ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

in cotutela con Università **di Grenoble-Alpes**

DOTTORATO DI RICERCA IN

Informatica

Ciclo XXXVII

Settore Concorsuale: 01/B1 Informatica Scienza e Ingegneria

Settore Scientifico Disciplinare: INF/01 Informatica

Reversibility for Concurrent Memory Models

Presentata da: ***Pietro Lami***

Coordinatore Dottorato

Ilaria Bartolini

Supervisore

Ivan Lanese

Jean-Bernard Stefani

Reversibility for Concurrent Memory Models

Reversibility for Concurrent Memory Models

Abstract: Reversibility in concurrent programming languages has been extensively studied in message-passing systems, but has not been studied in concurrent shared memory models. In this work, we explore causal consistent reversibility in shared memory models, addressing the challenges of making shared memory reversible, particularly when weak and non-standard memory models are involved. We begin by extending reversible Erlang to handle shared memory constructs, highlighting the difficulties that arise from non-standard memory operations, such as reading entire maps. Next, we propose a structured, two-step approach. First, we introduce a meta-model framework that defines memory models as synchronous products of labeled transition systems (LTSs), incorporating three components: threads, memory, and scheduler. This framework can describe various memory models, including sequential consistency, write buffer, and transactional. Second, we develop a compositional theory to make the product of LTSs reversible, ensuring causal consistency. We apply this theory to a sequential consistency memory model, comparing it to an automatic approach and showing that our method avoids capturing unwanted dependencies.

Key-words: reversibility, concurrency theory, reversible computing, memory models, synchronous product of labeled transition systems.

Réversibilité pour les Modèles de Mémoires Concurrentes

Résumé: La notion de réversibilité dans les langages de programmation concurrents a été bien étudiée dans le cas de systèmes à base d'échange de messages, mais pas dans le cas de systèmes concurrents à base de mémoire partagée. Dans cette thèse, nous étudions la réversibilité causalement cohérente dans des modèles de mémoire partagée, notamment dans le cas difficile de modèles mémoires faibles ou non-standards. Nous commençons par étendre un Erlang réversible pour traiter des constructions du langage impliquant une mémoire partagée, ce qui permet de mettre en évidence les difficultés qui apparaissent en présence d'opérations mémoire non-standards. Nous proposons ensuite une approche structurée en deux étapes. Tout d'abord, nous développons un méta-modèle permettant de définir un modèle mémoire comme un produit de synchronisation de systèmes de transitions étiquetés comprennants trois éléments: des fils d'exécution, une mémoire et un ordonnanceur. Ce méta-modèle est utilisé pour décrire plusieurs modèles mémoire, notamment mémoire séquentiellement cohérente, mémoire avec buffer d'écriture et mémoire transactionnelle. Nous développons ensuite une théorie compositionnelle pour rendre réversible un produit de systèmes de transition étiquetés tout en garantissant la cohérence causale. Nous appliquons cette théorie à une mémoire séquentiellement cohérente, en la comparant à une approche automatique et en montrant que notre méthode compositionnelle permet d'éviter l'introduction de dépendances causales non justifiées.

Mots-clés: réversibilité, théorie de la concurrence, programmation réversible, modèles de mémoire, produit synchrone de systèmes de transition étiquetés.

Acknowledgments

Scrivere i ringraziamenti qui

Contents

Chapter 1

Introduction

The main scope of this thesis is to study reversibility in concurrent systems with shared memory architecture. Reversibility in computation refers to the ability of a system to execute operations both forwards and backwards, thus allowing the system to go back to previous states. When this concept is studied in concurrent systems, where multiple threads or processes operate simultaneously and interact with shared memory, it is important to understand how the processes can interact with each other. For this reason, it is also interesting to focus on the behavior of the interactions between threads and memory.

1.1 Introduction to Reversibility

When we talk about a computation, we naturally think to a process that only progresses forward, from an initial state to a final state through a series of well-defined steps. This view underlies our understanding of most standard computational processes. However, when we consider the field of reversible computation, what we typically consider as the "normal flow" of computation takes on a new dimension. Reversible computation invites us to consider an execution where not only we can proceed forward through a series of operations, but we can also backtrack, performing a reverse operation to move backward along the same path. This concept is essential for understanding reversibility in computational systems, where the ability to perform operations forward and backward is crucial.

Reversibility studies have their origin in the studies on low power computing conducted by Landauer in 1961 [1]. These studies focused on analyzing the thermodynamic cost associated with irreversible actions in the computation process. In particular, it emerged that, in conventional computation, irreversible actions lead to a loss of information, which results in heat dissipation. Since heat dissipation is associated with increased energy consumption and the production of unwanted heat, the potential to perform reversible computation without generating heat emerged.

Historically, as we have shown, a main motivation behind the development of reversible computing is low power computing since reversible computations can be performed without dissipating heat. Nowadays, additional areas have emerged in which reversible computing has found new applications, such as simulation [2] where it can improve performance by decreasing state memory requirements and reducing the overhead associated with traditional state-saving

methods, robotics [3] since it allows one to handle errors efficiently and safely during some operations, biological modeling [4] since several biological reactions are by nature reversible, program debugging [5] and testing as it allows one to restore the state of the program to a previous point in the computation where certain conditions are met. Additionally, reversible computing has other applications.

In short, reversible computation invites us to reconsider our traditional concept of computational flow, opening up new possibilities and challenges in the design and implementation of advanced computational systems. In reversible computing the general problem being studied, which can be modified and specialized depending on the context, is that a given program can be executed in such a way that its forward execution can be paused at any time and the direction of its execution can be reversed to go back to the previous states. Backward execution can also be paused at any time and forward execution of the program can be re-started. The possibility of stopping forward or backward execution at any time and the possibility of reversing the direction of execution represent the generalized challenge of reversible computing.

