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck) \notin \{\bot_{thrd},\mathrm{T}\}$ and $\tilde{\textsc{st}}(\tilde{\mathbb{l}}\ lck) = unlocked$, then $\mathrm{T} \notin \mathbf{Thrd}_{\texttt{lock}}$ since $\tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck)$ has not yet acquired lck.)

If $\exists\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}} : \exists lck\in\mathbf{Lck} : (\textsc{stm}(\mathrm{T},pc_\mathrm{T}) = [\texttt{lock}\ lck]^{pc_\mathrm{T}} \wedge \tilde{\textsc{st}}(\tilde{\mathbb{l}}\ lck) = locked \wedge \tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck) \neq \bot_{thrd} \wedge \textsc{stm}(\tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck),pc_{\tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck)}) = [\texttt{halt}]^{pc_{\tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck)}})$, it is easy to see that $\tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck)$ will never issue $\texttt{unlock}\ lck$ (c.f., Tables 5.5 and 5.6) and thus $\forall c \in \gamma_{conf}(\tilde{c}) : \neg\exists c'@\langle[\mathrm{T},pc'_\mathrm{T},\mathbb{r}'_\mathrm{T},t^{a\prime}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}},\mathbb{x}',\mathbb{l}'\rangle \in \mathbf{Conf} : (c \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c' \wedge \forall\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}} : \textsc{stm}(\mathrm{T},pc'_\mathrm{T}) = [\texttt{halt}]^{pc'_\mathrm{T}})$.

This concludes the proof. ■

Lemma 6.6 (Soundness of ISTIMEOUT):
Given a configuration, $\tilde{c}@\langle[\mathrm{T},pc_\mathrm{T},\tilde{\mathbb{r}}_\mathrm{T},\tilde{t}^{a}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}},\tilde{\mathbb{x}},\tilde{\mathbb{l}}\rangle \in \tilde{\mathbf{Conf}}$*, and timeout,* $\tilde{t}_{to} \in \tilde{\mathbf{Time}}$*, such that* $\exists\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}} : \textsc{stm}(\mathrm{T},pc_\mathrm{T}) \neq [\texttt{halt}]^{pc_\mathrm{T}}$ *and* $\neg\textsc{isDeadlock}(\tilde{c})$, $\textsc{isTimeout}(\tilde{c},\tilde{t}_{to}) \Rightarrow \forall c \in \gamma_{conf}(\tilde{c}) : \neg\exists c'@$

$\langle[\mathrm{T}, pc'_{\mathrm{T}}, \mathbb{r}'_{\mathrm{T}}, t'^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \mathbb{x}', \mathbb{l}'\rangle \in \mathbf{Conf} : (c \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c' \wedge \forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : (\textsc{stm}(\mathrm{T}, pc'_{\mathrm{T}}) = [\mathtt{halt}]^{pc'_{\mathrm{T}}} \wedge t'^{a}_{\mathrm{T}} \leq \max(\gamma_t(\tilde{t}_{to}))))$, *where* c *and* c' *are valid concrete configurations (c.f., Definition 4.4); i.e., if* $\textsc{isTimeout}(\tilde{c}, \tilde{t}_{to})$, *then* $\tilde{c}$ *does not represent any concrete configuration that can possibly reach a final state before the given timeout (i.e., before* $\max(\gamma_t(\tilde{t}_{to}))$*).* □

Proof. Assume that $\tilde{c}@\langle[\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \tilde{\mathbf{Conf}}$ and $\tilde{t}_{to} \in \tilde{\mathbf{Time}}$ are such that $\exists \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) \neq [\mathtt{halt}]^{pc_{\mathrm{T}}}$, $\neg\textsc{isDeadlock}(\tilde{c})$ and $\textsc{isTimeout}(\tilde{c}, \tilde{t}_{to})$.

Since $\textsc{isTimeout}(\tilde{c}, \tilde{t}_{to})$, it must be that $\forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : (\textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) \neq [\mathtt{halt}]^{pc_{\mathrm{T}}} \Rightarrow (\tilde{t}_{to} \tilde{<}_t (\tilde{t}^{a}_{\mathrm{T}} \tilde{+}_t \textsc{absTime}(\tilde{c}, \mathrm{T})) \vee (\mathbf{Thrd}_{\tilde{c}} \subset \mathbf{Thrd} \wedge \exists lck \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) = [\mathtt{lock}\ lck]^{pc_{\mathrm{T}}} \wedge \textsc{oW̃n}(\tilde{\mathbb{l}}\ lck) \notin \{\bot_{thrd}, \mathrm{T}\}))))$. For all threads, $\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}$, such that $\textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) \neq [\mathtt{halt}]^{pc_{\mathrm{T}}} \wedge \tilde{t}_{to} \tilde{<}_t (\tilde{t}^{a}_{\mathrm{T}} \tilde{+}_t \textsc{absTime}(\tilde{c}, \mathrm{T}))$, it is easy to see that $\neg\exists c'@\langle[\mathrm{T}, pc'_{\mathrm{T}}, \mathbb{r}'_{\mathrm{T}}, t'^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \mathbb{x}', \mathbb{l}'\rangle \in \mathbf{Conf} : (c \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c' \wedge \textsc{stm}(\mathrm{T}, pc'_{\mathrm{T}}) = [\mathtt{halt}]^{pc'_{\mathrm{T}}} \wedge t'^{a}_{\mathrm{T}} \leq \max(\gamma_t(\tilde{t}_{to})))$ (c.f., Assumptions 4.1 and 5.50). Thus, for all other threads, $\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}$, such that $\exists lck \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) = [\mathtt{lock}\ lck]^{pc_{\mathrm{T}}} \wedge \textsc{oW̃n}(\tilde{\mathbb{l}}\ lck) \notin \{\bot_{thrd}, \mathrm{T}\})$, it must be that $\neg\exists c'@\langle[\mathrm{T}, pc'_{\mathrm{T}}, \mathbb{r}'_{\mathrm{T}}, t'^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \mathbb{x}', \mathbb{l}'\rangle \in \mathbf{Conf} : (c \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c' \wedge \textsc{stm}(\mathrm{T}, pc'_{\mathrm{T}}) = [\mathtt{halt}]^{pc'_{\mathrm{T}}} \wedge t'^{a}_{\mathrm{T}} \leq \max(\gamma_t(\tilde{t}_{to})))$ since the respective locks cannot possibly be released at any time, t, such that $t \leq \max(\gamma_t(\tilde{t}_{to}))$ (c.f., Assumptions 4.1 and 5.50).

This concludes the proof. ■

Lemma 6.7 (Soundness of ISVALID):
Given a configuration $\tilde{c}@\langle[\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \tilde{\mathbf{Conf}}$ *and a time,* $\tilde{t}_{to} \in \tilde{\mathbf{Time}}$*, such that* $\exists \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) \neq [\mathtt{halt}]^{pc_{\mathrm{T}}}$, $\neg\textsc{isDeadlock}(\tilde{c})$ *and* $\neg\textsc{isTimeout}(\tilde{c}, \tilde{t}_{to})$, $\neg\textsc{isValid}(\tilde{c}, \tilde{t}_{to}) \Rightarrow \neg\exists \tilde{c}' \in \tilde{\mathbf{Conf}} : (\tilde{c} \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}' \wedge \exists c@\langle[\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \mathbb{x}, \mathbb{l}\rangle \in \gamma_{conf}(\tilde{c}') : (\exists c' \in \mathbf{Conf} : c' \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c \wedge \forall lck \in \mathbf{Lck} : (\textsc{stt}(\mathbb{l}\ lck) = unlocked \Rightarrow \textsc{own}(\mathbb{l}\ lck) = \bot_{thrd})))$*; i.e.,* $\tilde{c}$ *can never lead to a configuration that could represent at least one valid concrete configuration (c.f., Definition 4.4).* □

Proof. Assume that $\tilde{c}@\langle[\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \tilde{\mathbf{Conf}}$ and $\tilde{t}_{to} \in \tilde{\mathbf{Time}}$ are such that $\exists \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : \textsc{stm}(\mathrm{T}, pc_{\mathrm{T}}) \neq [\mathtt{halt}]^{pc_{\mathrm{T}}}$, $\neg\textsc{isDeadlock}(\tilde{c})$, $\neg\textsc{isTimeout}(\tilde{c}, \tilde{t}_{to})$ and $\neg\textsc{isValid}(\tilde{c}, \tilde{t}_{to})$. Then it must be that

1. $\neg(\mathbf{Thrd}_{\tilde{c}} = \mathbf{Thrd} \Rightarrow \neg\textsc{cycle}(\mathbf{Thrd}_{\mathtt{lock}}, E))$ (i.e., $\mathbf{Thrd}_{\tilde{c}} = \mathbf{Thrd} \wedge \textsc{cycle}(\mathbf{Thrd}_{\mathtt{lock}}, E)$), where $\mathbf{Thrd}_{\mathtt{lock}} = \{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} \mid \exists lck \in$

$\mathbf{Lck} : (\text{STM}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc_{\mathrm{T}}} \wedge \text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck) \notin \{\bot_{thrd}, \mathrm{T}\})\}$ and $E = \{(\mathrm{T}, \mathrm{T}') \mid \mathrm{T}, \mathrm{T}' \in \mathbf{Thrd}_{\texttt{lock}} \wedge \exists lck \in \mathbf{Lck} : (\text{STM}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc_{\mathrm{T}}} \wedge \text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck) = \mathrm{T}')\}$, or

2. $\neg\forall lck \in \mathbf{Lck} : \forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : ((\text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck) = \mathrm{T} \wedge \text{S}\tilde{\text{T}}\text{T}(\tilde{\mathbb{l}}\ lck) = unlocked) \Rightarrow (\text{STM}(\mathrm{T}, pc_{\mathrm{T}}) \neq [\texttt{halt}]^{pc_{\mathrm{T}}} \wedge \text{D}\tilde{\text{L}}(\tilde{\mathbb{l}}\ lck)\ \tilde{\not>}_t\ (\tilde{t}^a_{\mathrm{T}} \dot{-}_t \text{ABSTIME}(\tilde{c}, \mathrm{T}))))$.

If $\mathbf{Thrd}_{\tilde{c}} = \mathbf{Thrd} \wedge \text{CYCLE}(\mathbf{Thrd}_{\texttt{lock}}, E)$, then it must be that there is a cycle in the dependency graph, $(\mathbf{Thrd}_{\texttt{lock}}, E)$, for threads waiting to acquire some lock (Lemma 6.1). Since $\neg\text{ISDEADLOCK}(\tilde{c})$, it must be that this cycle involves at least one lock, $lck \in \mathbf{Lck}$, such that for some thread, $\mathrm{T} \in \mathbf{Thrd}$, $\text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck) \notin \{\bot_{thrd}, \mathrm{T}\}$ and $\text{S}\tilde{\text{T}}\text{T}(\tilde{\mathbb{l}}\ lck) = unlocked$ (c.f., Algorithm 6.4). But then it is easy to see that $\neg\exists \tilde{c}' \in \tilde{\mathbf{Conf}} : (\tilde{c} \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}' \wedge \exists c @ \langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \mathbb{x}, \mathbb{l} \rangle \in \gamma_{conf}(\tilde{c}') : (\exists c' \in \mathbf{Conf} : c' \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c \wedge \forall lck \in \mathbf{Lck} : (\text{STT}(\mathbb{l}\ lck) = unlocked \Rightarrow \text{OWN}(\mathbb{l}\ lck) = \bot_{thrd})))$ (c.f., Tables 5.5 and 5.6 and Lemma 4.5).

Note that if $\neg\forall lck \in \mathbf{Lck} : \forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : ((\text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck) = \mathrm{T} \wedge \text{S}\tilde{\text{T}}\text{T}(\tilde{\mathbb{l}}\ lck) = unlocked) \Rightarrow (\text{STM}(\mathrm{T}, pc_{\mathrm{T}}) \neq [\texttt{halt}]^{pc_{\mathrm{T}}} \wedge \text{D}\tilde{\text{L}}(\tilde{\mathbb{l}}\ lck)\ \tilde{\not>}_t\ (\tilde{t}^a_{\mathrm{T}} \dot{-}_t \text{ABSTIME}(\tilde{c}, \mathrm{T}))))$, then it must be that $\exists lck \in \mathbf{Lck} : \exists \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : (\text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck) = \mathrm{T} \wedge \text{S}\tilde{\text{T}}\text{T}(\tilde{\mathbb{l}}\ lck) = unlocked \wedge (\text{STM}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{halt}]^{pc_{\mathrm{T}}} \vee \text{D}\tilde{\text{L}}(\tilde{\mathbb{l}}\ lck)\ \tilde{<}_t\ (\tilde{t}^a_{\mathrm{T}} \dot{-}_t \text{ABSTIME}(\tilde{c}, \mathrm{T}))))$.

If $\exists lck \in \mathbf{Lck} : \exists \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : (\text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck) = \mathrm{T} \wedge \text{S}\tilde{\text{T}}\text{T}(\tilde{\mathbb{l}}\ lck) = unlocked \wedge \text{STM}(\mathrm{T}, pc_{\mathrm{T}}) = [\texttt{halt}]^{pc_{\mathrm{T}}})$, then it is easy to see that $\neg\exists \tilde{c}' \in \tilde{\mathbf{Conf}} : (\tilde{c} \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}' \wedge \exists c @ \langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \mathbb{x}, \mathbb{l} \rangle \in \gamma_{conf}(\tilde{c}') : (\exists c' \in \mathbf{Conf} : c' \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c \wedge \forall lck \in \mathbf{Lck} : (\text{STT}(\mathbb{l}\ lck) = unlocked \Rightarrow \text{OWN}(\mathbb{l}\ lck) = \bot_{thrd})))$ since, for the given lock, the owner will be T but the state will remain *unlocked* for all configurations following $\tilde{c}$ (c.f., Tables 5.5 and 5.6).

If $\exists lck \in \mathbf{Lck} : \exists \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}} : (\text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck) = \mathrm{T} \wedge \text{S}\tilde{\text{T}}\text{T}(\tilde{\mathbb{l}}\ lck) = unlocked \wedge \text{D}\tilde{\text{L}}(\tilde{\mathbb{l}}\ lck)\ \tilde{<}_t\ (\tilde{t}^a_{\mathrm{T}} \dot{-}_t \text{ABSTIME}(\tilde{c}, \mathrm{T})))$, then it is easy to see that $\neg\exists \tilde{c}' \in \tilde{\mathbf{Conf}} : (\tilde{c} \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}' \wedge \exists c @ \langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^a_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \mathbb{x}, \mathbb{l} \rangle \in \gamma_{conf}(\tilde{c}') : (\exists c' \in \mathbf{Conf} : c' \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c \wedge \forall lck \in \mathbf{Lck} : (\text{STT}(\mathbb{l}\ lck) = unlocked \Rightarrow \text{OWN}(\mathbb{l}\ lck) = \bot_{thrd})))$ since $\text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck)$ must be one of the threads that issue $\texttt{lock}\ lck$ (and thus determines $\text{DL}(\mathbb{l}\ lck)$) in the concrete case for $\forall lck \in \mathbf{Lck} : (\text{STT}(\mathbb{l}\ lck) = unlocked \Rightarrow \text{OWN}(\mathbb{l}\ lck) = \bot_{thrd})$ to hold. But since $\text{D}\tilde{\text{L}}(\tilde{\mathbb{l}}\ lck)\ \tilde{<}_t\ (\tilde{t}^a_{\mathrm{T}} \dot{-}_t \text{ABSTIME}(\tilde{c}, \mathrm{T}))$ and $\text{DL}(\mathbb{l}\ lck) \in \gamma_t(\text{D}\tilde{\text{L}}(\tilde{\mathbb{l}}\ lck))$ for any given transitional sequence (c.f., Lemma 5.58), there cannot be any $c \in \gamma_{conf}(\tilde{c})$ such that $\text{O}\tilde{\text{W}}\text{N}(\tilde{\mathbb{l}}\ lck) = \mathrm{T}$ (c.f., Assumptions 4.1 and 5.50 and Tables 4.2, 4.3, 5.5 and 5.6).