1.2 Reversibility in Concurrent Systems

Reversible computation allows for the execution of a program in both forward and backward directions. However, not all programs can be executed in reverse due to potential loss of information during their execution. For instance, an assignment like $x = 96$ results in irreversible loss of the previous value of x, making it impossible to execute the program in reverse.

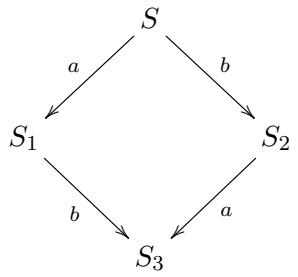
It is possible differentiate between programming languages that permit instructions resulting in information loss and those that exclusively allow instructions without such loss. Notably, among the latter category, Janus stands out, as the first structured reversible programming language, designed by Christopher Lutz and Howard Derby [6]. Programming languages like Janus only allow instructions with mathematical inverse functions, enabling straightforward execution in both forward and backward directions. The literature on pure reversible programming languages includes also object-oriented languages such as ROOPL [7, 8] and Joule [9], and functional languages such as RFun [10] and CoreFun [11].

Every mainstream programming language allows instructions with information loss, anyway it was noted that any irreversible computation can be transformed into a reversible computation by including it in a larger computation in which no information is lost, saving intermediate states each time so that no information is lost. The idea behind this work [1], called *Landauer embedding*, is that any programming language or formalism can be made reversible by adding the computation history to each state. This idea was further improved by Bennett [12] in order to avoid the generation of "junk" data by applying a series of analyses in order to limit the information required in the history as much as possible.

In a sequential system, undoing forward actions in reverse order of completion starting from the last one produces a backward computation. Undoing a forward action can be seen as a backward action. In a concurrent environment, one cannot easily decide which is the last action since many actions can be executed at the same time, and a total order of actions may not be

available. Even if a total order exists, undoing actions in reverse order may be too restrictive since the order of execution of concurrent actions may depend on the relative speed of the processors executing them and has no impact on the final state. For instance, when looking for a bug causing a visible misbehavior in a concurrent system, independent actions may be disregarded since they cannot contain the bug.

The first definition of reversibility in a concurrent setting has been proposed by Danos and Krivine [13]: *causal-consistent reversibility*. In short, it states that any action can be undone provided that all its effects (if any) have been undone. By this notion, reversing a computation means undoing actions in a way that maintains causal consistency. This means that during backward computation, states reached could have been reached by executing independent actions in a different order. This occurs because the model does not track the specific order in which concurrent actions are executed. To better understand this problem let us consider the following example:



where from the state S, there are two possible execution paths to reach the state S_3. We can either execute the sequence $a\ b$ (choosing the left path) or the sequence $b\ a$ (choosing the right path). Assume the actions a and b are concurrent and independent, as both can be executed from state S. Therefore, we can proceed either from S to S_1 to S_3 or from S to S_2 to S_3. Let's assume that the forward computation leading to S_3 follows the sequence $a\ b$. Now, from state S_3, we have the capability to go back to state S by first undoing action a, followed by action b. This takes us to state S_2, which is a state that could have been reached during the forward computation by simply swapping the execution order of concurrent actions, i.e., executing b before a. In other words, we can achieve the following execution: $S \to S_1 \to S_3 \rightsquigarrow S_2 \rightsquigarrow S$, where $\rightsquigarrow$ denotes a backward step.

1.3 Introduction to Memory Models

In concurrent programming, one of the main challenges is to ensure proper synchronization between threads running in parallel. Generally, there are various methods to achieve synchronization between threads, including shared memory, message passing, transactions and tuple spaces. *Shared memory* allows threads to communicate by sharing access to common variables stored in a shared memory. Threads can read from and write to these variables, but synchronization is required to prevent data races and ensure consistency. Mechanisms such as locks, semaphores, or atomic operations are often employed to coordinate access to shared memory among threads. *Message passing* [14], commonly used in languages like Erlang [15], involves threads communicating by explicitly sending and receiving messages.

Threads interact through message queues or channels, which facilitate communication and synchronization. Message passing can be implemented with blocking (synchronous) or non-blocking (asynchronous) operations, depending on the requirements of the application. *Transactions* [16, 17] treat sequences of operations as atomic units of work, ensuring that all operations within a transaction either succeed together or fail together. This atomicity simplifies synchronization by guaranteeing that concurrent operations do not interfere with each other's execution. Transactions are often used in database systems and have gained attention as an alternative to locks for synchronization, potentially improving efficiency. *Tuple space* [18], utilized by programming languages like Linda [18] and Klaim [19], define non-traditional synchronization primitives (`out()`, `in()`, `read()`). In this model, when two threads want to communicate, the first thread generates tuples and adds them to the tuple spaces, and the second thread withdraws them.

The synchronization methods we have discussed are just a few examples. Each programming language typically defines or adopts its own synchronization primitives and mechanisms, tailored to its specific needs and design goals. Synchronization primitives provide different levels of abstraction and flexibility, allowing programmers to choose the most appropriate method based on the application's concurrency requirements and design considerations.

A *memory consistency model*, or simply *memory model*, defines the rules and behaviors of communication and synchronization operations between processes or threads. The best known and most intuitive memory model, *sequential consistency* [20], is obtained by a sequential execution of the operations of the various threads. This model is too strong for modern languages, as it does not allow one to model many behaviors that can be observed on modern hardware due to compiler and CPU optimizations. This gave origin to the *weak* or *relaxed* memory models. In recent years, numerous weak memory models have been proposed for programming languages [21], with trade-offs in terms of programmability and optimization potential.

Let us consider an example to understand the importance of memory models. Here is a simplified version of Dekker's Lock [22]:

$x := 1$	$\|\|$	$y := 1$
$r_1 := y$	$\|\|$	$r_2 := x$
if $r_1 := 0$ {	$\|\|$	if $r_2 := 0$ {
$\quad$ *//critical section*	$\|\|$	$\quad$ *//critical section*
}	$\|\|$	}

In this scenario, two threads compete to enter the critical section. Each thread sets one variable (x or y) and then checks the other variable to determine if it is possible to enter the critical section. The algorithm ensures that both threads cannot read the value 0 at the same time; otherwise, both could enter the critical section, breaking the correctness of the algorithm. On a multi-core system, we would expect to see one of the following results: $[r_1 = 0, r_2 = 1]$, $[r_1 = 1, r_2 = 0]$, or $[r_1 = 1, r_2 = 1]$, these results are sequentially consistent. However, not all observable behavior on real concurrent systems is sequentially consistent. For example, if we consider the C or *Partial Store Order (PSO)* [23, 24, 25] memory models, we may observe non-sequentially consistent results like $[r_1 = 0, r_2 = 0]$. Such outcomes are called weak. Weak results

appear due to compiler and CPU optimizations. In this case considering the left thread, storing to x and loading from y are independent instructions and therefore they could be reordered. Then, the outcome $[r_1 = 0, r_2 = 0]$ is valid for the optimized program.

In summary, memory models define how threads and memory interact, detailing the possible execution orders of memory actions [26]. They range from the strict sequential consistency to various weak models like PSO, message passing and transactions, each offering different trade-offs between programmability and optimization potential. Understanding these models is essential when studying concurrent systems to gain a comprehensive and general understanding.

1.4 Contributions

Our goal is to explore causal consistent reversibility in programming languages based on shared memory concurrency models. Reversibility in concurrent languages has focused on message-passing, as we can see in RCCS [27], CCSK [28], reversible π [29], reversible higher-order π [30], and reversible Erlang [31], with reversible Klaim [32] being an exception that uses tuple spaces.

Initially, we focused on reversible Erlang [31] because it allows us to investigate the shared memory model based on maps. This model includes both classic operations like reading, writing, and deletion, as well as less standard operations such as reading the entire memory. Extending reversible Erlang helped us realize that making a language reversible is complex, especially when non-standard memory models are used. Automatic methods (like [33]) for obtaining reversible semantics do not always help, as they can capture more dependencies than we actually want, as in the case of Erlang maps (and also in the case of simple read and write operations). This ad-hoc extension worked for Erlang, but if we want to deal with different shared memory models, we can hardly use this ad-hoc approach, as this method is not easy to generalize to other shared memory models.

Given these challenges, we propose a more structured approach divided in two steps to address reversibility in shared memory concurrency models.

We first present a framework (which is actually a meta-model) that can represent most memory models as synchronous products of labeled transition systems (LTSs). This allows us to study reversibility through this framework. This framework is inspired by Effinger's framework (unpublished work [34]), which we extended to consider three components (thread, memory and scheduler) instead of two (thread and memory). This makes the framework more similar to a real system architecture. Our framework is described using the synchronous product $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ of three LTSs, each representing a different component: *thread semantics* (T), which defines the operational rules for the programming language; *memory semantics* (M), which manages memory representation and manipulation; and *scheduler semantics* (S), which controls process execution order and interactions. We categorize the labels of the LTSs into groups and define the synchronization relation $\mathcal{R}$ to manage the interactions among these components. This framework enables us to describe several specific memory models, including *sequential consistency*, *write buffer (PSO)*, and *transactional* memory models, and to demonstrate their bi-similarity with the corresponding models described in the literature.

In the second part we present a new compositional theory for making a product of LTSs

reversible, demonstrating that, given some properties of the individual LTSs, the synchronous product of these LTSs achieves causal consistency. We use an example to illustrate that our approach captures the desired dependencies, while the automatic method also introduces some unwanted dependencies. First, we introduce a monolithic LTS with a sequential consistency memory model and apply the automatic method described in [33] to make this model reversible. Then, we consider the sequential consistency memory model obtained through our framework and apply our compositional theory to make it a reversible product. This comparison highlights the advantages of our approach.

In conclusion, our work offers a modular framework for studying reversibility in shared memory concurrency models and provides a robust, compositional theory for achieving causal consistency in these systems. This approach not only generalizes across various memory models but also addresses the limitations of ad-hoc methods, opening the way for further exploration of reversibility in shared memory environments.

1.5 Outline

The thesis is structured in the following way:

Chapter 2: Background. This chapter provides a description of the works on which the thesis will be based. This chapter focuses on reversibility (more in general about reversible computing) and about the formalization of memory models using operational semantics.

Chapter 3: Reversibility in Erlang: Imperative constructs. In this chapter, we extend CauDEr, a causal-consistent debugger for Erlang, by adding imperative primitives for adding, deleting, and reading elements from a map, which can be seen as a form of shared memory. Additionally, we show the difficulties of extending and proving Causal Consistency in the system.

Chapter 4: A Meta-model for Memory Models. In this chapter, we describe our approach to presenting an operational semantics that generalizes the descriptions of memory models. We also present some instances of our framework and demonstrate the bi-similarity with corresponding memory models presented in the literature.

Chapter 5: A compositional theory of causal consistent reversibility. In this chapter, we explore Causal Consistency in systems made of multiple components and propose two approaches for establishing Causal Consistency in these systems. We then present an example that shows the benefits of applying this theory.

Chapter 6: Conclusion and Future Works. In this chapter we discuss the perspectives of future work and draw conclusions from our work.

Chapter 2

Background

In this chapter, we introduce the background and the works upon which this thesis is based.