This concludes the proof. ∎

Lemma 6.8 (Soundness of ABSEXE):
Given the sets of valid configurations $C \in \mathscr{P}(\mathbf{Conf})$ *(c.f., Definition 4.4) and* $\tilde{C} \in \mathscr{P}(\tilde{\mathbf{Conf}})$*, such that* $\forall c@\langle[\mathrm{T}, pc_\mathrm{T}, \mathbb{r}_\mathrm{T}, t^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_1}, \mathbb{x}, \mathbb{l}\rangle \in C$: $(\forall\langle[\mathrm{T}, pc^{\tilde{c}}_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_2}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \tilde{C} : (\mathbf{Thrd}_1 = \mathbf{Thrd}_2 = \mathbf{Thrd}) \wedge \exists\tilde{c} \in \tilde{C} : c \in \gamma_{conf}(\tilde{c})) \wedge |\mathbf{Thrd}| < \infty \wedge \forall\tilde{c}@\langle[\mathrm{T}, pc^{\tilde{c}}_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \tilde{C} : \forall lck \in \mathbf{Lck} : \min(\gamma_t(\tilde{\mathbb{l}}\ lck) = -\infty))$*, and the times* $t_{to} \in \mathbf{Time}$ *and* $\tilde{t}_{to} \in \tilde{\mathbf{Time}}$*, such that* $t_{to} = \max(\gamma_t(\tilde{t}_{to}))$*,* $(\tilde{C}^f, \tilde{C}^d, \tilde{C}^t)@\textsc{absExe}(\tilde{C}, \tilde{t}_{to})$ *is such that*

$$
\begin{array}{l}
\forall c \in C : \forall c'@\langle[\mathrm{T}, pc'_\mathrm{T}, \mathbb{r}'_\mathrm{T}, t^{a\prime}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}', \mathbb{l}'\rangle \in \mathbf{Conf} : \\
\quad ((c \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c' \wedge \forall \mathrm{T} \in \mathbf{Thrd} : \textsc{stm}(\mathrm{T}, pc'_\mathrm{T}) = [\texttt{halt}]^{pc'_\mathrm{T}}) \Rightarrow \\
\qquad (\tilde{C}^t \neq \emptyset \vee \\
\qquad\quad \exists\tilde{c}@\langle[\mathrm{T}, pc^{\tilde{c}}_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \tilde{C}^f : \forall \mathrm{T} \in \mathbf{Thrd} : \\
\qquad\qquad\qquad\qquad (pc^{\tilde{c}}_\mathrm{T} = pc'_\mathrm{T} \wedge t^{a\prime}_\mathrm{T} \in \gamma_t(\tilde{t}^a_\mathrm{T})))) \wedge \\
\forall c \in C : \forall c'@\langle[\mathrm{T}, pc'_\mathrm{T}, \mathbb{r}'_\mathrm{T}, t^{a\prime}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}', \mathbb{l}'\rangle \in \mathbf{Conf} : \\
\quad ((c \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c' \wedge (\textsc{cycle}(\mathbf{Thrd}^{c'}_{\texttt{lock}}, E^{c'}) \vee \\
\qquad\qquad\qquad \exists \mathrm{T} \in \mathbf{Thrd} : \exists lck \in \mathbf{Lck} : \\
\qquad\qquad\qquad\quad (\textsc{stm}(\mathrm{T}, pc'_\mathrm{T}) = [\texttt{lock}\ lck]^{pc'_\mathrm{T}} \wedge \\
\qquad\qquad\qquad\quad \textsc{own}(\mathbb{l}'\ lck) \notin \{\bot_{thrd}, \mathrm{T}\} \wedge \\
\qquad\qquad\qquad\quad \textsc{stm}(\textsc{own}(\mathbb{l}'\ lck), pc'_{\textsc{own}(\mathbb{l}'\ lck)}) = \\
\qquad\qquad\qquad\qquad\qquad\qquad [\texttt{halt}]^{pc'_{\textsc{own}(\mathbb{l}'\ lck)}}))) \Rightarrow \\
\quad (\tilde{C}^t \neq \emptyset \vee \tilde{C}^d \neq \emptyset))
\end{array}
$$

where $\mathbf{Thrd}^{c'}_{\texttt{lock}} = \{\mathrm{T} \in \mathbf{Thrd} \mid \exists lck \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}, pc'_\mathrm{T}) = [\texttt{lock}\ lck]^{pc'_\mathrm{T}} \wedge \textsc{own}(\mathbb{l}'\ lck) \notin \{\bot_{thrd}, \mathrm{T}\})\}$ *and* $E^{c'} = \{(\mathrm{T}, \mathrm{T}') \mid \mathrm{T}, \mathrm{T}' \in \mathbf{Thrd}^{c'}_{\texttt{lock}} \wedge \exists lck \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}, pc'_\mathrm{T}) = [\texttt{lock}\ lck]^{pc'_\mathrm{T}} \wedge \textsc{own}(\mathbb{l}'\ lck) = \mathrm{T}')\}$*, whenever it terminates. In other words, if a concrete configuration in C could be executed into a final configuration,* ABSEXE *will either find a final abstract configuration that safely represents the timing behavior of the concrete final configuration or reach a timeout for the corresponding abstract transition sequence, and thus,* $\tilde{C}^t \neq \emptyset$*. If a concrete configuration in C could be executed into a state from which a final configuration cannot be reached,* ABSEXE *will either find the corresponding abstract situation (i.e.,* $\tilde{C}^d \neq \emptyset$*) or reach a timeout for the corresponding abstract transition sequence, and thus,* $\tilde{C}^t \neq \emptyset$*.*

It is also the case that if $\tilde{C}^d \cup \tilde{C}^t = \emptyset$*, then all concrete configurations in C are guaranteed to, along all possible paths, reach a state in which all threads issue the* `halt`*-statement (i.e., reach the final state, or in other words, terminate).*

Furthermore (if $\tilde{C}^d \cup \tilde{C}^t = \emptyset$*):*

$$\begin{aligned}&\forall c \in C : \forall c' @ \langle [\mathrm{T}, pc'_{\mathrm{T}}, \mathbb{r}'_{\mathrm{T}}, t'^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}', \mathbb{l}' \rangle \in \mathbf{Conf} : \\ &((c \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c' \wedge \forall \mathrm{T} \in \mathbf{Thrd} : \mathrm{STM}(\mathrm{T}, pc'_{\mathrm{T}}) = [\mathtt{halt}]^{pc'_{\mathrm{T}}}) \Rightarrow \\ &\qquad \exists \langle [\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle \in \tilde{C}^f : (pc^{\tilde{c}}_{\mathrm{T}} = pc'_{\mathrm{T}} \wedge t'^{a}_{\mathrm{T}} \in \gamma_t(\tilde{t}^{a}_{\mathrm{T}})))\end{aligned} \quad \square$$

PROOF. Assume that the sets of valid configurations $C \in \mathscr{P}(\mathbf{Conf})$ and $\tilde{C} \in \mathscr{P}(\tilde{\mathbf{Conf}})$ are such that $\forall c @ \langle [\mathrm{T}, pc_{\mathrm{T}}, \mathbb{r}_{\mathrm{T}}, t^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_1}, \mathbb{x}, \mathbb{l} \rangle \in C : (\forall \langle [\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_2}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle \in \tilde{C} : (\mathbf{Thrd}_1 = \mathbf{Thrd}_2 = \mathbf{Thrd}) \wedge \exists \tilde{c} \in \tilde{C} : c \in \gamma_{conf}(\tilde{c})) \wedge |\mathbf{Thrd}| < \infty \wedge \forall \tilde{c} @ \langle [\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle \in \tilde{C} : \forall lck \in \mathbf{Lck} : \min(\gamma_t(\tilde{\mathbb{l}}\ lck) = -\infty))$, the times $t_{to} \in \mathbf{Time}$ and $\tilde{t}_{to} \in \tilde{\mathbf{Time}}$, are such that $t_{to} = \max(\gamma_t(\tilde{t}_{to}))$, and that $(\tilde{C}^f, \tilde{C}^d, \tilde{C}^t) = \mathrm{ABSEXE}(\tilde{C}, \tilde{t}_{to})$.

This proof will partly be conducted using induction on the considered level of recursion, where level 0 is the base level (i.e., the level where for any considered $\tilde{c}$, $\mathbf{Thrd}_{\tilde{c}} = \mathbf{Thrd}$) and level $n \geq 0$ is the bottom level (i.e., the level from which no more recursion occurs, which is also referred to as the maximum level of recursion), while assuming that all sequentially preceding `load`-statements in all threads for any considered configuration on any level of recursion have been safely approximated. Before beginning the induction part of the proof, first note that:

- The overall structure of the algorithm is of the work-list type; i.e., given an item (abstract configuration in this case) that is extracted from a work-list, new items are generated, based on some rules, and are either added to the work-list (and will thus eventually be extracted themselves) or saved as output items if some condition is fulfilled. When the work-list is empty, the algorithm terminates.

- Since $(\tilde{C}^f, \tilde{C}^d, \tilde{C}^t) = \mathrm{ABSEXE}(\tilde{C}, \tilde{t}_{to})$, it must be that the algorithm terminates for the considered input (i.e., $\tilde{C}$ and $\tilde{t}_{to}$).

- The structure of the algorithm is such that, for any $\tilde{c} \in \tilde{\mathbf{Conf}}$ and $\tilde{t}_{to} \in \tilde{\mathbf{Time}}$, $\mathrm{ISDEADLOCK}(\tilde{c})$ is only issued when $\neg\mathrm{ISFINAL}(\tilde{c})$, $\mathrm{ISTIMEOUT}(\tilde{c}, \tilde{t}_{to})$ is only issued when $\neg\mathrm{ISDEADLOCK}(\tilde{c})$, and $\mathrm{ISVALID}(\tilde{c}, \tilde{t}_{to})$ is only issued when $\neg\mathrm{ISTIMEOUT}(\tilde{c}, \tilde{t}_{to})$. This means that the requirements of Algorithms 6.4, 6.5 and 6.6 are fulfilled.

- The timing behaviors of the threads included on any recursion level are safely given by ABSTIME (Assumption 5.50).

- The maximum level (i.e., depth) of any recursion pattern is $|\mathbf{Thrd}| - 1$ since $|\text{EXETHRD}(\tilde{c})| > 1$ for recursion to occur and $|\mathbf{Thrd}| \geq |\text{EXETHRD}(\tilde{c})|$ for any $\tilde{c} \in \tilde{\mathbf{Conf}}$ (c.f., Algorithm 6.10). Since $|\mathbf{Thrd}| < \infty$, it must thus be that $0 \leq n \leq |\mathbf{Thrd}| - 1 < \infty$. But then, since the recursion depth is of a finite size, it must be that the recursion eventually stops for any considered case.

- When the considered level of recursion, i, is greater than 0, it is easy to see that $\mathbf{Thrd}_i \subset \mathbf{Thrd}$, where $\mathbf{Thrd}_i$ is the set of threads included in any configuration on recursion level i for the considered recursion pattern. Note that $\mathbf{Thrd}_0 = \mathbf{Thrd}$.

- The timeout, $\tilde{t}^i_{to}$, for recursion level $i > 0$ is such that $\max(\gamma_t(\tilde{t}^i_{to})) \leq \max(\gamma_t(\tilde{t}^{i-1}_{to}))$ since $\tilde{t}^i_{to} = \tilde{t}^{a\prime}_{\mathrm{T}} \,\tilde{\sqcap}_t\, (\tilde{t}^{i-1}_{to} \,\tilde{\sqcup}_t\, \alpha_t(\{-\infty\}))$, where $\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}$ is the thread that will be removed from the configurations on recursion level i. This means that the timeout cannot be shifted into the future when recursion occurs.

  Figure 6.1 illustrates a case where $n = 4$, $\tilde{t}^0_{to}$ is the timeout at the base level (i.e., recursion level 0) and for all $i \in \{1,2,3,4\}$, $\tilde{t}^i_{to}$ is the timeout at recursion level i, T_{i-1} is the thread not included in configurations at recursion level i and $\tilde{t}^{a\prime}_{\mathrm{T}_{i-1}} = \tilde{t}^{a}_{\mathrm{T}_{i-1}} \,\tilde{+}_t\, \text{ABSTIME}(\tilde{c}^{i-1}, \mathrm{T}_{i-1})$.

- Assume that a thread, $\mathrm{T} \in \mathbf{Thrd}$, issues a `load`-statement at some recursion level $i-2$, where $i \geq 2$, and has been removed from all configurations at recursion level $i-1$ and beyond, and that no events occurring after $\tilde{t}^{i-1}_{to}$ can affect the loaded value. If some other thread, $\mathrm{T}' \in \mathbf{Thrd}$, issues a possibly unsafe `load`-statement at recursion level $i-1$, then a new recursion level, i, will be created to determine a safe write history before the `load` in T' is evaluated. Then it is easy to see that any event occurring after $\tilde{t}^{a\prime}_{\mathrm{T}'}$ cannot affect the value loaded by T'. But then it is easy to see that the value loaded by T at recursion level $i-2$ cannot be affected by any event occurring after $\tilde{t}^i_{to} = \tilde{t}^{a\prime}_{\mathrm{T}'} \,\tilde{\sqcap}_t\, (\tilde{t}^{i-1}_{to} \,\tilde{\sqcup}_t\, \alpha_t(\{-\infty\}))$ for the considered recursion instance at level i. Thus, for all recursion levels $i \in \{1,\ldots,n\}$, the timeout for recursion level i, as determined by the algorithm, is safe since the accumulated time for a thread cannot decrease (c.f., Assumption 5.50).

- The structure of the algorithm (i.e., on a recursion level, one new recursion-instance is created for each thread that is executing a possibly unsafe `load`-statement) gives that all possible cases, for in which order `load`-statements in different threads can be issued, are considered.

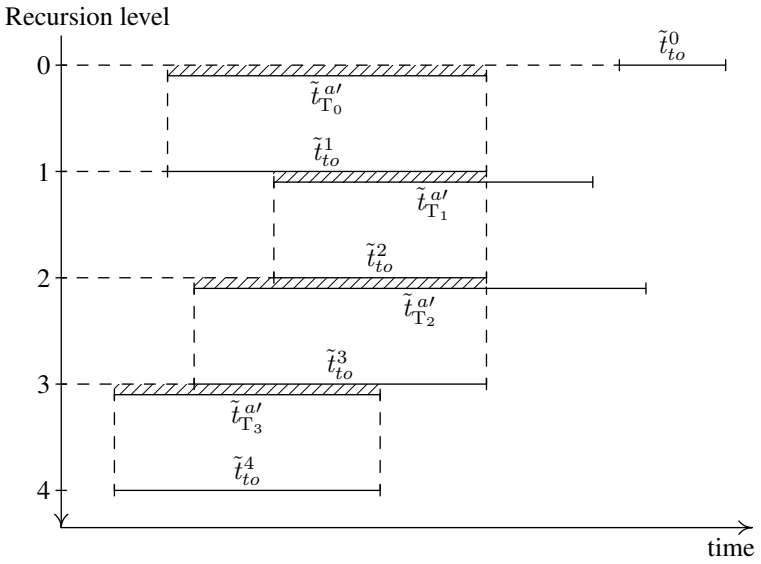


Figure 6.1: Illustration of how the timeout, $\tilde{t}_{to}$, for a new level of recursion in ABSEXE is determined.

Assume that, given some configuration, $\tilde{c}@\langle[\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_i}, \tilde{\mathbb{x}}, \tilde{\mathbb{I}}\rangle$, and timeout, $\tilde{t}_{to}$, on recursion level i, where $0 \leq i < n$, some thread, $\mathrm{T}_i \in \mathbf{Thrd}_i$, issues a possibly unsafe `load`-statement (which means that a deeper recursion level, $i+1$, will exist) that cannot be affected by any event occurring after $\tilde{t}_{to}$. Further assume that the local thread states (i.e., program counters, register values and accumulated execution times) and write history for all variables as given by $\tilde{c}$ safely approximate the possible concrete thread states and variable values given the considered program point and the corresponding concrete transition sequences (if any), and that all `load`-statements on recursion levels $i+1$ to n are safely approximated. This comprises the induction assumption.

Since the local thread states and write history for all variables as given by $\tilde{c}$ safely approximate the possible concrete thread states and variable values given by the configuration at the end of the corresponding concrete transition sequences and all `load`-statements on recursion levels $i+1$ to n are safely approximated, it must be that all $\tilde{c}' \in \mathbf{C\tilde{o}nf}$, such that $\tilde{c} \xrightarrow[prg]{\sim} \ldots \xrightarrow[prg]{\sim} \tilde{c}'$, safely approximate the possible concrete thread states

and variable values given the considered program point and the corresponding concrete transition sequences since $\xrightarrow[prg]{\sim}$ is used to approximate the execution of all statements except `load`-statements that are possibly unsafe (c.f., Lemmas 5.55, 5.56 and 5.57). Thus, it must be that $(\tilde{C}^f_{i+1}, \tilde{C}^d_{i+1}, \tilde{C}^t_{i+1})$ @ ABSEXE$(\{\langle [\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_i\setminus\{\mathrm{T}_i\}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle\}, (\tilde{t}^a_{\mathrm{T}_i} \mathbin{\tilde{+}_t} \text{ABSTIME}(\tilde{c}, \mathrm{T}_i)) \mathbin{\tilde{\sqcap}_t} (\tilde{t}_{to} \mathbin{\tilde{\sqcup}_t} \alpha_t(\{-\infty\})))$ is such that $\tilde{\bigsqcup}_{val}\{\text{READ}(\tilde{\mathbb{x}}', x, \mathrm{T}_i, \tilde{t}^a_{\mathrm{T}_i} \mathbin{\tilde{+}_t} \text{ABSTIME}(\tilde{c}, \mathrm{T}_i)) \mid \langle \tilde{\mathbf{T}}, \tilde{\mathbb{x}}', \tilde{\mathbb{l}}'\rangle \in \tilde{C}^f_{i+1} \cup \tilde{C}^d_{i+1} \cup \tilde{C}^t_{i+1} \cup \{\tilde{c}\}\}$ safely approximates all possible concrete values that could be read by the `load`-statement in T_i for the corresponding concrete transition sequence, since (the ABSEXE instance mentioned above, corresponding to recursion level $i+1$ is considered)

1. $\{\tilde{c}'' \in \tilde{\mathbf{Conf}} \mid \tilde{c}' \xrightarrow[prg]{\sim} \tilde{c}''\}$ safely collects all transition possibilities for any given configuration, $\tilde{c}' \in \tilde{\mathbf{Conf}}$, or rather, thread, for which no possibly unsafe `load`-statements are approximated by the transition (c.f., Lemmas 5.55, 5.56 and 5.57),

2. TRIM is not used to remove old writes from the write history since $i+1 > 0$ (c.f., Table 5.6),

3. (note that $\mathbf{Thrd}_{i+1} \subset \mathbf{Thrd}$) $\forall \tilde{c}' @ \langle [\mathrm{T}, pc'_{\mathrm{T}}, \tilde{\mathbb{r}}'_{\mathrm{T}}, \tilde{t}^{a\prime}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}'}, \tilde{\mathbb{x}}', \tilde{\mathbb{l}}'\rangle \in \tilde{\mathbf{Conf}}$: $(\mathbf{Thrd}' \subset \mathbf{Thrd} \Rightarrow \neg\text{ISDEADLOCK}(\tilde{c}'))$; i.e., even if a deadlock exists in $\tilde{c}'$, it is further evaluated just in case there are threads that are not part of the deadlock and thus could affect the value of the variable which is read on the lower recursion level (c.f., Algorithm 6.4),