We begin with a detailed description of CauDEr [5], a causal consistent debugger for Erlang, in Section 2.1. The principles and methodologies from this work will be extended further in Chapter 3.

Following this, in Section 2.2, we present the Effinger framework [34], which serves as the basis for our own framework introduced in Chapter 4. The Effinger framework provides an approach for describing shared memory concurrency models.

In Sections 2.3 and 2.4, we discuss two key works that are at the base of the theory developed in Chapter 5. Section 2.3 discusses the axiomatic approach to derive Causal Consistency from specific properties [35]. Section 2.4 introduces an automatic method for reversing a programming language [33], which we exploit in our exploration of reversible computation.

This chapter provides the basis for the contributions of subsequent chapters, providing the context and theoretical foundations necessary for our research.

2.1 CauDEr: a Causal-consistent Debugger for Erlang

The idea of a causal-consistent reversible debugger was introduced in [36]. Such a debugger allows one to explore a concurrent execution backwards, starting from a visible misbehavior, to identify the bug responsible for it. E.g., if the misbehavior is a wrong value of some variable X, one can look for the assignment that stored that value in X. If an assignment takes a value from some message M, one can look for the send of that message M, which originates possibly in another process. In general, one can define a rollback operator [37], which allows one to undo a target action far in the past, including all and only its consequences, which are possibly spread among multiple processes. Also, given a log of a concurrent execution one would like to replay a selected action, including all and only its causes. This is called a causal-consistent replay [38]. E.g., if the action is the misbehavior, this will replay the shortest computation showcasing it. Causal-consistent replay can be seen as dual of causal-consistent rollback.

In this Section we introduce the CauDEr debugger [5], described in [39, 40, 41, 38], a reversible debugger for the functional, concurrent and distributed fragment of the Erlang programming language [15]. CauDEr implements also the causal-consistent rollback and the causal-consistent replay. The rollback operator [37] allows one to undo a target action far in the past, including

$$
\begin{array}{rcl}
program & ::= & mod_1 \ldots mod_n \\
mod & ::= & fun\_def_1 \ldots fun\_def_n \\
fun\_def & ::= & fun\_rule\{';' fun\_rule\}'.' \\
fun\_rule & ::= & Atom\ fun \\
fun & ::= & ([exprs])\ [\textsf{when}\ expr] \rightarrow exprs \\
exprs & ::= & expr\ \{',' expr\} \\
expr & ::= & atomic \mid Var \mid\ '\{'[exprs]'\}' \mid\ '['[exprs|exprs]']' \mid \textsf{if}\ if\_clauses\ \textsf{end} \\
 & \mid & \textsf{case}\ expr\ \textsf{of}\ cr\_clauses\ \textsf{end} \mid \textsf{receive}\ cr\_clauses\ \textsf{end} \mid expr\ !\ expr \\
 & \mid & pattern = expr \mid [Mod:]expr([exprs]) \mid fun\_expr \mid Opexprs \\
atomic & ::= & Atom \mid Char \mid Float \mid Integer \mid String \\
if\_clauses & ::= & expr \rightarrow exprs\ \{';' expr \rightarrow exprs\} \\
cr\_clause & ::= & pattern\ [\textsf{when}\ expr] \rightarrow exprs\ \{';' pattern\ [\textsf{when}\ expr] \rightarrow exprs\} \\
fun\_expr & ::= & \textsf{fun}\ fun\ \{';' fun\}\ \textsf{end} \\
patterns & ::= & pattern\ \{',' pattern\} \\
pattern & ::= & atomic \mid Var \mid\ '\{'[patterns]'\}' \mid\ '['[patterns|pattern]']'
\end{array}
$$

Figure 2.1: Language syntax

all and only its consequences, which are possibly spread among multiple processes. Given a log of a concurrent execution one would like to replay a selected action, including all and only its causes. This is the causal-consistent replay [38]. E.g., if the action is the misbehavior, this will replay the shortest computation showcasing it. Causal-consistent replay can be seen as dual of causal-consistent rollback.

2.1.1 The Language Syntax

Erlang is a functional, concurrent and distributed programming language based on the actor paradigm [14] (concurrency based on asynchronous *message-passing*).

The syntax of the language is shown in Fig. 2.1. A program is a collection of module definitions, a module is a collection of function definitions, a function is a mapping between the function name and the function expression. An expression can be a variable, an atom, a list, a tuple, a call to a function, a case expression, an if expression, or a pattern matching equation. We distinguish expressions and patterns. Here, patterns are built from atomic values, variables, tuples and lists. When we have a case *expr* of *cr_clauses* end expression we first evaluate *expr* to a value, say v, then we search for a clause that matches v and such that the guard when *expr* is satisfied. If one is found then the case construct evaluates to the clause expression. The *if* expression is very similar to the evaluation of the *case* expression just described. Pattern matching is written as $pattern = expr$. Then, $expr_1$! $expr_2$ allows a process to send a message to another one. Expression $expr_1$ must evaluate to a pid and $expr_2$ evaluates to the message payload, indicated with v. The whole function evaluates to v and, as a side-effect, the message will be sent to the target process. The complementary operation of message sending is receive *cr_clauses* end. This construct takes a message targeting the process that matches one of the clauses. If no message is found then the process suspends.

Erlang includes a number of built-in functions (BIFs).

In [40], they only consider self, which returns the process identifier of the current process,

and spawn, that creates a new process. BIFs supporting distribution are considered in [41]. For a deeper discussion we refer to [40, 41].

2.1.2 Formal Semantics of Erlang

Here we describe the semantics of the language. We begin by providing the definitions of *process* and *system*.