4. for any $\tilde{c}' \in \tilde{\mathbf{Conf}}$ and $\tilde{t}'_{to} \in \tilde{\mathbf{Time}}$, ISTIMEOUT$(\tilde{c}', \tilde{t}'_{to}) \Rightarrow \forall c \in \gamma_{conf}(\tilde{c}') : \neg\exists c' @ \langle [\mathrm{T}, pc'_{\mathrm{T}}, \mathbb{r}'_{\mathrm{T}}, t^{a\prime}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}'}}, \mathbb{x}', \mathbb{l}'\rangle \in \mathbf{Conf} : (c \xrightarrow[prg]{} \dots \xrightarrow[prg]{} c' \wedge \forall \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}'} : (\text{STM}(\mathrm{T}, pc'_{\mathrm{T}}) = [\texttt{halt}]^{pc'_{\mathrm{T}}} \wedge t^{a\prime}_{\mathrm{T}} \leq \max(\gamma_t(\tilde{t}'_{to}))))$, where c and c' are valid concrete configurations (c.f., Definition 4.4); i.e., if ISTIMEOUT$(\tilde{c}', \tilde{t}'_{to})$, then $\tilde{c}'$ does not represent any concrete configuration that can possibly reach a final state before the given timeout (Lemma 6.6), or in other words, no thread in $\tilde{c}'$ can affect the system state so that the effects are visible at or before $\tilde{t}'_{to}$ (c.f., Algorithm 6.5 and Assumption 5.50),

5. (note that $\mathbf{Thrd}_{i+1} \subset \mathbf{Thrd}$) $\forall \tilde{c}' @ \langle [\mathrm{T}, pc'_{\mathrm{T}}, \tilde{\mathbb{r}}'_{\mathrm{T}}, \tilde{t}^{a\prime}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}'}, \tilde{\mathbb{x}}', \tilde{\mathbb{l}}'\rangle \in \tilde{\mathbf{Conf}} : ((\mathbf{Thrd}' \subset \mathbf{Thrd} \wedge \neg\text{ISVALID}(\tilde{c}')) \Rightarrow \neg\forall lck \in \mathbf{Lck} : \forall \mathrm{T} \in \mathbf{Thrd}' : ((\text{OW}\tilde{\text{N}}(\tilde{\mathbb{l}}'\ lck) = \mathrm{T} \wedge \text{ST}\tilde{\text{T}}(\tilde{\mathbb{l}}'\ lck) = \mathit{unlocked}) \Rightarrow (\text{STM}(\mathrm{T}, pc'_{\mathrm{T}}) \neq [\texttt{halt}]^{pc'_{\mathrm{T}}} \wedge \text{D}\tilde{\text{L}}(\tilde{\mathbb{l}}\ lck) \mathbin{\tilde{\not<}_t} (\tilde{t}^a_{\mathrm{T}} \mathbin{\tilde{+}_t} \text{ABSTIME}(\tilde{c}, \mathrm{T})))))$, which follows

directly from Algorithm 6.6 and means that there is no possibility that $\tilde{c}'$ has any (or could lead to a configuration that has a) valid concrete counterpart (c.f., Definition 4.4 and the proof of Lemma 6.7).

It is important to notice that ISTIMEOUT captures all configurations such that all threads have either executed beyond the timeout or are waiting to acquire a lock that is currently owned by a thread that has executed beyond the timeout or is also waiting to acquire some lock (c.f., Algorithm 6.5), which means that the first mentioned thread cannot possibly acquire the lock before the timeout has passed (c.f., Tables 5.5 and 5.6 and Assumption 5.50). This means that ISTIMEOUT captures all deadlocked configurations, since ISDEADLOCK does not capture any configuration at all when the considered recursion level is greater than 0 (c.f., Algorithm 6.4), and also all configurations allowed by ISVALID, although they lack valid concrete counterparts (c.f., Algorithm 6.6).

Since $\tilde{t}^{a\prime}_{\mathrm{T}_i} = \tilde{t}^{a}_{\mathrm{T}_i} \tilde{+}_t \textsc{absTime}(\tilde{c}^i, \mathrm{T}_i)$, $pc'_{\mathrm{T}_i} = pc_{\mathrm{T}_i} + 1$ and $\tilde{\mathbb{r}}'_{\mathrm{T}_i} = \tilde{\mathbb{r}}_{\mathrm{T}_i}[r \mapsto \tilde{\bigsqcup}_{val}\{\textsc{read}(\tilde{\mathbb{x}}', x, \mathrm{T}_i, \tilde{t}^{a}_{\mathrm{T}_i} \tilde{+}_t \textsc{absTime}(\tilde{c}^i, \mathrm{T}_i)) \mid \langle \tilde{\mathbf{T}}, \tilde{\mathbb{x}}', \tilde{\mathbb{l}}' \rangle \in \tilde{C}^{f}_{i+1} \cup \tilde{C}^{d}_{i+1} \cup \tilde{C}^{t}_{i+1} \cup \{\tilde{c}^i\}\}]$ (assuming that the possibly unsafe `load`-statement issued by T_i is $[\texttt{load}\ r\ \texttt{from}\ x]^{pc_{\mathrm{T}_i}}$ for some $r \in \mathbf{Reg}_{\mathrm{T}_i}$ and $x \in \mathbf{Var}$), it must thus be that the `load`-statement in thread T_i on recursion level i is safely approximated and that the new configuration, which is added to the work-list on line 28, therefore safely approximates the local thread states (i.e., program counters, register values and accumulated execution times) for all threads and the write history for all variables as given by the possible concrete thread states and variable values in the considered program point and the corresponding concrete transition sequences (if any). But this means that all possibly unsafe `load`-statements on recursion level i are safely approximated.

Now consider recursion level n (i.e., the level from which no more recursion will occur for a given recursion pattern, which is the base case for the induction part of the proof) for the first ever occurring recursion pattern for a given transition sequence, such that no potentially unsafe `load`-statement has yet been approximated. Since no potentially unsafe `load`-statement has yet been approximated and $\forall c \in C : \exists \tilde{c} \in \tilde{C} : c \in \gamma_{conf}(\tilde{c})$, it must be that any concrete state for all threads individually, and the write history for each variable, must be safely approximated up until the considered point of the considered transition sequence (since $\xrightarrow[prg]{\sim}$ has been safely used for all transitions and $\{\tilde{c}' \in \tilde{\mathbf{Conf}} \mid \tilde{c} \xrightarrow[prg]{\sim} \tilde{c}'\}$ collects all abstract transition possibilities for any given configuration, $\tilde{c} \in \tilde{\mathbf{Conf}}$, or rather, thread; c.f., Lemmas 5.55, 5.56 and 5.57). Since no more (i.e.,

deeper) recursion will occur, it must be that for any considered configuration, $\tilde{c}@\langle[\mathrm{T}, pc_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_n}, \tilde{\mathbb{x}}, \tilde{\mathbb{I}}\rangle \in \tilde{\mathbf{Conf}}$, at level n, $|\textsc{exeThrd}(\tilde{c})| \not> 1$ or $\textsc{exeLoadThrd}(\tilde{c}) = \emptyset$. But since for any given configuration $\tilde{c}@\langle[\mathrm{T}, pc_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{I}}\rangle \in \tilde{\mathbf{Conf}}$, $\mathbf{Thrd}^{\tilde{c}}_{exe} \subseteq \textsc{exeThrd}(\tilde{c})$ (Lemma 6.2) and $\{\mathrm{T} \in \mathbf{Thrd}^{\tilde{c}}_{exe} \mid \exists r \in \mathbf{Reg}_\mathrm{T} : \exists x \in \textsc{globalVar}(\mathbf{Thrd}_{\tilde{c}}) : \textsc{stm}(\mathrm{T}, pc_\mathrm{T}) = [\texttt{load}\ r\ \texttt{from}\ x]^{pc_\mathrm{T}}\} \subseteq \textsc{exeLoadThrd}(\tilde{c})$ (Lemma 6.4), where $\mathbf{Thrd}^{\tilde{c}}_{exe}$ is as defined in Table 5.6, it must thus be that $|\mathbf{Thrd}^{\tilde{c}}_{exe}| \not> 1 \vee \{\mathrm{T} \in \mathbf{Thrd}^{\tilde{c}}_{exe} \mid \exists r \in \mathbf{Reg}_\mathrm{T} : \exists x \in \mathbf{Var_g} : \textsc{stm}(\mathrm{T}, pc^{\tilde{c}}_\mathrm{T}) = [\texttt{load}\ r\ \texttt{from}\ x]^{pc^{\tilde{c}}_\mathrm{T}}\} = \emptyset$ for all $\tilde{c} \in \tilde{\mathbf{Conf}}$ at recursion level n. Thus, it must be that $\{\tilde{c}' \in \tilde{\mathbf{Conf}} \mid \tilde{c} \xrightarrow[prg]{\sim} \tilde{c}'\}$ will be safely used to collect all the possible transitions on recursion level n until all threads either reach the final state (i.e., issue `halt`-statements) or execute beyond the timeout (c.f., Lemmas 5.55, 5.56, 5.57, 5.58, 6.5, 6.6 and 6.7). But, then it must be that all the possible concrete transition sequences for each thread are safely approximated up until the timeout point (if ever reached, and if reached before the final state) since $\forall c \in C : \exists \tilde{c} \in \tilde{C} : c \in \gamma_{conf}(\tilde{c})$. This concludes the induction part of the proof.

Given $c \in C$ and $c'@\langle[\mathrm{T}, pc'_\mathrm{T}, \mathbb{r}'_\mathrm{T}, t'^{a\prime}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}', \mathbb{I}'\rangle \in \mathbf{Conf}$, the following concrete cases (corresponding to a terminating program, a program reaching a deadlocked state and a more general case of a non-terminating program, respectively) must be considered.

1. Assume that $c \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c' \wedge \forall \mathrm{T} \in \mathbf{Thrd} : \textsc{stm}(\mathrm{T}, pc'_\mathrm{T}) = [\texttt{halt}]^{pc'_\mathrm{T}}$. Note that since all possible concrete transition sequences for each thread individually are safely approximated up until the timeout point and a final configuration is reached in the concrete case, there must be an abstract trace of transitions such that all configurations, $\tilde{c} \in \tilde{\mathbf{Conf}}$, on that trace are such that $\neg\textsc{isDeadlock}(\tilde{c})$ (c.f., Algorithm 6.4 and Lemma 6.5) and $\textsc{isValid}(\tilde{c}, \tilde{t}_{to})$ (c.f., Algorithm 6.6 and Lemma 6.7). It must also be that, eventually, a configuration, $\tilde{c}@\langle[\mathrm{T}, pc^{\tilde{c}}_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}}, \tilde{\mathbb{x}}, \tilde{\mathbb{I}}\rangle \in \tilde{\mathbf{Conf}}$, for which either $\forall \mathrm{T} \in \mathbf{Thrd} : \textsc{stm}(\mathrm{T}, pc^{\tilde{c}}_\mathrm{T}) = [\texttt{halt}]^{pc^{\tilde{c}}_\mathrm{T}}$ or $\forall \mathrm{T} \in \mathbf{Thrd} : (\textsc{stm}(\mathrm{T}, pc^{\tilde{c}}_\mathrm{T}) \neq [\texttt{halt}]^{pc^{\tilde{c}}_\mathrm{T}} \Rightarrow \tilde{t}_{to} \tilde{<}_t \tilde{t}^a_\mathrm{T} \tilde{+}_t \textsc{absTime}(\tilde{c}, \mathrm{T}))$ is derived along the corresponding (over-approximating) abstract trace of transitions.

   If $\forall \mathrm{T} \in \mathbf{Thrd} : \textsc{stm}(\mathrm{T}, pc^{\tilde{c}}_\mathrm{T}) = [\texttt{halt}]^{pc^{\tilde{c}}_\mathrm{T}}$, then it is easy to see that $\textsc{isFinal}(\tilde{c})$ (c.f., Algorithm 6.3), which means that $\tilde{c} \in \tilde{C}^f$. Thus, it must be that $\exists \tilde{c}'@\langle[\mathrm{T}, pc^{\tilde{c}'}_\mathrm{T}, \tilde{\mathbb{r}}'_\mathrm{T}, \tilde{t}'^{a\prime}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}}, \tilde{\mathbb{x}}', \tilde{\mathbb{I}}'\rangle \in \tilde{C}^f : \forall \mathrm{T} \in \mathbf{Thrd} : (pc^{\tilde{c}'}_\mathrm{T} = pc'_\mathrm{T} \wedge t'^{a\prime}_\mathrm{T} \in \gamma_t(\tilde{t}'^{a\prime}_\mathrm{T}))$.

If $\exists \mathrm{T} \in \mathbf{Thrd} : \textsc{stm}(\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}) \neq [\texttt{halt}]^{pc^{\tilde{c}}_{\mathrm{T}}} \wedge \forall \mathrm{T} \in \mathbf{Thrd} : (\textsc{stm}(\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}) \neq [\texttt{halt}]^{pc^{\tilde{c}}_{\mathrm{T}}} \Rightarrow \tilde{t}_{to} \lesssim_t \tilde{t}^{a}_{\mathrm{T}} \dot{+}_t \textsc{absTime}(\tilde{c}, \mathrm{T}))$, then it is easy to see that $\neg\textsc{isFinal}(\tilde{c})$, $\neg\textsc{isDeadlock}(\tilde{c})$ (since the program terminates in the concrete case) and $\textsc{isTimeout}(\tilde{c}, \tilde{t}_{to})$ (c.f., Algorithms 6.3 and 6.5), which means that $\tilde{c} \in \tilde{C}^{t}$. Thus, it must be that $\tilde{C}^{t} \neq \emptyset$.

2. Assume that $c \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c' \wedge (\textsc{cycle}(\mathbf{Thrd}^{c'}_{\texttt{lock}}, E^{c'}) \vee \exists \mathrm{T} \in \mathbf{Thrd} : \exists lck \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}, pc'_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc'_{\mathrm{T}}} \wedge \textsc{own}(\mathbb{l}'\ lck) \notin \{\bot_{thrd}, \mathrm{T}\} \wedge \textsc{stm}(\textsc{own}(\mathbb{l}'\ lck), pc'_{\textsc{own}(\mathbb{l}'\ lck)}) = [\texttt{halt}]^{pc'_{\textsc{own}(\mathbb{l}'\ lck)}}))$, where $\mathbf{Thrd}^{c'}_{\texttt{lock}} = \{\mathrm{T} \in \mathbf{Thrd} \mid \exists lck \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}, pc'_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc'_{\mathrm{T}}} \wedge \textsc{own}(\mathbb{l}'\ lck) \notin \{\bot_{thrd}, \mathrm{T}\})\}$ and $E^{c'} = \{(\mathrm{T}, \mathrm{T}') \mid \mathrm{T}, \mathrm{T}' \in \mathbf{Thrd}^{c'}_{\texttt{lock}} \wedge \exists lck \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}, pc'_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc'_{\mathrm{T}}} \wedge \textsc{own}(\mathbb{l}'\ lck) = \mathrm{T}')\}$ (remember that $\textsc{own}(\mathbb{l}'\ lck) \neq \bot_{thrd} \Rightarrow \textsc{own}(\mathbb{l}'\ lck) = locked$ since c is valid and $\xrightarrow[prg]{}$ preserves validity; c.f., Definition 4.4 and Lemma 4.5). Note that since all possible concrete transition sequences for each thread individually are safely approximated up until the timeout point and a deadlocked configuration is reached in the concrete case, there must be an abstract trace of transitions such that all configurations, $\tilde{c} \in \tilde{\mathbf{Conf}}$, on that trace are such that $\neg\textsc{isFinal}(\tilde{c})$ (c.f., Algorithm 6.3) and $\textsc{isValid}(\tilde{c}, \tilde{t}_{to})$ (c.f., Algorithm 6.6 and Lemma 6.7). It must also be that, eventually, a configuration, $\tilde{c}@\langle[\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \tilde{\mathbf{Conf}}$, will be derived (along the corresponding, over-approximating abstract trace of transitions) for which either $(\textsc{cycle}(\mathbf{Thrd}^{\tilde{c}}_{\texttt{lock}}, E^{\tilde{c}}) \vee \exists \mathrm{T} \in \mathbf{Thrd} : \exists lck \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc^{\tilde{c}}_{\mathrm{T}}} \wedge \tilde{\textsc{st}}(\tilde{\mathbb{l}}\ lck) = locked \wedge \tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck) \neq \bot_{thrd} \wedge \textsc{stm}(\tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck), pc^{\tilde{c}}_{\tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck)}) = [\texttt{halt}]^{pc^{\tilde{c}}_{\tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck)}}))$, where $\mathbf{Thrd}^{\tilde{c}}_{\texttt{lock}} = \{\mathrm{T} \in \mathbf{Thrd} \mid \exists lck \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc^{\tilde{c}}_{\mathrm{T}}} \wedge \tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck) \notin \{\bot_{thrd}, \mathrm{T}\} \wedge \tilde{\textsc{st}}(\tilde{\mathbb{l}}\ lck) = locked)\}$ and $E^{\tilde{c}} = \{(\mathrm{T}, \mathrm{T}') \mid \mathrm{T}, \mathrm{T}' \in \mathbf{Thrd}^{\tilde{c}}_{\texttt{lock}} \wedge \exists lck \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc^{\tilde{c}}_{\mathrm{T}}} \wedge \tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck) = \mathrm{T}')\}$, or $\forall \mathrm{T} \in \mathbf{Thrd} : (\textsc{stm}(\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}) \neq [\texttt{halt}]^{pc^{\tilde{c}}_{\mathrm{T}}} \Rightarrow \tilde{t}_{to} \lesssim_t \tilde{t}^{a}_{\mathrm{T}} \dot{+}_t \textsc{absTime}(\tilde{c}, \mathrm{T})) \wedge \neg(\textsc{cycle}(\mathbf{Thrd}^{\tilde{c}}_{\texttt{lock}}, E^{\tilde{c}}) \vee \exists \mathrm{T} \in \mathbf{Thrd} : \exists lck \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc^{\tilde{c}}_{\mathrm{T}}} \wedge \tilde{\textsc{st}}(\tilde{\mathbb{l}}\ lck) = locked \wedge \tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck) \neq \bot_{thrd} \wedge \textsc{stm}(\tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck), pc^{\tilde{c}}_{\tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck)}) = [\texttt{halt}]^{pc^{\tilde{c}}_{\tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck)}}))$.