Definition 1 (Process)**.** *A process is a tuple $\langle p, \theta, e, S\rangle$, where p is the process pid, θ is the process environment, e is the expression under evaluation and S is the stack of process environments.*

In the definition above, θ is a substitution, namely a mapping from variables to values. Given a substitution θ, $\mathrm{Dom}(\theta) = \{X \in Var \mid X \neq \theta(X)\}$ is its domain. Substitutions are usually denoted by (finite) sets of bindings like, e.g., $\{X_1 \mapsto v_1, \ldots, X_n \mapsto v_n\}$. Substitutions are extended to morphisms from expressions to expressions in the natural way. The identity substitution is denoted by id. Composition of substitutions is denoted by juxtaposition, i.e., $\theta\theta'$ denotes a substitution θ'' such that $\theta''(X) = \theta'(\theta(X))$ for all $X \in Var$. Stack S is used to store away the process state to start a sub-computation of the expression under evaluation and then to restore it, once the sub-computation ends. This mechanism is needed to avoid the production of illegal or unwanted Erlang expressions. We refer to [40] for a discussion on why it is needed. Elements of S are either *evaluation contexts* $C[\_]$, namely expressions with a hole, or pairs composed of an environment and an evaluation context. We denote as $el : S$ a stack with el as first element and S as rest of the stack.

Definition 2 (System)**.** *A system is a tuple $\Gamma; \Pi$. The global mailbox, Γ, is a set of messages of the form $(sender\_pid, receiver\_pid, payload)$. Given a global mailbox Γ we indicate a new mailbox including also (p, p', m) by $\Gamma \cup \{(p, p', m)\}$. The pool of running processes, Π, is denoted by an expression of the form*

$$\langle p_1, \theta_1, e_1, S_1\rangle \mid \ldots \mid \langle p_n, \theta_n, e_n, S_n\rangle$$

where "|" is an associative and commutative parallel operator.

The semantics in [40] is defined in a modular way, similarly to the ones presented in [39, 41]: there is a semantics for the expression level and one for the system level. This approach simplifies the design of the reversible semantics since only the system one needs to be made reversible. The expression semantics is defined as a labeled transition relation, where the label describes side-effects (e.g., creation of a message) or requests of information to the system level. In the latter case, the label contains the symbol κ, which is a placeholder for the result of the evaluation, and is not known at the expression level. The system rules will replace κ with the correct value.

The semantics is a classical call-by-value semantics for a higher-order language. The labels used in the expression semantics are:

- τ, denoting the evaluation of a (sequential) expression without side-effects;
- send(v_1, v_2), requiring the sending of a message, where v_1 and v_2 represent the pid of the sender and the value of the message respectively;

$$\textsc{Var}\ \frac{}{\theta, C[X], S \xrightarrow{\tau} \theta, C[\theta(X)], S}$$

$$\textsc{If}\ \frac{\mathsf{eval\_guard}(g_1\theta, \ldots, g_n\theta) = i}{\theta, C[\mathsf{if}\ g_1\ \rightarrow\ e_1; \ldots; g_n\ \rightarrow\ e_n\ \mathsf{end}], S \xrightarrow{\tau} \theta, e_i, C[\_] : S}$$

$$\textsc{Case}\ \frac{\mathsf{match\_case}(\theta, v, cl_1, \ldots, cl_n) = \langle \theta_i, e_i \rangle}{\theta, C[\mathsf{case}\ v\ \mathsf{of}\ cl_1; \ldots; cl_n\ \mathsf{end}], S \xrightarrow{\tau} \theta\theta_i, e_i, C[\_] : S}$$

$$\textsc{Seq1}\ \frac{}{\theta, C[v, e], S \xrightarrow{\tau} \theta, C[e], S} \quad \textsc{Seq2}\ \frac{}{\theta, v, C[\_] : S \xrightarrow{\tau} \theta, C[v], S}$$

$$\textsc{Match}\ \frac{\mathsf{match}(pat\theta, v) = \sigma}{\theta, C[pat = v], S \xrightarrow{\tau} \theta\sigma, C[v], S}$$

$$\textsc{Op}\ \frac{\mathsf{eval}(op, v_1, \ldots, v_n) = v}{\theta, C[op\ (v_1, \ldots, v_n)], S \xrightarrow{\tau} \theta, C[v], S}$$

$$\textsc{Fun1}\ \frac{}{\theta, C[\mathsf{fun} fun_1; \ldots; fun_m \mathsf{end}], S \xrightarrow{\tau} \theta, C[\langle \theta, \mathsf{fun} fun_1; \ldots; fun_m \mathsf{end} \rangle], S}$$

$$\textsc{Fun2}\ \frac{\mathsf{def}(f/n, \mathsf{P_{Mod}}) = \mathsf{fun} fun_1; \ldots; fun_m \mathsf{end}}{\theta, C[Mod : f(v_1, \ldots, v_n)], S \xrightarrow{\tau} \sigma, \langle \theta, \mathsf{fun} fun_1; \ldots; fun_m \mathsf{end} \rangle(v_1, \ldots, v_n), S}$$

$$\textsc{Call}\ \frac{\mathsf{match\_fun}((v_1, \ldots, v_n), \langle \theta', \mathsf{fun} fun_1; \ldots; fun_m \mathsf{end} \rangle) = (\sigma, e)}{\theta, C[\langle \theta', \mathsf{fun} fun_1; \ldots; fun_m \mathsf{end} \rangle(v_1, \ldots, v_n)], S \xrightarrow{\tau} \sigma, e, (\theta, C[\_]) : S}$$

$$\textsc{Return}\ \frac{}{\sigma, v, (\theta, C[\_]) : S \xrightarrow{\tau} \theta, C[v], S}$$

Figure 2.2: Semantics of sequential expressions

- $\mathsf{rec}(\kappa, \overline{cl_n})$, requiring to receive a message matching one of the n clauses $\overline{cl_n}$ (we use overline to denote tuples) of a receive expression; at the system level, the expresssion of the matching clause will replace κ;
- $\mathsf{spawn}(\kappa, exprs)$, requiring to spawn a process to execute the expression $exprs$. At the system level, κ will be replaced by the pid of the spawned process.