If $\textsc{cycle}(\mathbf{Thrd}^{\tilde{c}}_{\texttt{lock}}, E^{\tilde{c}}) \vee \exists \mathrm{T} \in \mathbf{Thrd} : \exists lck \in \mathbf{Lck} : (\textsc{stm}(\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc^{\tilde{c}}_{\mathrm{T}}} \wedge \tilde{\textsc{st}}(\tilde{\mathbb{l}}\ lck) = locked \wedge \tilde{\textsc{own}}(\tilde{\mathbb{l}}\ lck) \neq \bot_{thrd} \wedge$

$\textsc{stm}(\textsc{oŵn}(\tilde{\mathbb{l}}\ lck), pc^{\tilde{c}}_{\textsc{oŵn}(\tilde{\mathbb{l}}\ lck)}) = [\texttt{halt}]^{pc^{\tilde{c}}_{\textsc{oŵn}(\tilde{\mathbb{l}}\ lck)}})$, then it is easy to see that $\neg\textsc{isFinal}(\tilde{c})$ and $\textsc{isDeadlock}(\tilde{c})$ (c.f., Algorithm 6.4 and Lemma 6.5), which means that $\tilde{C}^d \neq \emptyset$.

If $\forall \mathrm{T} \in \textbf{Thrd} : (\textsc{stm}(\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}) \neq [\texttt{halt}]^{pc^{\tilde{c}}_{\mathrm{T}}} \Rightarrow \tilde{t}_{to} \tilde{<}_t \tilde{t}^a_{\mathrm{T}} \tilde{+}_t \textsc{absTime}(\tilde{c}, \mathrm{T})) \wedge \neg(\textsc{cycle}(\textbf{Thrd}^{\tilde{c}}_{\texttt{lock}}, E^{\tilde{c}}) \vee \exists \mathrm{T} \in \textbf{Thrd} : \exists lck \in \textbf{Lck} : (\textsc{stm}(\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc^{\tilde{c}}_{\mathrm{T}}} \wedge \tilde{\textsc{st}}(\tilde{\mathbb{l}}\ lck) = locked \wedge \textsc{oŵn}(\tilde{\mathbb{l}}\ lck) \neq \bot_{thrd} \wedge \textsc{stm}(\textsc{oŵn}(\tilde{\mathbb{l}}\ lck), pc^{\tilde{c}}_{\textsc{oŵn}(\tilde{\mathbb{l}}\ lck)}) = [\texttt{halt}]^{pc^{\tilde{c}}_{\textsc{oŵn}(\tilde{\mathbb{l}}\ lck)}}))$, then it is easy to see that $\neg\textsc{isFinal}(\tilde{c})$, $\neg\textsc{isDeadlock}(\tilde{c})$ and $\textsc{isTimeout}(\tilde{c}, \tilde{t}_{to})$ (c.f., Algorithm 6.5 and Lemma 6.6), which means that $\tilde{C}^t \neq \emptyset$.

To prove the last part of the lemma, assume that $\tilde{C}^d \cup \tilde{C}^t = \emptyset$. Since $\forall\langle[\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^a_{\mathrm{T}}]_{\mathrm{T}\in\textbf{Thrd}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \tilde{C}^f : \forall \mathrm{T} \in \textbf{Thrd} : \textsc{stm}(\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}) = [\texttt{halt}]^{pc^{\tilde{c}}_{\mathrm{T}}}$ (c.f., Algorithm 6.3) and $\neg\textsc{isValid}(\tilde{c}, \tilde{t}_{to})$ only if $\tilde{c}$ can never lead to a configuration that might have a valid concrete counterpart (Lemma 6.7), it is easy to see that all concrete executions of the configurations in C will terminate since all possible concrete transition sequences are safely approximated. Further assume that $c \in C$ and $c'@\langle[\mathrm{T}, pc'_{\mathrm{T}}, \mathbb{r}'_{\mathrm{T}}, t^{a\prime}_{\mathrm{T}}]_{\mathrm{T}\in\textbf{Thrd}}, \mathbb{x}', \mathbb{l}'\rangle \in \textbf{Conf}$ are such that $c \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c' \wedge \forall \mathrm{T} \in \textbf{Thrd} : \textsc{stm}(\mathrm{T}, pc'_{\mathrm{T}}) = [\texttt{halt}]^{pc'_{\mathrm{T}}}$. Since $\tilde{C}^d \cup \tilde{C}^t = \emptyset$ and $\forall c \in C : \exists \tilde{c} \in \tilde{C} : c \in \gamma_{conf}(\tilde{c})$, it is easy to see that $\exists\langle[\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^a_{\mathrm{T}}]_{\mathrm{T}\in\textbf{Thrd}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \tilde{C}^f : (pc^{\tilde{c}}_{\mathrm{T}} = pc'_{\mathrm{T}} \wedge t^{a\prime}_{\mathrm{T}} \in \gamma_t(\tilde{t}^a_{\mathrm{T}}))$ since all possible concrete transition sequences are safely approximated.

This concludes the proof. ∎

NOTE. ABSEXE *has not been proven to terminate for all inputs. However, when it does terminate, it safely approximates the transition sequences for the corresponding concrete input set.*

One case for which ABSEXE *will not terminate is when some thread could execute an infinite amount of statements in zero amount of time; c.f., an infinite loop where all the statements of the loop could be executed without any progression of time.*

Algorithm 6.13 BCET/WCET Analysis

1: **function** ANALYSIS($\tilde{C}, \tilde{t}_{to}$)
2: $(\tilde{C}^f, \tilde{C}^d, \tilde{C}^t) \leftarrow$ ABSEXE($\tilde{C}, \tilde{t}_{to}$)
3: **if** $\tilde{C}^d \cup \tilde{C}^t \neq \emptyset$ **then**
4: **return** $(-\infty, \infty)$
5: **end if**
6: $BCET \leftarrow \infty$
7: $WCET \leftarrow -\infty$
8: **while** $\tilde{C}^f \neq \emptyset$ **do**
9: $\tilde{c}@\langle[\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \leftarrow$ CHOOSE($\tilde{C}^f$)
10: $\tilde{C}^f \leftarrow \tilde{C}^f \setminus \{\tilde{c}\}$
11: $BCET_{\tilde{c}} \leftarrow \max(\{\min(\gamma_t(\tilde{t}^a_{\mathrm{T}})) \mid \mathrm{T} \in \mathbf{Thrd}\})$
12: $WCET_{\tilde{c}} \leftarrow \max(\{\max(\gamma_t(\tilde{t}^a_{\mathrm{T}})) \mid \mathrm{T} \in \mathbf{Thrd}\})$
13: **if** $BCET > BCET_{\tilde{c}}$ **then**
14: $BCET \leftarrow BCET_{\tilde{c}}$
15: **end if**
16: **if** $WCET < WCET_{\tilde{c}}$ **then**
17: $WCET \leftarrow WCET_{\tilde{c}}$
18: **end if**
19: **end while**
20: **return** $(BCET, WCET)$
21: **end function**

6.2 Timing Analysis

The BCET and WCET (c.f., Definition 6.9) of a program, given an initial system state, is safely derived by ANALYSIS, which is defined in Algorithm 6.13 (Lemma 6.10), whenever it terminates.

Definition 6.9 (BCET and WCET):
The Best-Case Execution Time, BCET, and the Worst-Case Execution Time, WCET, of a given configuration, $\langle [\mathrm{T}, pc_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle \in \tilde{\mathbf{Conf}}$ *are defined as:*

$$\begin{cases} BCET = \max(\{\min(\gamma_t(\tilde{t}^a_\mathrm{T})) \mid \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}\}) \\ WCET = \max(\{\max(\gamma_t(\tilde{t}^a_\mathrm{T})) \mid \mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}\}) \end{cases}$$

□

Lemma 6.10 (Soundness of ANALYSIS):
Given the sets of valid concrete configurations $C \in \mathscr{P}(\mathbf{Conf})$ *(c.f., Definition 4.4) and abstract configurations* $\tilde{C} \in \mathscr{P}(\tilde{\mathbf{Conf}})$*, such that* $\forall c @ \langle [\mathrm{T}, pc_\mathrm{T}, \mathbb{r}_\mathrm{T}, t^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_1}, \mathbb{x}, \mathbb{l} \rangle \in C : (\forall \langle [\mathrm{T}, pc^{\tilde{c}}_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_2}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle \in \tilde{C} : (\mathbf{Thrd}_1 = \mathbf{Thrd}_2 = \mathbf{Thrd}) \wedge \exists \tilde{c} \in \tilde{C} : c \in \gamma_{conf}(\tilde{c})) \wedge |\mathbf{Thrd}| < \infty \wedge \forall \tilde{c} @ \langle [\mathrm{T}, pc^{\tilde{c}}_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle \in \tilde{C} : \forall lck \in \mathbf{Lck} : \min(\gamma_t(\tilde{\mathrm{DL}}(\tilde{\mathbb{l}}\ lck)) = -\infty))$*, and the times* $t_{to} \in \mathbf{Time}$ *and* $\tilde{t}_{to} \in \tilde{\mathbf{Time}}$*, such that* $t_{to} = \max(\gamma_t(\tilde{t}_{to}))$*,* $(BCET, WCET) @ \mathrm{ANALYSIS}(\tilde{C}, \tilde{t}_{to})$ *is such that*

$$\begin{aligned} &\forall c \in C : \forall c' @ \langle [\mathrm{T}, pc'_\mathrm{T}, \mathbb{r}'_\mathrm{T}, t^{a\prime}_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}}, \mathbb{x}', \mathbb{l}' \rangle \in \mathbf{Conf} : \\ &\quad (((c \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c' \wedge \forall \mathrm{T} \in \mathbf{Thrd} : \mathrm{STM}(\mathrm{T}, pc'_\mathrm{T}) = [\mathtt{halt}]^{\mathrm{T}, pc'_\mathrm{T}}) \Rightarrow \\ &\qquad\qquad \forall \mathrm{T} \in \mathbf{Thrd} : BCET \leq t^{a\prime}_\mathrm{T} \leq WCET) \wedge \\ &\quad (c \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c' \Rightarrow t^{a\prime}_\mathrm{T} \leq WCET)) \end{aligned}$$

given that the algorithm terminates. □

PROOF. Assume that the sets of valid concrete configurations $C \in \mathscr{P}(\mathbf{Conf})$ (c.f., Definition 4.4) and abstract configurations $\tilde{C} \in \mathscr{P}(\tilde{\mathbf{Conf}})$ are such that $\forall c @ \langle [\mathrm{T}, pc_\mathrm{T}, \mathbb{r}_\mathrm{T}, t^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_1}, \mathbb{x}, \mathbb{l} \rangle \in C : (\forall \langle [\mathrm{T}, pc^{\tilde{c}}_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}_2}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle \in \tilde{C} : (\mathbf{Thrd}_1 = \mathbf{Thrd}_2 = \mathbf{Thrd}) \wedge \exists \tilde{c} \in \tilde{C} : c \in \gamma_{conf}(\tilde{c})) \wedge |\mathbf{Thrd}| < \infty \wedge \forall \tilde{c} @ \langle [\mathrm{T}, pc^{\tilde{c}}_\mathrm{T}, \tilde{\mathbb{r}}_\mathrm{T}, \tilde{t}^a_\mathrm{T}]_{\mathrm{T}\in\mathbf{Thrd}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle \in \tilde{C} : \forall lck \in \mathbf{Lck} : \min(\gamma_t(\tilde{\mathrm{DL}}(\tilde{\mathbb{l}}\ lck)) = -\infty))$, that the times $t_{to} \in \mathbf{Time}$ and $\tilde{t}_{to} \in \tilde{\mathbf{Time}}$ are such that $t_{to} = \max(\gamma_t(\tilde{t}_{to}))$, and that $(BCET, WCET) = \mathrm{ANALYSIS}(\tilde{C}, \tilde{t}_{to})$.

Since $(BCET, WCET) = \text{ANALYSIS}(\tilde{C}, \tilde{t}_{to})$, it must be that $(\tilde{C}^f, \tilde{C}^d, \tilde{C}^t)$@ $\text{ABSEXE}(\tilde{C}, \tilde{t}_{to})$ terminates at some point and that

$$
\begin{array}{l}
\forall c \in C : \forall c' @ \langle [\mathrm{T}, pc'_{\mathrm{T}}, \mathbb{r}'_{\mathrm{T}}, t^{a\prime}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}}, \mathbb{x}', \mathbb{l}' \rangle \in \mathbf{Conf} : \\
\quad ((c \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c' \wedge \forall \mathrm{T} \in \mathbf{Thrd} : \text{STM}(\mathrm{T}, pc'_{\mathrm{T}}) = [\texttt{halt}]^{pc'_{\mathrm{T}}}) \Rightarrow \\
\quad\quad (\tilde{C}^t \neq \emptyset \vee \\
\quad\quad\quad \exists \tilde{c} @ \langle [\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle \in \tilde{C}^f : \forall \mathrm{T} \in \mathbf{Thrd} : \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad (pc^{\tilde{c}}_{\mathrm{T}} = pc'_{\mathrm{T}} \wedge t^{a\prime}_{\mathrm{T}} \in \gamma_t(\tilde{t}^{a}_{\mathrm{T}})))) \wedge \\
\forall c \in C : \forall c' @ \langle [\mathrm{T}, pc'_{\mathrm{T}}, \mathbb{r}'_{\mathrm{T}}, t^{a\prime}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}}, \mathbb{x}', \mathbb{l}' \rangle \in \mathbf{Conf} : \\
\quad ((c \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c' \wedge (\text{CYCLE}(\mathbf{Thrd}^{c'}_{\texttt{lock}}, E^{c'}) \vee \\
\quad\quad\quad\quad \exists \mathrm{T} \in \mathbf{Thrd} : \exists lck \in \mathbf{Lck} : \\
\quad\quad\quad\quad\quad (\text{STM}(\mathrm{T}, pc'_{\mathrm{T}}) = [\texttt{lock}\ lck]^{pc'_{\mathrm{T}}} \wedge \\
\quad\quad\quad\quad\quad\ \text{OWN}(\mathbb{l}'\ lck) \notin \{\bot_{thrd}, \mathrm{T}\} \wedge \\
\quad\quad\quad\quad\quad\ \text{STM}(\text{OWN}(\mathbb{l}'\ lck), pc'_{\text{OWN}(\mathbb{l}'\ lck)}) = \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad [\texttt{halt}]^{pc'_{\text{OWN}(\mathbb{l}'\ lck)}}))) \Rightarrow \\
\quad\quad (\tilde{C}^t \neq \emptyset \vee \tilde{C}^d \neq \emptyset))
\end{array}
$$

(Lemma 6.8). It is thus apparent that if $\tilde{C}^d \cup \tilde{C}^t \neq \emptyset$, there might exist an infinite transition sequence in the concrete case. However, it is easy to see that (as returned by the algorithm) $-\infty$ is a safe approximation of the BCET and that ∞ is a safe approximation of the WCET for all such (and all other) cases.

If $\tilde{C}^d \cup \tilde{C}^t = \emptyset$, then all concrete transition sequences are of finite length and $\forall c \in C : \forall c' @ \langle [\mathrm{T}, pc'_{\mathrm{T}}, \mathbb{r}'_{\mathrm{T}}, t^{a\prime}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}}, \mathbb{x}', \mathbb{l}' \rangle \in \mathbf{Conf} : ((c \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c' \wedge \forall \mathrm{T} \in \mathbf{Thrd} : \text{STM}(\mathrm{T}, pc'_{\mathrm{T}}) = [\texttt{halt}]^{pc'_{\mathrm{T}}}) \Rightarrow \exists \langle [\mathrm{T}, pc^{\tilde{c}}_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle \in \tilde{C}^f : (pc^{\tilde{c}}_{\mathrm{T}} = pc'_{\mathrm{T}} \wedge t^{a\prime}_{\mathrm{T}} \in \gamma_t(\tilde{t}^{a}_{\mathrm{T}})))$ (Lemma 6.8). Thus, since the structure of the algorithm trivially gives that the smallest possible estimation of the BCET, *BCET*, and the largest possible estimation of the WCET, *WCET*, among the derived final abstract configurations in $\tilde{C}^f$ are found (c.f., Definition 6.9), it must be that $\forall c \in C : \forall c' @ \langle [\mathrm{T}, pc'_{\mathrm{T}}, \mathbb{r}'_{\mathrm{T}}, t^{a\prime}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}}, \mathbb{x}', \mathbb{l}' \rangle \in \mathbf{Conf} : ((c \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c' \wedge \forall \mathrm{T} \in \mathbf{Thrd} : \text{STM}(\mathrm{T}, pc'_{\mathrm{T}}) = [\texttt{halt}]^{pc'_{\mathrm{T}}}) \Rightarrow \forall \mathrm{T} \in \mathbf{Thrd} : BCET \leq t^{a\prime}_{\mathrm{T}} \leq WCET)$. But, then it must also be that $\forall c \in C : \forall c' @ \langle [\mathrm{T}, pc'_{\mathrm{T}}, \mathbb{r}'_{\mathrm{T}}, t^{a\prime}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}}, \mathbb{x}', \mathbb{l}' \rangle \in \mathbf{Conf} : (c \xrightarrow[prg]{} \ldots \xrightarrow[prg]{} c' \Rightarrow \forall \mathrm{T} \in \mathbf{Thrd} : t^{a\prime}_{\mathrm{T}} \leq WCET)$ since time only moves forward (c.f., Assumption 5.50), which concludes the proof. ∎

Chapter 7

Examples

To clarify and explain the analysis defined in Chapters 5 and 6, this chapter instantiates it for some example PPL programs.

7.1 Communication

This case shows the recursive behavior of ABSEXE; i.e., how it peeks into the future to derive safe write histories for unsafe `load`-statements.

For the program, $\mathbf{Thrd} = \{\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3\}$, defined in Table 7.1, it is easy to see that $\mathbf{Reg}_{\mathrm{T}_1} = \{\mathtt{r}\}$, $\mathbf{Reg}_{\mathrm{T}_2} = \{\mathtt{r}\}$, $\mathbf{Reg}_{\mathrm{T}_3} = \{\mathtt{r}\}$, $\mathbf{Var} = \{\mathtt{x}, \mathtt{y}, \mathtt{z}\}$ and $\mathbf{Lck} = \emptyset$. Note that `r` represents local memory within each thread; i.e., the register-name `r` can refer to three different memory locations – what location it refers to depends on which thread is considered.

Assume that ABSTIME$(\tilde{c}, \mathrm{T})$, where $\tilde{c}@\langle [\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathtt{r}}_{\mathrm{T}}, \tilde{\imath}^{a}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}}, \tilde{\mathtt{x}}, \tilde{\mathbb{I}} \rangle \in$

$\mathrm{T}_1@(1, [\mathtt{load\ r\ from\ x}]^1; [\mathtt{store\ r\ to\ y}]^2; [\mathtt{halt}]^3)$
$\mathrm{T}_2@(2, [\mathtt{load\ r\ from\ y}]^1; [\mathtt{store\ r\ to\ z}]^2; [\mathtt{halt}]^3)$
$\mathrm{T}_3@(3, [\mathtt{if\ r <= 3\ goto\ 4}]^1; [\mathtt{store\ r\ to\ x}]^2; [\mathtt{skip}]^3; [\mathtt{halt}]^4)$

Table 7.1: Communicating threads – Program.

Cõnf and $\mathrm{T} \in \mathbf{Thrd}_{\tilde{c}}$, is such that for any $\tilde{c}$, it assumes the values described by the below table.

pc_{T}	T_1	T_2	T_3
1	$[1,5]$	$[2,6]$	$[1,4]$
2	$[1,3]$	$[2,3]$	$[3,4]$
3	$-$	$-$	$[3,3]$

Also assume that $\tilde{c}_0^0@\langle[\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}_{\mathrm{T}}^a]_{\mathrm{T}\in\mathbf{Thrd}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle$ is as described in Table 7.2. (Due to the semantics of the program, the parts of the states that are left out from the table are of no interest for this case study.)

Tables 7.2 and 7.3 collect all the configurations derived by ABSEXE($\{\tilde{c}_0^0\}$, $[-\infty,\infty]$) during the analysis described by ANALYSIS($\{\tilde{c}_0^0\}, [-\infty,\infty]$). A '$-$' indicates that the entry is not applicable to (i.e., not included in) the configuration. Figure 7.4 shows the order in which the configurations are derived; i.e., the relation between the derived configurations. In the figure, final configurations are circled and timed-out configurations are circled and marked with a 't'. To see how new recursive instances of ABSEXE are created, note that when $\mathbf{Thrd}_{\tilde{c}} = \{\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3\}$, then $\mathbf{Var_g} = \{\mathtt{x}, \mathtt{y}\}$; when $\mathbf{Thrd}_{\tilde{c}} = \{\mathrm{T}_1, \mathrm{T}_3\}$, then $\mathbf{Var_g} = \{\mathtt{x}\}$; and when $\mathbf{Thrd}_{\tilde{c}} = \{\mathrm{T}_2, \mathrm{T}_3\}$, then $\mathbf{Var_g} = \emptyset$.

It is apparent that ABSEXE($\{\tilde{c}_0^0\}, [-\infty,\infty]$) $= (\{\tilde{c}_{11}^0, \tilde{c}_{23}^0\}, \emptyset, \emptyset)$; i.e., $\tilde{c}_{11}^0$ and $\tilde{c}_{23}^0$ are final-state configurations and there are no deadlocked or timed-out configurations. According to Algorithm 6.13, it is thus easy to see that the estimated timing bounds are:

$$\begin{cases} \mathit{BCET} = \min(\{\max(\{\min(\gamma_t(\tilde{t}_{\mathrm{T}}^a)) \mid \mathrm{T} \in \mathbf{Thrd}\}) \mid \\ \qquad\qquad \langle[\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}_{\mathrm{T}}^a]_{\mathrm{T}\in\mathbf{Thrd}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \{\tilde{c}_{11}^0, \tilde{c}_{23}^0\}\}) = 4 \\ \mathit{WCET} = \max(\{\max(\{\max(\gamma_t(\tilde{t}_{\mathrm{T}}^a)) \mid \mathrm{T} \in \mathbf{Thrd}\}) \mid \\ \qquad\qquad \langle[\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}_{\mathrm{T}}^a]_{\mathrm{T}\in\mathbf{Thrd}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \{\tilde{c}_{11}^0, \tilde{c}_{23}^0\}\}) = 11 \end{cases}$$

$\tilde{c}$	pc_{T_1}	pc_{T_2}	pc_{T_3}	$\tilde{\mathbb{r}}_{\mathrm{T}_1}$ r	$\tilde{\mathbb{r}}_{\mathrm{T}_2}$ r	$\tilde{\mathbb{r}}_{\mathrm{T}_3}$ r	$\tilde{t}^a_{\mathrm{T}_1}$	$\tilde{t}^a_{\mathrm{T}_2}$	$\tilde{t}^a_{\mathrm{T}_3}$	($\tilde{\mathbb{x}}$ x) T_3	($\tilde{\mathbb{x}}$ y) T_1	($\tilde{\mathbb{x}}$ z) T_2
$\tilde{c}^0_0$	1	1	1	$[0,0]$	$[0,0]$	$[2,4]$	$[0,0]$	$[0,0]$	$[0,0]$	$\{([1,1],[0,0])\}$	$\{([5,5],[0,0])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t)\}$
$\tilde{c}^{11}_0$	–	1	1	–	$[0,0]$	$[2,4]$	–	$[0,0]$	$[0,0]$	$\{([1,1],[0,0])\}$	$\{([5,5],[0,0])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t)\}$
$\tilde{c}^{11}_{11}$	–	2	4	–	$[5,5]$	$[2,3]$	–	$[2,6]$	$[1,4]$	$\{([1,1],[0,0])\}$	$\{([5,5],[0,0])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t)\}$
$\tilde{c}^{11}_{12}$	–	3	4	–	$[5,5]$	$[2,3]$	–	$[4,9]$	$[1,4]$	$\{([1,1],[0,0])\}$	$\{([5,5],[0,0])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t),$ $([5,5],[4,9])\}$
$\tilde{c}^{11}_{21}$	–	2	2	–	$[5,5]$	$[4,4]$	–	$[2,6]$	$[1,4]$	$\{([1,1],[0,0])\}$	$\{([5,5],[0,0])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t)\}$
$\tilde{c}^{11}_{22}$	–	3	3	–	$[5,5]$	$[4,4]$	–	$[4,9]$	$[4,8]$	$\{([1,1],[0,0]),$ $([4,4],[4,8])\}$	$\{([5,5],[0,0])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t),$ $([5,5],[4,9])\}$
$\tilde{c}^{12}_0$	1	–	1	$[0,0]$	–	$[2,4]$	$[0,0]$	–	$[0,0]$	$\{([1,1],[0,0])\}$	$\{([5,5],[0,0])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t)\}$
$\tilde{c}^{22}_0$	–	–	1	–	–	$[2,4]$	–	–	$[0,0]$	$\{([1,1],[0,0])\}$	$\{([5,5],[0,0])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t)\}$
$\tilde{c}^{22}_{11}$	–	–	4	–	–	$[2,3]$	–	–	$[1,4]$	$\{([1,1],[0,0])\}$	$\{([5,5],[0,0])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t)\}$
$\tilde{c}^{22}_{21}$	–	–	2	–	–	$[4,4]$	–	–	$[1,4]$	$\{([1,1],[0,0])\}$	$\{([5,5],[0,0])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t)\}$
$\tilde{c}^{22}_{22}$	–	–	3	–	–	$[4,4]$	–	–	$[4,8]$	$\{([1,1],[0,0]),$ $([4,4],[4,8])\}$	$\{([5,5],[0,0])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t)\}$
$\tilde{c}$	pc_{T_1}	pc_{T_2}	pc_{T_3}	$\tilde{\mathbb{r}}_{\mathrm{T}_1}$ r	$\tilde{\mathbb{r}}_{\mathrm{T}_2}$ r	$\tilde{\mathbb{r}}_{\mathrm{T}_3}$ r	$\tilde{t}^a_{\mathrm{T}_1}$	$\tilde{t}^a_{\mathrm{T}_2}$	$\tilde{t}^a_{\mathrm{T}_3}$	($\tilde{\mathbb{x}}$ x) T_3	($\tilde{\mathbb{x}}$ y) T_1	($\tilde{\mathbb{x}}$ z) T_2

Table 7.2: Communicating threads – Configurations (First half).

$\tilde{c}$	pc_{T_1}	pc_{T_2}	pc_{T_3}	$\tilde{\mathbb{I}}_{\mathrm{T}_1}$ r	$\tilde{\mathbb{I}}_{\mathrm{T}_2}$ r	$\tilde{\mathbb{I}}_{\mathrm{T}_3}$ r	$\tilde{t}^a_{\mathrm{T}_1}$	$\tilde{t}^a_{\mathrm{T}_2}$	$\tilde{t}^a_{\mathrm{T}_3}$	($\tilde{\mathbb{x}}$ x) T_3	($\tilde{\mathbb{x}}$ y) T_1	($\tilde{\mathbb{x}}$ z) T_2
$\tilde{c}^{12}_1$	2	–	1	$[1,4]$	–	$[2,4]$	$[1,5]$	–	$[0,0]$	$\{([1,1],[0,0])\}$	$\{([5,5],[0,0])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t)\}$
$\tilde{c}^{12}_{11}$	3	–	4	$[1,4]$	–	$[2,3]$	$[2,8]$	–	$[1,4]$	$\{([1,1],[0,0])\}$	$\{([5,5],[0,0]),$ $([1,4],[2,8])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t)\}$
$\tilde{c}^{12}_{21}$	3	–	2	$[1,4]$	–	$[4,4]$	$[2,8]$	–	$[1,4]$	$\{([1,1],[0,0])\}$	$\{([5,5],[0,0]),$ $([1,4],[2,8])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t)\}$
$\tilde{c}^{12}_{22}$	3	–	3	$[1,4]$	–	$[4,4]$	$[2,8]$	–	$[4,8]$	$\{([1,1],[0,0]),$ $([4,4],[4,8])\}$	$\{([5,5],[0,0]),$ $([1,4],[2,8])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t)\}$
$\tilde{c}^{0}_1$	2	2	1	$[1,4]$	$[1,5]$	$[2,4]$	$[1,5]$	$[2,6]$	$[0,0]$	$\{([1,1],[0,0])\}$	$\{([5,5],[0,0])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t)\}$
$\tilde{c}^{0}_{11}$	3	3	4	$[1,4]$	$[1,5]$	$[2,3]$	$[2,8]$	$[4,9]$	$[1,4]$	$\{([1,1],[0,0])\}$	$\{([5,5],[0,0]),$ $([1,4],[2,8])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t),$ $([1,5],[4,9])\}$
$\tilde{c}^{0}_{21}$	3	3	2	$[1,4]$	$[1,5]$	$[4,4]$	$[2,8]$	$[4,9]$	$[1,4]$	$\{([1,1],[0,0])\}$	$\{([5,5],[0,0]),$ $([1,4],[2,8])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t),$ $([1,5],[4,9])\}$
$\tilde{c}^{0}_{22}$	3	3	3	$[1,4]$	$[1,5]$	$[4,4]$	$[2,8]$	$[4,9]$	$[4,8]$	$\{([1,1],[0,0]),$ $([4,4],[4,8])\}$	$\{([5,5],[0,0]),$ $([1,4],[2,8])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t),$ $([1,5],[4,9])\}$
$\tilde{c}^{0}_{23}$	3	3	4	$[1,4]$	$[1,5]$	$[4,4]$	$[2,8]$	$[4,9]$	$[7,11]$	$\{([1,1],[0,0]),$ $([4,4],[4,8])\}$	$\{([5,5],[0,0]),$ $([1,4],[2,8])\}$	$\{(\tilde{\perp}_{val},\tilde{\perp}_t),$ $([1,5],[4,9])\}$
$\tilde{c}$	pc_{T_1}	pc_{T_2}	pc_{T_3}	$\tilde{\mathbb{I}}_{\mathrm{T}_1}$ r	$\tilde{\mathbb{I}}_{\mathrm{T}_2}$ r	$\tilde{\mathbb{I}}_{\mathrm{T}_3}$ r	$\tilde{t}^a_{\mathrm{T}_1}$	$\tilde{t}^a_{\mathrm{T}_2}$	$\tilde{t}^a_{\mathrm{T}_3}$	($\tilde{\mathbb{x}}$ x) T_3	($\tilde{\mathbb{x}}$ y) T_1	($\tilde{\mathbb{x}}$ z) T_2

Table 7.3: Communicating threads – Configurations (Second half).

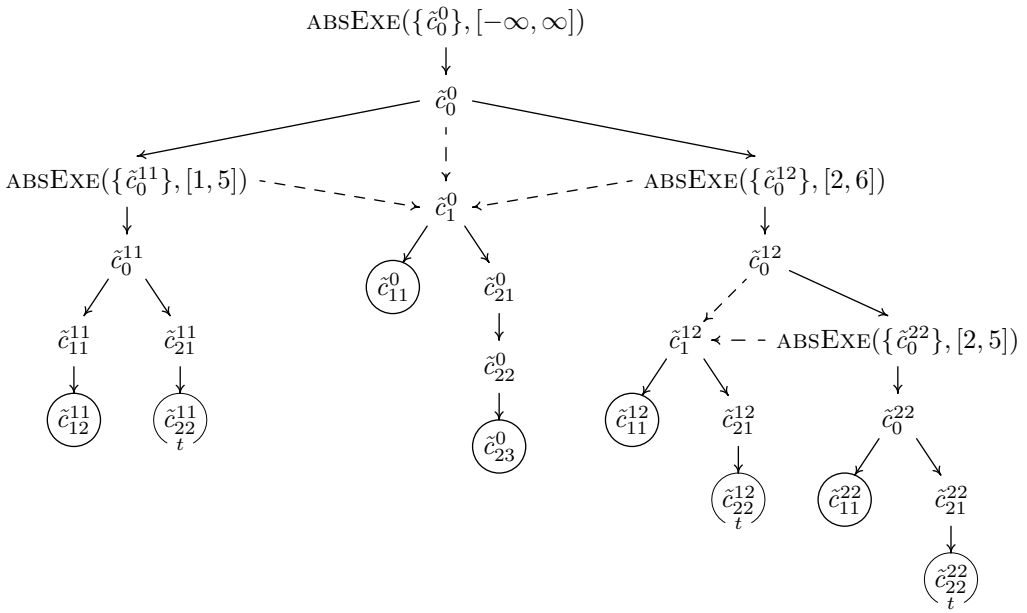


Figure 7.4: Communicating threads – Configuration relations.

$\mathrm{T}_1@(1,[\texttt{lock la}]^1;[\texttt{lock lb}]^2;[\texttt{unlock la}]^3;[\texttt{unlock lb}]^4;[\texttt{halt}]^5)$

$\mathrm{T}_2@(2,[\texttt{lock la}]^1;[\texttt{lock lb}]^2;[\texttt{halt}]^3)$

Table 7.5: Synchronization (Deadlock) – Program.

7.2 Synchronization – Deadlocks

This case shows how ABSEXE identifies deadlocked configurations and how it discontinues deadlocked configurations that lack concrete counterparts.

For the program, $\mathbf{Thrd} = \{\mathrm{T}_1,\mathrm{T}_2\}$, defined in Table 7.5, it is easy to see that $\mathbf{Reg}_{\mathrm{T}_1} = \emptyset$, $\mathbf{Reg}_{\mathrm{T}_2} = \emptyset$, $\mathbf{Var} = \emptyset$ and $\mathbf{Lck} = \{\texttt{la},\texttt{lb}\}$. Assume that $\textsc{absTime}(\tilde{c},\mathrm{T})$, where $\tilde{c}@\langle[\mathrm{T},pc_{\mathrm{T}},\tilde{\mathbb{r}}_{\mathrm{T}},\tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}},\tilde{\mathbb{x}},\tilde{\mathbb{l}}\rangle \in \mathbf{C\tilde{o}nf}$ and $\mathrm{T}\in\mathbf{Thrd}$, is such that for any $\tilde{c}$, it assumes the values described by the below table.

pc_{T}	T_1	T_2
1	$[2,2]$	$[1,2]$
2	$[1,2]$	$[1,2]$
3	$[1,1]$	–
4	$[1,1]$	–

Also assume that $\tilde{c}^0_0@\langle[\mathrm{T},pc_{\mathrm{T}},\tilde{\mathbb{r}}_{\mathrm{T}},\tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}},\tilde{\mathbb{x}},\tilde{\mathbb{l}}\rangle$ is as described in Table 7.6. (Due to the semantics of the program, the parts of the states that are left out from the table are of no interest for this case study.)