We divide the rules of the expression level semantics in two sets: the set of sequential expressions, depicted in Fig. 2.2, and the set of concurrent expressions, depicted in Fig. 2.3.

The rules for sequential expressions define the behavior of the constructs of the language without side-effects, in fact we label all the transitions with τ, since we do not need to distinguish them in the system semantics. Now we briefly describe these rules. Rule Var evaluates a variable by searching its value in the environment θ. A possibly empty *evaluation context* C is used to find the redex in a larger term according to an eager semantics. Rule If evaluates the guards of a conditional using the auxiliary function eval_guard, which returns the index of the first guard that evaluates to true, then pushes the current context to the stack, to avoid giving rise to an illegal expression if e_i is a sequence. Rule Case matches the argument of the case v against the case clauses (possibly including guards) using the auxiliary function match_case. It returns a pair (θ_i, e_i) with the matching substitution θ_i and the expression e_i in the selected branch. As before, the context is pushed to the stack. Rule Seq1 removes the first value from a sequence; in this way, it is possible to evaluate the following elements. Indeed, the result of the evaluation of a sequence of expressions is the evaluation of the last element. When a single value is left,

$$\textsc{Send}\ \frac{}{\theta, C[v_1\ !\ v_2], S \xrightarrow{\mathsf{send}(v_1,v_2)} \theta, C[v_2], S}$$

$$\textsc{Receive}\ \frac{}{\theta, C[\mathsf{receive}\ cl_1; \ldots; cl_n\ \mathsf{end}], S \xrightarrow{\mathsf{rec}(\kappa,\overline{cl_n})} \theta, \kappa, \mathsf{seq}(C[\_]) : S}$$

$$\textsc{Spawn1}\ \frac{exprs = mod : f(\overline{v_n})}{\theta, C[\mathsf{spawn}(mod, f, [\overline{v_n}])], S \xrightarrow{\mathsf{spawn}(\kappa,exprs)} \theta, C[\kappa], S}$$

$$\textsc{Spawn2}\ \frac{}{\theta, C[\mathsf{spawn}(\mathsf{fun}() \to exprs\ \mathsf{end})], S \xrightarrow{\mathsf{spawn}(\kappa,exprs)} \theta, C[\kappa], S}$$

$$\textsc{Self}\ \frac{}{\theta, C[\mathsf{self}()], S \xrightarrow{\mathsf{self}(\kappa)} \theta, C[\kappa], S}$$

Figure 2.3: Semantics of concurrent expressions

rule Seq2 recovers from the stack the context where the evaluation needs to be performed, if any. Rule Match evaluates a pattern matching equation using auxiliary function match, which returns the resulting substitution σ. The Op rule is used to evaluate arithmetic and relational operators, using the auxiliary function eval. Rule Fun1 evaluates an anonymous function by reducing it to a closure. Rule Fun2 builds a closure from a named function and its parameters, by joining the current environment and the function code $\mathsf{P_{Mod}}$ taken from its module. The closure is immediately applied to parameters. Rule Call evaluates a closure by moving the current context and environment to the stack and then reducing the function body using the auxiliary function match_fun. This function takes as arguments the parameters of the function call as well as a closure. Rule Return recovers the context and environment from the stack once the function body is reduced to a value.

Fig. 2.3 shows the semantics of concurrent expressions. The system semantics will use the labels from the expression level to execute the associated side-effect or to provide the necessary information.

Rule Send, used to send a message to a process, reduces the expression $v_1!v_2$ to v_2. The side-effect, specified by the label $\mathsf{send}(v_1, v_2)$, is that the message v_2 is sent, i.e., the message will be added to the global mailbox Γ at the system level. Rule Receive, used to receive a message, returns a placeholder, κ, since the resulting value of the receive expression is not known at this level, since it depends on the global mailbox. Like before, we label the step with enough information for rule Receive at the system level to complete the reduction. Rules Spawn1 and Spawn2, used to spawn a process, return a placeholder, κ, that will be bound to the pid of the spawned process at the system level. Rules Spawn1 deals with named functions and rule Spawn2 with anonymous ones. Rule Self returns a placeholder, κ, as well, that the system rule Self binds with the pid of the current process.

In Fig. 2.4 we show the system rules. Rule Seq is used for expressions without side-effects, and lifts the corresponding step at the system level. We apply rule Send when a process performs a send and, as a side-effect, we update Γ by adding the triplet (p, p', m), where p is the pid of the sender, p' of the receiver and m is the sent message. Rule Receive nondeterministically selects from the global mailbox Γ a message that matches one of the n clauses (thanks to the auxiliary function matchrec). Then, it updates the process' environment θ' with the new

$$\textsc{Seq}\ \frac{\theta, e, S \xrightarrow{\tau} \theta', e', S'}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi \hookrightarrow \Gamma; \langle p, \theta', e', S'\rangle \mid \Pi}$$

$$\textsc{Send}\ \frac{\theta, e, S \xrightarrow{\mathsf{send}(p', v)} \theta', e', S'}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi \hookrightarrow \Gamma \cup \{(p, p', v,)\}; \langle p, \theta', e', S'\rangle \mid \Pi}$$

$$\textsc{Receive}\ \frac{\theta, e, S \xrightarrow{\mathsf{rec}(\kappa, \overline{cl_n})} \theta', e', S' \quad \mathsf{matchrec}(\overline{cl_n}\theta, v) = (\theta_i, e_i)}{\Gamma \cup (\{p, p', v\}); \langle p, \theta, e, S\rangle \mid \Pi \hookrightarrow \Gamma; \langle p, (\theta'\theta_i, e'\{\kappa \mapsto e_i\}), S'\rangle \mid \Pi}$$