Table 7.6 collects all the configurations derived by $\textsc{absExe}(\{\tilde{c}^0_0\},[-\infty,\infty])$ during the analysis described by $\textsc{analysis}(\{\tilde{c}^0_0\},[-\infty,\infty])$. Figure 7.7 shows the order in which the configurations are derived; i.e., the relation between the derived configurations. In the figure, final configurations are circled, deadlocked configurations are circled and marked with a '*d*' and discontinued configurations are crossed out. Note that $\tilde{c}^4_2$ occurs since T_2 has been waiting to acquire `la` and is now assigned it, which means that T_2's accumulated execution time will be advanced to simulate the spin-lock waiting of the concrete semantics (c.f., the discussion in the proof of Lemma 5.57).

It is apparent that $\textsc{absExe}(\{\tilde{c}^0_0\},[-\infty,\infty]) = (\{\tilde{c}^7_2\},\{\tilde{c}^2_4\},\emptyset)$; i.e., $\tilde{c}^7_2$ is a final-state configuration, $\tilde{c}^2_4$ is a deadlocked configuration, and there are no timed-out configurations.

$\tilde{c}$	pc_{T_1}	pc_{T_2}	$\tilde{t}^a_{\mathrm{T}_1}$	$\tilde{t}^a_{\mathrm{T}_2}$	$\tilde{\mathbb{I}}$ `la`	$\tilde{\mathbb{I}}$ `lb`
$\tilde{c}^0_0$	1	1	$[0,0]$	$[0,0]$	$(\mathit{unlocked}, \bot_{\mathit{thrd}}, \tilde{\bot}_t, \bot_{\mathit{thrd}}, \tilde{\bot}_t)$	$(\mathit{unlocked}, \bot_{\mathit{thrd}}, \tilde{\bot}_t, \bot_{\mathit{thrd}}, \tilde{\bot}_t)$
$\tilde{c}^1_1$	2	1	$[2,2]$	$[0,0]$	$(\mathit{locked}, \mathrm{T}_1, [-\infty,2], \bot_{\mathit{thrd}}, \tilde{\bot}_t)$	$(\mathit{unlocked}, \bot_{\mathit{thrd}}, \tilde{\bot}_t, \bot_{\mathit{thrd}}, \tilde{\bot}_t)$
$\tilde{c}^2_1$	2	1	$[2,2]$	$[0,0]$	$(\mathit{locked}, \mathrm{T}_1, [-\infty,2], \bot_{\mathit{thrd}}, \tilde{\bot}_t)$	$(\mathit{unlocked}, \mathrm{T}_2, [-\infty,4], \bot_{\mathit{thrd}}, \tilde{\bot}_t)$
$\tilde{c}^2_2$	3	1	$[3,4]$	$[0,0]$	$(\mathit{locked}, \mathrm{T}_1, [-\infty,2], \bot_{\mathit{thrd}}, \tilde{\bot}_t)$	$(\mathit{locked}, \mathrm{T}_1, [-\infty,4], \bot_{\mathit{thrd}}, \tilde{\bot}_t)$
$\tilde{c}^3_1$	4	1	$[4,5]$	$[0,0]$	$(\mathit{unlocked}, \bot_{\mathit{thrd}}, [-\infty,2], \mathrm{T}_1, [4,5])$	$(\mathit{locked}, \mathrm{T}_1, [-\infty,4], \bot_{\mathit{thrd}}, \tilde{\bot}_t)$
$\tilde{c}^4_1$	5	1	$[5,6]$	$[0,0]$	$(\mathit{unlocked}, \mathrm{T}_1, [-\infty,12], \mathrm{T}_1, [4,5])$	$(\mathit{unlocked}, \bot_{\mathit{thrd}}, [-\infty,4], \mathrm{T}_1, [5,6])$
$\tilde{c}^4_2$	4	1	$[4,5]$	$[1,2]$	$(\mathit{unlocked}, \mathrm{T}_2, [-\infty,12], \mathrm{T}_1, [4,5])$	$(\mathit{locked}, \mathrm{T}_1, [-\infty,4], \bot_{\mathit{thrd}}, \tilde{\bot}_t)$
$\tilde{c}^5_1$	4	2	$[4,5]$	$[4,12]$	$(\mathit{locked}, \mathrm{T}_2, [-\infty,12], \mathrm{T}_1, [4,5])$	$(\mathit{locked}, \mathrm{T}_1, [-\infty,4], \bot_{\mathit{thrd}}, \tilde{\bot}_t)$
$\tilde{c}^6_1$	5	2	$[5,6]$	$[4,12]$	$(\mathit{locked}, \mathrm{T}_2, [-\infty,12], \mathrm{T}_1, [4,5])$	$(\mathit{unlocked}, \bot_{\mathit{thrd}}, [-\infty,4], \mathrm{T}_1, [5,6])$
$\tilde{c}^7_1$	5	2	$[5,6]$	$[4,12]$	$(\mathit{locked}, \mathrm{T}_2, [-\infty,12], \mathrm{T}_1, [4,5])$	$(\mathit{unlocked}, \mathrm{T}_1, [-\infty,18], \mathrm{T}_1, [5,6])$
$\tilde{c}^7_2$	5	3	$[5,6]$	$[5,18]$	$(\mathit{locked}, \mathrm{T}_2, [-\infty,12], \mathrm{T}_1, [4,5])$	$(\mathit{locked}, \mathrm{T}_2, [-\infty,18], \mathrm{T}_1, [5,6])$
$\tilde{c}^1_2$	1	2	$[0,0]$	$[1,2]$	$(\mathit{locked}, \mathrm{T}_2, [-\infty,2], \bot_{\mathit{thrd}}, \tilde{\bot}_t)$	$(\mathit{unlocked}, \bot_{\mathit{thrd}}, \tilde{\bot}_t, \bot_{\mathit{thrd}}, \tilde{\bot}_t)$
$\tilde{c}^2_3$	1	2	$[0,0]$	$[1,2]$	$(\mathit{locked}, \mathrm{T}_2, [-\infty,2], \bot_{\mathit{thrd}}, \tilde{\bot}_t)$	$(\mathit{unlocked}, \mathrm{T}_1, [-\infty,4], \bot_{\mathit{thrd}}, \tilde{\bot}_t)$
$\tilde{c}^2_4$	1	3	$[0,0]$	$[2,4]$	$(\mathit{locked}, \mathrm{T}_2, [-\infty,2], \bot_{\mathit{thrd}}, \tilde{\bot}_t)$	$(\mathit{locked}, \mathrm{T}_2, [-\infty,4], \bot_{\mathit{thrd}}, \tilde{\bot}_t)$
$\tilde{c}$	pc_{T_1}	pc_{T_2}	$\tilde{t}^a_{\mathrm{T}_1}$	$\tilde{t}^a_{\mathrm{T}_2}$	$\tilde{\mathbb{I}}$ `la`	$\tilde{\mathbb{I}}$ `lb`

Table 7.6: Synchronization (Deadlock) – Configurations.

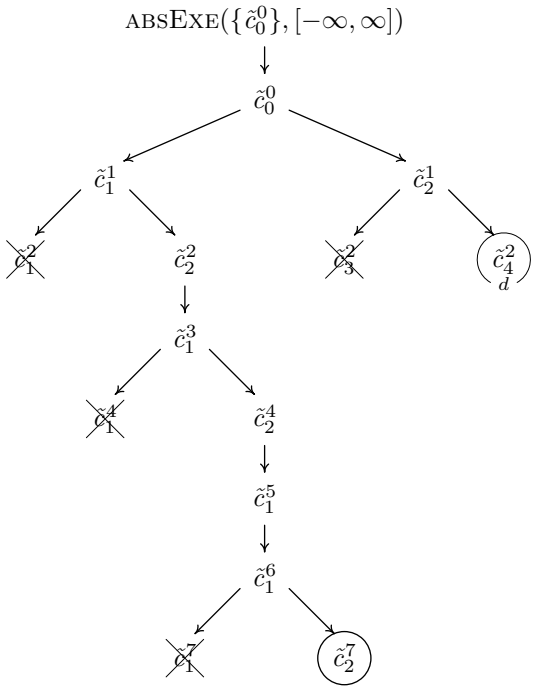


Figure 7.7: Synchronization (Deadlock) – Configuration relations.

According to Algorithm 6.13, it is thus easy to see that the estimated timing bounds are:

$$\begin{cases} BCET = -\infty \\ WCET = \infty \end{cases}$$

$\mathrm{T}_1@(1,[\texttt{lock l}]^1;[\texttt{halt}]^2)$

$\mathrm{T}_2@(2,[\texttt{lock l}]^1;[\texttt{halt}]^2)$

Table 7.8: Synchronization (Deadline miss) – Program.

7.3 Synchronization – Deadline Miss

This case illustrates how the analysis discontinues configurations for which an assigned lock owner does not acquire the lock in time. It also illustrates how the analysis detects deadlocks.

For the program, $\mathbf{Thrd} = \{\mathrm{T}_1, \mathrm{T}_2\}$, defined in Table 7.8, it is easy to see that $\mathbf{Reg}_{\mathrm{T}_1} = \emptyset$, $\mathbf{Reg}_{\mathrm{T}_2} = \emptyset$, $\mathbf{Var} = \emptyset$ and $\mathbf{Lck} = \{\mathtt{l}\}$. Assume that $\textsc{absTime}(\tilde{c}, \mathrm{T})$, where $\tilde{c}@\langle[\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{l}^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle \in \tilde{\mathbf{Conf}}$ and $\mathrm{T} \in \mathbf{Thrd}$, is such that for any $\tilde{c}$, it assumes the values described by the below table.

pc_{T}	T_1	T_2
1	$[5,5]$	$[10,10]$

Also assume that $\tilde{c}_0@\langle[\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{l}^a_{\mathrm{T}}]_{\mathrm{T}\in\mathbf{Thrd}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}}\rangle$ is as described in Table 7.9. (Due to the semantics of the program, the parts of the states that are left out from the table are of no interest for this case study.)

Table 7.9 collects all the configurations derived by $\textsc{absExe}(\{\tilde{c}_0\}, [-\infty, \infty])$ during the analysis described by $\textsc{analysis}(\{\tilde{c}_0\}, [-\infty, \infty])$. Figure 7.10 shows the order in which the configurations are derived; i.e., the relation between the derived configurations. In the figure, deadlocked configurations are circled and marked with a '*d*' and discontinued configurations are crossed out. It is apparent that $\textsc{absExe}(\{\tilde{c}_0\}, [-\infty, \infty]) = (\emptyset, \{\tilde{c}_1\}, \emptyset)$; i.e., there are no final-state or timed-out configurations, and $\tilde{c}_1$ is a deadlocked configuration. According to Algorithm 6.13, it is thus easy to see that the estimated timing bounds are:

$$\begin{cases} BCET = -\infty \\ WCET = \infty \end{cases}$$

$\tilde{c}$	pc_{T_1}	pc_{T_2}	$\tilde{t}^{a}_{\mathrm{T}_1}$	$\tilde{t}^{a}_{\mathrm{T}_2}$	$\tilde{\mathbb{I}}\,\mathtt{l}$
$\tilde{c}_0$	1	1	$[0,0]$	$[0,0]$	$(\mathit{unlocked}, \bot_{\mathit{thrd}}, \tilde{\bot}_t, \bot_{\mathit{thrd}}, \tilde{\bot}_t)$
$\tilde{c}_1$	2	1	$[5,5]$	$[0,0]$	$(\mathit{locked}, \mathrm{T}_1, [-\infty,5], \bot_{\mathit{thrd}}, \tilde{\bot}_t)$
$\tilde{c}_2$	1	1	$[0,0]$	$[10,10]$	$(\mathit{unlocked}, \mathrm{T}_2, [-\infty,5], \bot_{\mathit{thrd}}, \tilde{\bot}_t)$
$\tilde{c}$	pc_{T_1}	pc_{T_2}	$\tilde{t}^{a}_{\mathrm{T}_1}$	$\tilde{t}^{a}_{\mathrm{T}_2}$	$\tilde{\mathbb{I}}\,\mathtt{l}$

Table 7.9: Synchronization (Deadline miss) – Configurations.

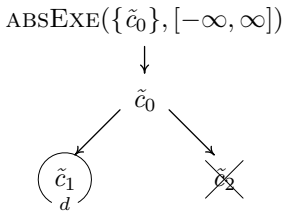


Figure 7.10: Synchronization (Deadline miss) – Configuration relations.

Chapter 8

Conclusions

In this chapter, some distinguishing properties of the defined analysis will be discussed. Feedback on the Research Questions and issues to be further considered and investigated will also be given.

8.1 The Underlying Architecture

The analysis is defined for an arbitrary underlying architecture (that is, however, restricted to the constraints in Assumptions 4.1 and 4.3). The actual underlying system could be an operating system as well as raw hardware as long as both thread-private and globally shared memory, and some form of synchronization primitive, correlating to the description given in the beginning of Chapter 4 is provided. The assumed architecture should be fairly realistic since any mature operating system and any common (single- or multi-core) CPU provides the described features at some abstraction level. For example, any Real-Time operating system should provide spin-locks for thread synchronization and any CPU instruction set should provide the ability to lock the system bus to provide atomic execution of a set of machine operations (since one single instruction of the instruction set often is mapped to a set of machine instructions).

The `lock`- and `unlock`-statements could be used to model the `LOCK` prefix in the x86 instruction set. This prefix is used for asserting atomic execution of an instruction [41, 81]. The `lock`- and `unlock`-statements also trivially correspond to higher level spin-locking primitives, such as those provided by

the POSIX thread library [10, 39].

Many of the principles applied in the analysis presented in Chapters 5 and 6 to solve the problems arising from abstracting time using intervals are also applicable to analysis of systems with distributed address spaces. If considering processes on one and the same CPU, then communication between these processes is often implemented using a memory buffer which is then to be considered as shared memory. This means that the same principles as those presented in this thesis would be applicable to such an analysis. If communication is performed using, for example, message passing and "Any"-communication is available (i.e., several processes could send a message to a given receiving process and/or several processes could receive a message sent from a given process), this would also require some form of prediction of what values could be transferred between processes.

The necessity of allowing $\text{TIME}(c, \text{T}) = 0$ for some configuration, $c \in \mathbf{Conf}$, and thread, $\text{T} \in \mathbf{Thrd}$, is apparent when considering the following case. If mutual exclusion is inherent in some instruction of the modeled instruction set, for example, `store`, then the `lock`- and `unlock`-statements could be regarded as macros without timing that should encapsulate all `store`-statements in a program.

PPL is designed to bring the focus of the analysis to thread synchronization and global data flow. The method presented in this thesis might have to be extended in order to cover all the aspects of a real instruction set, such as those of for example the ARM or PowerPC architectures [5, 42]. This will be further investigated.

If limiting the register (and variable) sizes the architecture would become more realistic. However, wrap-around effects could render loops non-terminating in the abstract case even if this would not concretely occur. See Section 8.3 for a discussion on more non-terminating cases.

8.2 Algorithmic Structure & Complexity

The analysis presented in Chapters 5 and 6 is based on synchronously advancing the threads of a program between their respective program points while keeping the threads fairly synchronized in time (c.f., Algorithm 6.1 and Tables 5.5 and 5.6). The advantage of this approach (i.e., abstracting time using intervals) in conjunction with the defined domain for variable states (c.f., Section 5.5) is that a relatively high precision is achieved. And, when $|\mathbf{Thrd}| = 1$, the analysis result will be equivalent to that of the corresponding sequential ana-

lysis (c.f., [28]). Another advantage is that the time-complexity of the analysis is more dependent on the number of program points in each thread than on the timing behavior of the program, compared to stepping through strict timing events, like in the concrete semantics.

Keeping the threads fairly synchronized in the analysis is also an advantage when considering its memory-complexity. Keeping the threads synchronized means that the write history for any thread on any variable will always be as small as possible since writes become outdated after a minimal amount of steps in the analysis and are then trimmed away from the history. In other words, the write history for any thread on any variable will never be larger than absolutely necessary.

One of the main disadvantages with keeping write history for each thread on each variable (which is expected to be necessary in order to keep the over-approximations at a reasonable level) is that the history must be trimmed. Trimming is an advantage for the memory-complexity as discussed above, but could be a serious disadvantage for the time-complexity if the analyzed program consists of many variables and many write-intensive threads.

The definition of the abstract state for locks contains some concrete parts (e.g., the owners of the locks). This is necessary since too much precision would be lost, and the timing approximations would become useless (i.e., too over-approximate), otherwise. However, this is very bad from a complexity point of view. The result of not abstracting some parts of a state is that (at least) all the concrete counterparts must be evaluated. Any reasonably precise abstractions of the parts of the lock states that are currently kept concrete have not been found. In case of serious complexity issues (which are very likely to occur), candidate domains for such abstractions must be further investigated.

It should be apparent that a given (abstract) configuration could result in two or more configurations for each thread issuing an `if`- or a `lock`-statement in a transition (c.f., Tables 5.5 and 5.6). Merging of configurations could be performed to reduce the complexity of the analysis. Using the Control Flow Graph (CFG) of the program, suitable merge-points within each thread can be found [25]. Typically, such points have multiple incoming edges. However, even if adding merging to the analysis, it will most probably happen very infrequently (if ever at all). This is since all the concrete parts (i.e., the program counters, lock owners, etc.) must be equal between the configurations to merge. As discussed above, domains for abstraction of the concrete parts of the configuration might have to be further investigated in case of serious complexity issues. Abstracting more parts of the configurations would also increase the possibility that merging could be more frequently performed. Although, the

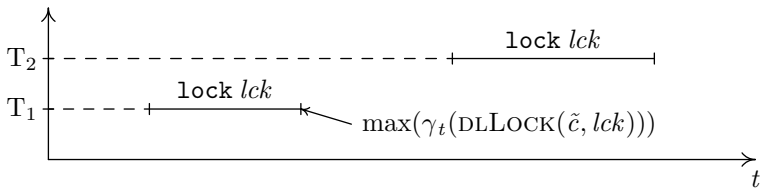


Figure 8.1: Lock owner assignments based on $\tilde{c} \in \tilde{\mathbf{Conf}}$ resulting in one valid and one invalid configuration.

derived timing approximations could become very pessimistic.