$$\textsc{Spawn}\ \frac{\theta, e, S \xrightarrow{\mathsf{spawn}(\kappa, exprs)} \theta', e', S' \quad p' \text{ is a fresh pid}}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi \hookrightarrow \Gamma; \langle p, (\theta', e'\{\kappa \mapsto p'\}), S'\rangle \mid \langle p', (), (id, exprs), ()\rangle \mid \Pi}$$

$$\textsc{Self}\ \frac{\theta, e, S \xrightarrow{\mathsf{self}(\kappa)} \theta', e', S'}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi \hookrightarrow \Gamma; \langle p, (\theta', e'\{\kappa \mapsto p\}), S'\rangle \mid \Pi}$$

Figure 2.4: System semantics

variables introduced by the match (from θ_i), replaces κ with the corresponding expression e_i, and removes the message from Γ. Rule Spawn evaluates the spawn expression, it chooses a fresh identifier p as the pid of the new process, replaces κ with p and finally adds to Π the new process.

Finally, rule Self is used for the self BIF, which evaluates to the current process pid.

2.1.3 Reversible Semantics

Two relations describe the reversible semantics: one computes forwards ($\rightharpoonup$) and one backwards ($\leftharpoondown$). The former extends the system semantics using a *Landauer embedding* [42], namely by executing the process expression while tracking enough *history information* so to be able to recover past states.

The backward relation proceeds in the opposite direction and allows us to undo an action, namely recovering the previous state from the current one, by ensuring Causal Consistency. Causal Consistency ensures that before undoing an action all its consequences have been undone.

Syntactically, every process is extended with a history, denoted with h, which stores the information needed in the backward semantics to undo an action.

Definition 3 ((Reversible) process)**.** *A (reversible) process is a tuple $\langle p, h, \theta, e, S\rangle$, where p is the process pid, h is the process history, θ is the process environment, e is the expression under evaluation and S is the stack of process environments. The process history is a stack of* history items.

The definition of a system needs to be updated accordingly.

Definition 4 ((Reversible) system)**.** *A (reversible) system is a tuple $\Gamma; \Pi$. The global mailbox Γ is a set of messages of the form $(sender\_pid, receiver\_pid, \{payload, \lambda\})$, where λ is a unique identifier. The pool of running reversible processes is denoted by Π.*

In reversible systems, messages also carry unique identifiers, denoted with λ, without which messages with the same value could not be distinguished. This choice is discussed in [39].

$$\textsc{Seq}\ \frac{\theta, e, S \xrightarrow{\tau} \theta', e', S'}{\Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi; \rightharpoonup \Gamma; \langle p, \tau(\theta, e, S){:}h, \theta', e', S'\rangle \mid \Pi}$$

$$\textsc{Send}\ \frac{\theta, e, S \xrightarrow{\mathsf{send}(p', v)} \theta', e', S' \quad \lambda \text{ is a fresh identifier}}{\Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi \rightharpoonup \Gamma \cup \{(p, p', \{v, \lambda\})\}; \langle p, \mathbf{send}(\theta, e, S, \{v, \lambda\}){:}h, \theta', e', S'\rangle \mid \Pi}$$

$$\textsc{Receive}\ \frac{\theta, e, S \xrightarrow{\mathsf{rec}(\kappa, \overline{cl_n})} \theta', e', S' \quad \mathsf{matchrec}(\overline{cl_n}\theta, v) = (\theta_i, e_i)}{\Gamma \cup (\{p, p', \{v, \lambda\}\}); \langle p, h, \theta, e, S\rangle \mid \Pi \rightharpoonup \Gamma; \langle p, \mathbf{rec}(\theta, e, S, p', \{v, \lambda\}){:}h, (\theta'\theta_i, e'\{\kappa \mapsto e_i\}), S'\rangle \mid \Pi}$$

$$\textsc{Spawn}\ \frac{\theta, e, S \xrightarrow{\mathsf{spawn}(\kappa, exprs)} \theta', e', S' \quad p' \text{ is a fresh pid}}{\Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi \rightharpoonup \Gamma; \langle p, \mathbf{spawn}(\theta, e, S, p'){:}h, (\theta', e'\{\kappa \mapsto p'\}), S'\rangle \mid \langle p', (), (id, exprs), ()\rangle \mid \Pi}$$

$$\textsc{Self}\ \frac{\theta, e, S \xrightarrow{\mathsf{self}(\kappa)} \theta', e', S'}{\Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi \rightharpoonup \Gamma; \langle p, \mathbf{self}(\theta, e, S){:}h, (\theta', e'\{\kappa \mapsto p\}), S'\rangle \mid \Pi}$$

Figure 2.5: Forward reversible semantics

In Fig. 2.5 we show the forward reversible semantics, which extends the system semantics (cf. Fig. 2.4) with history information. In the semantic rules (from now on) we highlight the history in red.

The history is composed of history items, to distinguish the last rule executed by a process and track the related information. The history items introduced in [40] are:

$$\{\tau(\theta, e, S), \mathbf{send}(\theta, e, S, \{v, \lambda\}), \mathbf{rec}(\theta, e, S, p, \{v, \lambda\}), \mathbf{spawn}(\theta, e, S, p), \mathbf{self}(\theta, e, S)\}$$

Each history item carries the information needed to undo the corresponding action. For rules Seq and Self it is enough to save θ, e and S, so that they can be restored. For the other rules, we must carry additional information, to allow one to check that every consequence has been undone. These pieces of information are:

- for the send, the message sent $\{v, \lambda\}$, to check that it is still available in Γ;
- for the receive, the message received $\{v, \lambda\}$ and the pid p of the process that sent it, to restore the message in Γ;
- for the spawn, the pid p of the spawned process, to ensure all its actions have been undone so the spawned process can be removed.