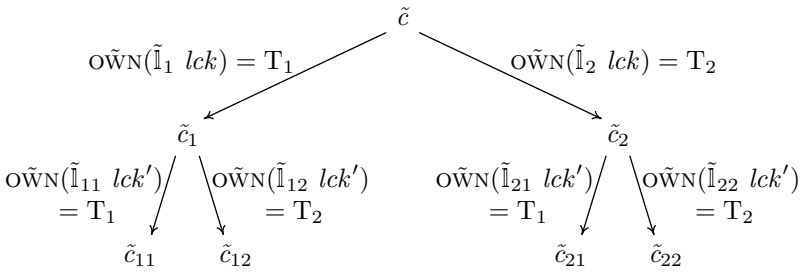
It is very important to note that several configurations that lack valid concrete counterparts (c.f., Definition 4.4) are added to the work-list for several situations. One such situation is when one sole thread issues `lock` *lck* for some free lock, $lck \in \mathbf{Lck}$, in a transition. A unique transition (i.e., resulting configuration) is possible for each thread that might issue `lock` *lck* somewhere in the program. A new configuration for each such thread, where the given thread is the new owner of *lck*, will thus be derived. Consider the situation depicted in Figure 8.1 (c.f., the case study in Section 7.3). T_1 is obviously the thread issuing `lock` *lck* first in any considered case. However, two new configurations are derived on the transition; one where T_1 is the new owner of *lck* and one where T_2 is the new owner of *lck*. Obviously, only the configuration for which T_1 is the owner of *lck* has valid concrete counterparts since T_2 will not acquire *lck* before some other thread (i.e., T_1) is guaranteed to have acquired *lck*. Thus, the case that T_2 is the new owner of *lck* will be discontinued since the lock is not acquired by T_2 before the deadline expires (c.f., Algorithm 6.6).

Another such situation will result for the program described in Figure 8.2a, assuming that the timing of the first `lock`-statement in the two threads overlap (c.f., the case study in Section 7.2). (Note that the given code is guaranteed not to deadlock, provided that both threads eventually release the two locks again.) Assume that the program is described by $\tilde{c}$@ $\langle [\mathrm{T}, pc_{\mathrm{T}}, \tilde{\mathbb{r}}_{\mathrm{T}}, \tilde{t}^{a}_{\mathrm{T}}]_{\mathrm{T} \in \mathbf{Thrd}}, \tilde{\mathbb{x}}, \tilde{\mathbb{l}} \rangle \in \tilde{\mathbf{Conf}}$ and that $pc_{\mathrm{T}_1} = pc_{\mathrm{T}_2} = 1$. The resulting lock-owner assignments (i.e., configurations) are given in Figure 8.2b. Obviously, only $\tilde{c}_{11}$ and $\tilde{c}_{22}$ have valid concrete counterparts. $\tilde{c}_{12}$ and $\tilde{c}_{21}$ will be discontinued (i.e., removed from the work-list) since there is a cycle in the dependency graph containing at least one lock (here, that lock is lck') that has the state *unlocked* (c.f., Algorithm 6.6). If $\tilde{c}_{12}$ and $\tilde{c}_{21}$ were not discontinued, the analysis would itself deadlock.

It is easy to see that all these complexity hazards really explode when com-

$\mathrm{T}_1 : [\texttt{lock}\ lck]^1\texttt{;}\,[\texttt{lock}\ lck']^2\texttt{;}\ \ldots$
$\mathrm{T}_2 : [\texttt{lock}\ lck]^1\texttt{;}\,[\texttt{lock}\ lck']^2\texttt{;}\ \ldots$

(a) The program described by $\tilde{c}$.



(b) Resulting configurations.

Figure 8.2: Lock owner assignments based on $\tilde{c} \in \tilde{\mathbf{Conf}}$ resulting in two valid and two invalid (i.e., falsely deadlocked) configurations.

bined (e.g., when they occur in different threads for a single transition). This could render the analysis extremely complex. A good thing to notice, though, is that a well structured parallel program will be less complex to analyze [53]. The threads in a well structured program typically work as much as possible on local data and do not synchronize more than is absolutely necessary.

Another good thing to notice is that the complexity is lowered by keeping a high precision in the calculation of the accumulated execution time for threads issuing `lock`-statements. Since $\tilde{\mathbf{Time}} = \mathbf{Intv}$, a high precision in this calculation will give a narrow accumulated execution time. This will lead to that a minimum number of states need to be explored since the timing of individual threads will not overlap more than necessary. Of course, this part of the complexity is also dependent on the precision of ABSTIME; i.e., the accuracy in the model of the underlying architecture.

8.3 Non-terminating Transition Sequences

As previously discussed, ISDEADLOCK catches some configurations that will never reach the final state (c.f., Algorithm 6.4). However, it is not guaranteed to identify all such configurations. This means that the analysis could actually deadlock for some cases that ISDEADLOCK misses to identify as never reach-

ing the final state. The corresponding can be said if ISVALID wrongly identifies a configuration as valid (c.f., Algorithm 6.6).

Infinite loops are recognized by ISTIMEOUT$(\tilde{c}, \tilde{t}_{to})$, given that time moves forward and the timeout is finite; i.e., it cannot be that $0 \in$ ABSTIME$(\tilde{c}', \mathrm{T})$ for all $\tilde{c}' \in$ **Cõnf** occurring in the loop in T and $\max(\gamma_t(\tilde{t}_{to})) = \infty$. If ABSTIME includes 0 for all statements of an infinite loop in some thread, then the algorithm will not terminate.

To avoid part of this problem, another timeout variable could be added to the analysis. This timeout could be used to identify that the upper bound of a single thread's accumulated execution time has reached a limit. However, this does not resolve the case that, for all $\tilde{c}' \in$ **Cõnf** in the loop, ABSTIME$(\tilde{c}', \mathrm{T}) = [0,0]$.

To address this case, a transition counter could be used. There could be one counter for each thread individually and/or one counter for all threads combined. The counter(s) could either count all transitions or only transitions that are consecutively done in $[0,0]$ amount of time, depending on whether a second timeout is used. When the counter reaches a specific limit, the configuration could be considered to be timed out, which means that the corresponding transition sequence could be of infinite length.

Even if all concrete transition sequences given some initial configuration terminate, all abstract transition sequences resulting from the corresponding abstract initial configuration are not guaranteed to terminate. This is due to over-approximations inherent in the abstract interpretation of the PPL semantics. Thus, all the complications discussed above can occur in the abstract case even if they do not in the concrete case.

8.4 The Research Questions

Question 1: "*What are the distinguishing features of a parallel computer system (i.e., the hardware and software combination) that must be taken into account in a timing analysis on the code level?*"

The most important aspects found are some means of communication and synchronization between the parallel entities. In this thesis, only the software (i.e., code) level is considered: shared memory is used to represent the communication medium; locks that can be acquired in a mutually exclusive manner using spin-locking are used to represent the synchronization medium; and threads are used to represent the parallel entities.

Question 2: "*How can a parallel computer system be analyzed to derive safe and tight estimations on its timing bounds?*"

It has been shown that Abstract Interpretation is a suitable technique for deriving safe timing bound estimates for a given program and timing model. The resulting tightness of the estimates depends both on the precision of the used abstract domains, the precision of the timing analysis itself and the precision of the timing model. Further evaluation, preferably based on an implementation of the analysis, must be performed before the tightness of the approach used in this thesis can be commented upon.

Question 3: "*How can analysis termination be guaranteed?*"

Some techniques to increase the termination-probability have been already incorporated into the analysis and were discussed in the previous sections; one such technique is the discontinuation of configurations that lack concrete counterparts. Further techniques should be investigated and could most probably be derived based on an implementation and evaluation of the analysis. One such example, as discussed above, is to include a second timeout variable in the analysis. Another example, as also discussed above, is to include a transition counter.

As previously discussed, merging of configurations is not expected to be a usable technique since the configuration contains a lot of concrete information (e.g., the threads' program counters and the owners of the program locks).

The techniques discussed above (a second timeout combined with a transition counter) should basically guarantee termination of the analysis, provided that suitable limits are chosen. But note that this is still an open question for the analysis presented in this thesis.

8.5 Other Applications of the Analysis

Given that the analysis terminates, some interesting results follow. The analysis could be used as a precise deadlock analysis including the timing behavior of the program. If the set of deadlocked configurations (c.f., $\tilde{C}^d$ in Algorithm 6.1) is empty, the program is deadlock free up until (and including) the point in time described by the timeout.

Furthermore, the analysis could also be used to determine whether a program is guaranteed to terminate. If the sets containing deadlocked and timed-out configurations (i.e., $\tilde{C}^d$ and $\tilde{C}^t$ in Algorithm 6.1, respectively) are empty, the program is guaranteed to terminate within the returned timing bounds.

8.6 Future Work

Some concrete tasks that will be performed in the near future is to implement and evaluate the analysis presented in this thesis. The implementation will be done in a suitable programming language. Which language is yet to be decided, but C/C++ and Erlang are top candidates.

Since PPL is rudimentary and designed to put focus on global data flow and thread synchronization, the implementation could use and analyze a more realistic instruction set. Some candidate instruction sets are LLVM [87], ALF [26, 27], ARM [5] and PowerPC [42]. It could also be possible to make the implementation a flexible framework which could allow the analyzed instruction set to be switched. This could be done by dividing instructions into special classes.

Some model of the underlying architecture (i.e., the function ABSTIME) must also be derived. Since the focus of this thesis has excluded any form of definition of ABSTIME, some very simple, and perhaps even non-realistic, timing model will most probably be used. Several different timing models should be evaluated to investigate how the characteristics of their definitions affect the complexity of the analysis.

The evaluation will be performed on some suitable benchmark suite of parallel programs. Such a suite is currently being established within the TACLe EU COST Action [86] (a European network of leading researchers within the field of WCET analysis) and will include parallel versions of some of the programs in the Mälardalen WCET Benchmark suite [24]. The benchmark suite should include different types of parallel programs, each of them stressing the analysis in a different way.

It is expected that the evaluation will result in hints pointing to some parts of the analysis that suffer from severe complexity problems. Thus, improvements and strategies for complexity reduction for these parts should be derived and implemented. One point that is already apparent in the case study in Section 7.2 is that the calculations in DLLOCK, defined in Algorithm 5.11, should be made tighter if possible (c.f., the result of the Study in Section 7.2, as shown in Table 7.6).

Bibliography

[1] S. V. Adve and K. Gharachorloo. Shared memory consistency models: A tutorial. Technical report, Rice University and Western Research Laboratory, 1995.

[2] R. Alur. Timed automata. In *Lecture Notes in Computer Science*, volume 1633/1999. Springer Berlin / Heidelberg, Jan. 1999.

[3] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183 – 235, Apr. 1994.

[4] A. Andrei, P. Eles, Z. Peng, and J. Rosén. Predictable implementation of real-time applications on multiprocessor systems-on-chip. *International Conference on VLSI Design*, 21:103–110, 2008.

[5] ARM Ltd. Arm Information Center, 2013. `http://infocenter.arm.com/help/index.jsp`.

[6] C. Ballabriga, H. Cassé, C. Rochange, and P. Sainrat. OTAWA: An open toolbox for adaptive WCET analysis. In *Software Technologies for Embedded and Ubiquitous Systems*, pages 35–46. Springer, 2011.

[7] G. Behrmann, A. David, and K. G. Larsen. A tutorial on UPPAAL. In *Proc. 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems*, volume 3185, pages 200–236. Springer Berlin / Heidelberg, Dec. 2004.

[8] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In *Lecture Notes in Computer Science*, volume 3098/2004, pages 87–124. Springer Berlin / Heidelberg, July 2004.

[9] A. Burns and A. Wellings. *Real-Time Systems and Programming Languages*. Addison-Wesley, third edition, 2001.

[10] D. R. Butenhof. *Programming with POSIX Threads*. Addison-Wesley, 1997.

[11] S. Chattopadhyay, C.-L. Kee, A. Roychoudhury, T. Kelter, P. Marwedel, and H. Falk. A unified WCET analysis framework for multi-core platforms. In *18th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS'12)*, Beijing, China, Apr. 2012.

[12] A. Colin and G. Bernat. Scope-tree: a program representation for symbolic worst-case execution time analysis. In *Proc. 14th Euromicro Conference on Real-Time Systems, (ECRTS'02)*, pages 50–59, Vienna, June 2002.

[13] A. Colin and I. Puaut. Worst case execution time analysis for a processor with branch prediction. *Journal of Real-Time Systems*, 18(2/3):249–274, May 2000.

[14] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, Jan. 1977.

[15] M. B. Dwyer and L. A. Clarke. Data flow analysis for verifying properties of concurrent programs. In *Proc. ACM SIGSOFT '94 Symposium on the Foundations of Software Engineering*, pages 62–75, Dec. 1994.

[16] J. Engblom. *Processor Pipelines and Static Worst-Case Execution Time Analysis*. PhD thesis, Uppsala University, Dept. of Information Technology, Uppsala, Sweden, Apr. 2002. ISBN 91-554-5228-0.

[17] A. Ermedahl. *A Modular Tool Architecture for Worst-Case Execution Time Analysis*. PhD thesis, Uppsala University, Dept. of Information Technology, Uppsala University, Sweden, June 2003.

[18] A. Ermedahl, J. Gustafsson, and B. Lisper. Deriving WCET bounds by abstract execution. In C. Healy, editor, *Proc. 11th International Workshop on Worst-Case Execution Time Analysis (WCET'2011)*, Porto, Portugal, July 2011.

[19] A. Ermedahl and M. Sjödin. Interval analysis of C-variables using abstract interpretation, 1996.

[20] C. Ferdinand, R. Heckmann, and B. Franzen. Static memory and timing analysis of embedded systems code. In *3rd European Symposium on Verification and Validation of Software Systems (VVSS'07), Eindhoven, The Netherlands*, number 07-04 in TUE Computer Science Reports, pages 153–163, Mar. 2007.

[21] D. Grunwald and H. Srinivasan. Data flow equations for explicitly parallel programs. In *Proc. Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 159–168, 1993.

[22] N. Guan, M. Stigge, W. Yi, and G. Yu. New response time bounds for fixed priority multiprocessor scheduling. In *Proc. 30th IEEE Real-Time Systems Symposium (RTSS'09)*, pages 387–397, Dec. 2009.

[23] J. Gustafsson. *Analyzing Execution-Time of Object-Oriented Programs Using Abstract Interpretation*. PhD thesis, Dept. of Information Technology, Uppsala University, Sweden, May 2000.

[24] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper. The Mälardalen WCET benchmarks – past, present and future. In B. Lisper, editor, *Proc. 10th International Workshop on Worst-Case Execution Time Analysis (WCET'2010)*, pages 137–147, Brussels, Belgium, July 2010. OCG.

[25] J. Gustafsson and A. Ermedahl. Merging techniques for faster derivation of WCET flow information using abstract execution. In R. Kirner, editor, *Proc. 8th International Workshop on Worst-Case Execution Time Analysis (WCET'2008)*, Prague, Czech Republic, July 2008.

[26] J. Gustafsson, A. Ermedahl, and B. Lisper. ALF (ARTIST2 Language for Flow Analysis) specification. Technical report, Mälardalen University, Västerås, Sweden, Jan. 2009.

[27] J. Gustafsson, A. Ermedahl, B. Lisper, C. Sandberg, and L. Källberg. ALF – a language for WCET flow analysis. In N. Holsti, editor, *Proc. 9th International Workshop on Worst-Case Execution Time Analysis (WCET'2009)*, pages 1–11, Dublin, Ireland, June 2009. OCG.

[28] J. Gustafsson, A. Ermedahl, C. Sandberg, and B. Lisper. Automatic derivation of loop bounds and infeasible paths for WCET analysis using

abstract execution. In *Proc. 27th IEEE Real-Time Systems Symposium (RTSS'06)*, pages 57–66, Rio de Janeiro, Brazil, Dec. 2006. IEEE Computer Society.

[29] A. Gustavsson. Worst-case execution time analysis of parallel systems. In *Proc. of Real Time in Sweden 2011 (RTiS2011)*, 2011.

[30] A. Gustavsson, A. Ermedahl, B. Lisper, and P. Pettersson. Towards WCET analysis of multicore architectures using UPPAAL. In B. Lisper, editor, *Proc. 10th International Workshop on Worst-Case Execution Time Analysis (WCET'2010)*, pages 103–113, Brussels, Belgium, July 2010. OCG.

[31] A. Gustavsson, J. Gustafsson, and B. Lisper. Toward static timing analysis of parallel software. In T. Vardanega, editor, *Proc. 12th International Workshop on Worst-Case Execution Time Analysis (WCET'2012)*, volume 23 of *OpenAccess Series in Informatics (OASIcs)*, pages 38–47, July 2012.

[32] A. Gustavsson, J. Gustafsson, and B. Lisper. Toward static timing analysis of parallel systems – technical report. Technical Report 2796, Dept. of Computer Science and Engineering, Mälardalen University, Apr. 2012.

[33] A. Gustavsson, J. Gustafsson, and B. Lisper. Timing analysis of parallel software using abstract execution. In *Will be submitted to the 15th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, 2014.

[34] D. Hardy, T. Piquet, and I. Puaut. Using bypass to tighten WCET estimates for multi-core processors with shared instruction caches. In *Proc. 30th IEEE Real-Time Systems Symposium (RTSS'09)*, pages 68–77, 2009.

[35] C. Healy, R. Arnold, F. Müller, D. Whalley, and M. Harmon. Bounding pipeline and instruction cache performance. *IEEE Transactions on Computers*, 48(1):53–70, Jan. 1999.

[36] C. Healy, M. Sjödin, V. Rustagi, D. Whalley, and R. van Engelen. Supporting timing analysis by automatic bounding of loop iterations. *Journal of Real-Time Systems*, 18(2-3):129–156, May 2000.

[37] N. Holsti and S. Saarinen. Status of the Bound-T WCET tool. In *Proc. 2nd International Workshop on Worst-Case Execution Time Analysis (WCET'2002)*, 2002.

[38] B. Huber and M. Schoeberl. Comparison of implicit path enumeration and model checking based WCET analysis. In *Proc. 9th International Workshop on Worst-Case Execution Time Analysis (WCET'2009)*, 2009.

[39] IEEE and The Open Group. The Open Group Base Specifications Issue 6, 2004. `http://pubs.opengroup.org/onlinepubs/009695399/.functions/pthread_spin_lock.html`.

[40] V. Illingworth, editor. *Oxford Dictionary of Computing*. Oxford University Press, fourth edition, 2003.

[41] Intel Corporation. *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference*, 1999.

[42] International Business Machines (IBM) Corporation. *PowerPC User Instruction Set Architecture*, 2005.

[43] J.-P. Katoen and Friedrich-Alexander. Concepts, algorithms, and tools for model checking. In *Lecture Notes of the Course "Mechanised Validation of Parallel Systems" (course number 10359)* Semester 1998/1999, at Universität Erlangen-Nürnberg.

[44] T. Kelter, T. Harde, P. Marwedel, and H. Falk. Evaluation of resource arbitration methods for multi-core real-time systems. In C. Maiza, editor, *Proc. 13th International Workshop on Worst-Case Execution Time Analysis (WCET'2013)*, pages 1–10. Schloss Dagstuhl, 2013.

[45] S. C. Kleene. *Introduction to Metamathematics*. North-Holland Publishing Co, Jan. 1980.

[46] J. Knoop, B. Steffen, and J. Vollmer. Parallelism for free: Efficient and optimal bitvector analyses for parallel programs. *ACM Trans. Program. Lang. Syst.*, 18(3):268–299, May 1996.

[47] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1-2):134–152, Dec. 1997.

[48] J. Lee, S. P. Midkiff, and D. A. Padua. A constant propagation algorithm for explicitly parallel programs. *International Journal of Parallel Programming*, 26(5):563–589, Oct. 1998.

[49] X. Li, Y. Liang, T. Mitra, and A. Roychoudhury. Chronos: A timing analyzer for embedded software. *Science of Computer Programming*, 69(1 - 3):56–67, 2007.

[50] Y.-T. S. Li and S. Malik. Performance analysis of embedded software using implicit path enumeration. In *Proc. ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems (LCT-RTS'95)*, La Jolla, CA, June 1995.

[51] S. Lim, J. Han, J. Kim, and S. L. Min. A worst case timing analysis technique for multiple-issue machines. In *Proc. 19th IEEE Real-Time Systems Symposium (RTSS'98)*, Dec. 1998.

[52] S.-S. Lim, Y. H. Bae, C. T. Jang, B.-D. Rhee, S. L. Min, C. Y. Park, H. Shin, K. Park, and C. S. Ki. An accurate worst-case timing analysis for RISC processors. *IEEE Transactions on Software Engineering*, 21(7):593–604, Jul 1995.

[53] B. Lisper. Towards parallel programming models for predictability. In T. Vardanega, editor, *Proc. 12th International Workshop on Worst-Case Execution Time Analysis (WCET'2012)*, volume 23 of *OpenAccess Series in Informatics (OASIcs)*, pages 48–58, July 2012.

[54] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[55] T. Lundqvist. *A WCET Analysis Method for Pipelined Microprocessors with Cache Memories*. PhD thesis, Chalmers University of Technology, Göteborg, Sweden, June 2002.

[56] T. Lundqvist and P. Stenström. Timing Anomalies in Dynamically Scheduled Microprocessors. Technical Report 99-5, Chalmers University of Technology, Apr 1999.

[57] M. Lv, N. Guan, W. Yi, Q. Deng, and G. Yu. Efficient instruction cache analysis with model checking. In *Proc. 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'10), Work-in-Progress Session*, pages 33–36, Apr. 2010.

[58] M. Lv, N. Guan, W. Yi, and G. Yu. Combining abstract interpretation with model checking for timing analysis of multicore software. In S. Brandt, editor, *Proc. 31st IEEE Real-Time Systems Symposium (RTSS'10)*, pages 339–349, San Diego, CA, Dec. 2010. IEEE.

[59] MERASA. MERASA project, 2013. `http://www.merasa.org`.

[60] A. Metzner. Why model checking can improve WCET analysis. In *Lecture Notes in Computer Science*, volume 3114/2004, pages 298–301. Springer Berlin / Heidelberg, July 2004.

[61] R. Mittermayr and J. Blieberger. Timing analysis of concurrent programs. In *Proc. 12th International Workshop on Worst-Case Execution Time Analysis (WCET'2012)*, pages 59–68, 2012.

[62] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis, 2nd edition*. Springer, 2005. ISBN 3-540-65410-0.

[63] H. R. Nielson and F. Nielson. *Semantics with Applications – A Formal Introduction*. John Wiley & Sons (1992), July 1999.

[64] OpenMP. OpenMP Application Program Interface, Version 3.0, May 2008. `http://www.openmp.org/mp-documents/spec30.pdf`.

[65] H. Ozaktas, C. Rochange, and P. Sainrat. Automatic WCET Analysis of Real-Time Parallel Applications. In *Proc. 13th International Workshop on Worst-Case Execution Time Analysis (WCET'2013)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013.

[66] M. Paolieri, E. Quiñones, F. J. Cazorla, G. Bernat, and M. Valero. Hardware support for WCET analysis of hard real-time multicore systems. In *Proc. 36th International Symposium on Computer Architecture (ISCA 2009)*, pages 57–68, 2009.

[67] M. Paolieri, E. Quiñones, F. J. Cazorla, and M. Valero. GAMC: A generic analyzable memory controller for hard real-time multicore processors. Technical report, Departament d'Arquitectura de Computadors, Universitat Politècnica de Catalunya, May 2009.

[68] C. Y. Park and A. C. Shaw. Experiments with a program timing tool based on a source-level timing schema. *IEEE Computer*, 24(5):48–57, 1991.

[69] parMERASA. parMERASA | multi-core execution of parallelised hard real-time applications supporting analysability, 2013. `http://www.parmerasa.eu`.

[70] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design*. Morgan Kaufmann Publishers Inc., 4th edition, Nov. 2008. ISBN 9780123744937.

[71] D. Potop-Butucaru and I. Puaut. Integrated Worst-Case Execution Time Estimation of Multicore Applications. In *Proc. 13th International Workshop on Worst-Case Execution Time Analysis (WCET'2013)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013.

[72] A. Prantl, M. Schordan, and J. Knoop. TuBound – a conceptually new tool for worst-case execution time analysis. In R. Kirner, editor, *Proc. 8th International Workshop on Worst-Case Execution Time Analysis (WCET'2008)*, pages 141–148, Prague, Czech Republic, July 2008.

[73] P. P. Puschner and A. Burns. A Review of Worst-Case Execution-Time Analysis. *Real-Time Systems*, 18(2/3):115–128, 2000.

[74] P. P. Puschner and A. V. Schedl. Computing maximum task execution times – a graph-based approach. *Journal of Real-Time Systems*, 13(1):67–91, July 1997.

[75] Rapita Systems. Rapitime white paper, 2009. `www.rapitasystems.com/system/files/RapiTime-WhitePaper.pdf`.

[76] J. Reineke, B. Wachter, S. Thesing, R. Wilhelm, I. Polian, J. Eisinger, and B. Becker. A definition and classification of timing anomalies. In *Proc. 6th International Workshop on Worst-Case Execution Time Analysis (WCET'2006)*, July 2006.

[77] M. Rinard. Analysis of multithreaded programs. In P. Cousot, editor, *Proc. 8th Static Analysis Symposium*, Vol. 2621 of *Lecture Notes in Comput. Sci.*, pages 1–19, Paris, France, July 2001. Springer-Verlag.

[78] C. Rochange, A. Bonenfant, P. Sainrat, M. Gerdes, J. Wolf, T. Ungerer, Z. Petrov, and F. Mikulu. WCET analysis of a parallel 3D multigrid solver executed on the MERASA multi-core. In B. Lisper, editor, *Proc. 10th International Workshop on Worst-Case Execution Time Analysis (WCET'2010)*, pages 90–100, Brussels, Belgium, July 2010. OCG.

[79] J. Rosen, A. Andrei, P. Eles, and Z. Peng. Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In *Proc. 28th IEEE Real-Time Systems Symposium*

(RTSS'07), pages 49–60, Washington, DC, USA, 2007. IEEE Computer Society.

[80] J. Schneider and C. Ferdinand. Pipeline behaviour prediction for superscalar processors by abstract interpretation. In *Proc. ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems (LCTES'99)*. ACM Press, May 1999.

[81] T. Shanley. *x86 Instruction Set Architecture*. Mindshare Press, Dec. 2009.

[82] A. C. Shaw. Reasoning about time in higher-order software. In *IEEE Transactions on Software Engineering*, volume 15, pages 737–750, 1989.

[83] F. Stappert and P. Altenbernd. Complete worst-case execution time analysis of straight-line hard real-time programs. *Journal of Systems Architecture*, 46(4):339–355, 2000.

[84] F. Stappert, A. Ermedahl, and J. Engblom. Efficient longest executable path search for programs with complex flows and pipeline effects. In *Proc. 4th International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, (CASES'01)*, Nov. 2001.

[85] J. Staschulat, S. Schliecker, M. Ivers, and R. Ernst. Analysis of memory latencies in multi-processor systems. In R. Wilhelm, editor, *Proc. 5th International Workshop on Worst-Case Execution Time Analysis (WCET'2005)*, Palma de Mallorca, July 2005.

[86] TACLe. Timing Analysis on the Code Level (TACLe), 2013. `http://www.tacle.eu`.

[87] The LLVM Project. The LLVM Compiler Infrastructure Project, 2013. `http://llvm.org`.

[88] S. Thesing. *Safe and Precise WCET Determination by Abstract Interpretation of Pipeline Models*. PhD thesis, Saarland University, 2004.

[89] UPPAAL. UPPAAL website, 2013. `http://uppaal.org`.

[90] R. White, F. Müller, C. Healy, D. Whalley, and M. Harmon. Timing Analysis for Data Caches and Set-Associative Caches. In *Proc. 3rd IEEE Real-Time Technology and Applications Symposium (RTAS'97)*, pages 192–202, June 1997.

[91] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution time problem — overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):1–53, 2008.

[92] Wind River. Wind River VxWorks RTOS, 2013. `http://www.windriver.com/products/vxworks`.

[93] L. Wu and W. Zhang. Bounding worst-case execution time for multicore processors through model checking. In *Proc. 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'10), Work-in-Progress Session*, pages 17–20, Apr. 2010.

[94] J. Yan and W. Zhang. WCET analysis for multi-core processors with shared L2 instruction caches. In *Proc. 14th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'08)*, pages 80–89, June 2008.

[95] J. Yan and W. Zhang. Accurately estimating worst-case execution time for multi-core processors with shared direct-mapped instruction caches. In *Proc. 15th International Conference on Real-Time Computing Systems and Applications (RTCSA'09)*, pages 455–463, Aug. 2009.

Appendix A

Notation & Nomenclature

$exp_1 @ exp_2$	exp_1 and exp_1 denote the same thing, often a short and a long notation for a configuration.
$b \; ? \; exp_1 : exp_2$	If b, then exp_1, otherwise exp_2.
$(o_1, \ldots, o_n)$	Ordinary tuple containing n elements.
$\langle o_1, \ldots, o_n \rangle$	Special tuple containing n elements. Used to denote complete lattices, Galois connections, configurations, etc.
$[o_1, \ldots, o_n]_{e \in \{e_1, \ldots, e_m\}}$	Expands to $o_1^1, \ldots, o_n^1, \ldots, o_1^m, \ldots, o_n^m$; i.e., one instance of $o_1, \ldots, o_n$ for each $e \in \{e_1, \ldots, e_m\}$. Used inside special tuples.
S	An arbitrary set (capitalized, italic notation).
$\mathbb{S}$	A standard set (capitalized, blackboard bold notation); e.g., $\mathbb{Z}$.
Set	A set of analysis-specific elements (first letter capitalized, bold notation); e.g., **Thrd**.
$\mathscr{P}(S)$	The powerset of S; i.e., $\{S' \mid S' \subseteq S\}$.
$S \times S'$	The Cartesian product; i.e., $\{(e, e') \mid e \in S \wedge e' \in S'\}$.
$\mathscr{P}_{e \in \{e_1, \ldots, e_m\}}(exp(e))$	Expands to $exp(e_1) \times \ldots \times exp(e_m)$.

$e, e' \in S$	Short for $e \in S \wedge e' \in S$.
$\lambda e \in S.exp$	A function from e, which is an element of S, to exp, which is often dependent on the specific e.
$f(s)$	The function f applied on s.
$f \circ g(o)$	Equivalent to $f(g(o))$.
$f[\![o_1]\!]o_2$	Equivalent to $(f(o_1))(o_2)$.
$f\ s$	The function f applied on s. This notation is used when dereferencing mappings.
$\mathbb{f}$	Denotes a state (i.e., a function/mapping from elements to values); e.g., $\mathbb{r}$.
$\mathbb{f}[s' \mapsto exp]$	Remap; defined as: $\mathbb{f}[s' \mapsto exp]\ s = \begin{cases} exp & \textbf{if } s = s' \\ \mathbb{f}\ s & \textbf{otherwise} \end{cases}$
ALGFUNC	A function defined in a table or algorithm.
T	One of the threads defined in the analyzed program.
Π, **Thrd**	The analyzed program; i.e., a set of threads.
r	Register (thread-local memory).
$\mathbf{Reg}_{\mathrm{T}}$	The set of registers used by thread T.
x	Variable (global memory).
Var	The variables defined in the program.
lck	Lock (shared resource).
Lck	The locks defined in the program.
pc	Program counter (unique for each thread).
$\tilde{f}$	f defined in some abstract domain; i.e., an abstraction of f.
$\mathbb{r}$, $\tilde{\mathbb{r}}$	Mapping from registers to their values (unique for each thread).
t^a, $\tilde{t}^a$	Accumulated execution time (unique for each thread).
$\mathbb{x}$, $\tilde{\mathbb{x}}$	Mapping from variables to mappings from threads to their write history for the given variable.

$\mathbb{l}, \tilde{\mathbb{l}}$	Mapping from locks to their values.
$c, \tilde{c}$	Configuration (system state).
$\sqsubseteq$	Partial order relation.
$\bot$	The bottom element in a complete lattice.
$\top$	The top element in a complete lattice.
$\bigsqcup$	The least upper bound operator.
$\bigsqcap$	The greatest lower bound operator.
α	Abstraction function.
γ	Concretization function.
$\xrightarrow[ax]{}, \xrightarrow[ax]{\sim}$	Transition relation for statements (i.e., axioms).
$\xrightarrow[prg]{}, \xrightarrow[prg]{\sim}$	Transition relation for threads (i.e., the program).
$t_{to}, \tilde{t}_{to}$	The timeout variable used by the analysis.
$\triangleright$	Begins a comment within algorithms.

Final configurations are configurations in which all the threads issue the `halt`-statement.

Final states is an alternative notation for final configurations.

Deadlocked configurations are configurations that can never reach the final state.

Timed-out configurations are configurations that cannot reach the final state before a given point in time, the timeout.

Truly deadlocked configurations are abstract configurations that are deadlocked and have valid concrete counterparts; i.e., there is at least one semantically valid concrete configuration that can be abstracted by the given configuration. It must thus be that all threads included in the deadlock are owners of some lock, which has the state *locked*, and are waiting to acquire some other lock, which also has the state *locked*.

Falsely deadlocked configurations are abstract configurations that are deadlocked and do not have any valid concrete counterpart; i.e., there is no

semantically valid concrete configuration that can be abstracted by the given configuration. It could thus be that some thread included in the deadlock is the owner of some lock, which has the state *unlocked*, and that some other thread included in the deadlock is waiting to acquire that lock.

Axiom statements are labeled statements; i.e., statements that are not composed of several statements.

Composed statements are statements that are composed by two or more axiom (i.e., labeled) statements.

Active statements are the axiom statements pointed to by the threads' program counters. The active statement is the statement that is executed when the thread is executed. Only one statement in each thread can be active at any given point in time since all the axiom statements within a thread are uniquely labeled.

Frozen threads are threads in an abstract configuration whose active statements are `lock`-statements and the locks they are trying to acquire are currently owned by some other thread.

Active threads are not frozen and their active statements are not `halt`. Note that this applies to all threads in any concrete configuration, given that they are not issuing the `halt`-statement, since only threads in an abstract configuration can be frozen.

Executing threads are the active threads that will execute their active statement at the nearest point in time.

BCET (Best-Case Execution Time) is the shortest possible execution time of the program, given a certain set of initial states.

WCET (Worst-Case Execution Time) is the longest possible execution time of the program, given a certain set of initial states.

Appendix B

List of Assumptions

Appendix C

List of Definitions

Appendix D

List of Figures

Appendix E

List of Tables

Appendix F

List of Algorithms

Appendix G

List of Lemmas

Appendix H

List of Theorems

Index