Fig. 2.6 depicts the uncontrolled backward semantics. The backward semantics restores previous states of a process computation if all of the consequences of the target action have been undone. Rules $\overline{\textsc{Seq}}$ and $\overline{\textsc{Self}}$ can always be applied since they never have consequences. We apply rule $\overline{\textsc{Send}}$ when a message is in Γ because in this case we are sure that the receiver has not taken it yet. Rule $\overline{\textsc{Receive}}$ puts back a received message in the global mailbox Γ and can always be applied. Finally, rule $\overline{\textsc{Spawn}}$ can be applied when the child p' has an empty history, since this means it has not taken any step yet.

$\overline{\textsc{Seq}}\quad \Gamma; \langle p, \tau(\theta, e, S){:}h, \theta', e', S'\rangle \mid \Pi \leftharpoondown \Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi$

$\overline{\textsc{Send}}\quad \Gamma \cup \{(p, p', \{v, \lambda\})\}; \langle p, \mathbf{send}(\theta, e, S, \{v, \lambda\}){:}h, \theta', e', S'\rangle \mid \Pi \leftharpoondown \Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi$

$\overline{\textsc{Receive}}\quad \Gamma; \langle p, \mathbf{rec}(\theta, e, S, p', \{v, \lambda\}){:}h, \theta', e', S'\rangle \mid \Pi \leftharpoondown \Gamma \cup \{(p, p', \{v, \lambda\})\}; \langle p, h, \theta, e, S\rangle \mid \Pi$

$\overline{\textsc{Spawn}}\quad \Gamma; \langle p, \mathbf{spawn}(\theta, e, S, p'){:}h, \theta', e', S'\rangle \mid \langle p', (), (id, e''), ()\rangle \mid \Pi \leftharpoondown \Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi$

$\overline{\textsc{Self}}\quad \Gamma; \langle p, \mathbf{self}(\theta, e, S){:}h, \theta', e', S'\rangle \mid \Pi \leftharpoondown \Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi$

Figure 2.6: Backward reversible semantics

$\overline{\textsc{Seq}}\quad \Gamma; \langle p, \tau(\theta, e, S){:}h, \theta', e', S'\rangle \mid \Pi \leftharpoondown_{p,\textsc{Seq},\{s\}} \Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi$

$\overline{\textsc{Send}}\quad \Gamma \cup \{(p, p', \{v, \lambda\})\}; \langle p, \mathbf{send}(\theta, e, S, p', \{v, \lambda\}){:}h, \theta', e', S'\rangle \mid \Pi \leftharpoondown_{p,\textsc{Send}\{s,\mathsf{send}(\lambda)\}} \Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi$

$\overline{\textsc{Receive}}\quad \Gamma; \langle p, \mathbf{rec}(\theta, e, S, p', \{v, \lambda\}){:}h, \theta', e', S'\rangle \mid \Pi \leftharpoondown_{p,\textsc{Receive},\{s,\mathsf{rec}(\lambda)\}} \Gamma \cup \{(p, p', \{v, \lambda\})\}; \langle p, h, \theta, e, S\rangle \mid \Pi$

$\overline{\textsc{Spawn}}\quad \Gamma; \langle p, \mathbf{spawn}(\theta, e, S, p'){:}h, \theta', e', S'\rangle \mid \langle p', (), (id, e''), ()\rangle \mid \Pi \leftharpoondown_{p,\textsc{Spawn},\{s,\mathsf{spawn}(p')\}} \Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi$

$\overline{\textsc{Self}}\quad \Gamma; \langle p, \mathbf{self}(\theta, e, S){:}h, \theta', e', S'\rangle \mid \Pi \leftharpoondown_{p,\textsc{Self},\{s\}} \Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi$

Figure 2.7: Extended backward reversible semantics

2.1.4 Rollback Semantics

Now we introduce a rollback semantics, which undoes the execution of a past action, selected by the user of the debugger and possibly far in the past of the computation, by undoing all and only its consequences in an automatic manner. Rollback is handy when debugging a faulty program to follow causal links backwards from a misbehavior to its causes. We describe the approach following [40, 41, 38].

A rollback is formalized as a sequence of steps derived using the backward semantics, driven by the target action we want to undo. We denote a system in the rollback mode with $\llbracket \mathcal{S} \rrbracket_{\{p,\psi\}}$, where we want to start a backward derivation to undo the action ψ performed by the process p; to do so we want to undo all and only the actions that depend on ψ.

More generally, we consider systems $\llbracket \mathcal{S} \rrbracket_{\Psi}$, where Ψ is the sequence of undo requests that need to be satisfied; Ψ can be seen as a stack where the first element is the most recent request and once the stack is empty the action selected by the user has been undone. Stack elements $\{p, \psi\}$ can require to undo the following actions:

- $\{p, s\}$: the last action of the process p;
- $\{p, \mathsf{rec}(\lambda)\}$: the receive of the message uniquely identified by λ;
- $\{p, \mathsf{send}(\lambda)\}$: the send of the message uniquely identified by λ;
- $\{p, \mathsf{spawn}(p')\}$: the spawn of process p';
- $\{p, \mathsf{sp}\}$: the action of the process p up to the point immediately after its creation.

$$\textsc{U-Act}\ \frac{\mathcal{S} \leftharpoondown_{p,r,\Psi'} \mathcal{S}' \ \wedge\ \psi \notin \Psi'}{[\![\mathcal{S}]\!]_{\{p,\psi\}:\Psi} \leftsquigarrow [\![\mathcal{S}']\!]_{\{p,\psi\}:\Psi}} \qquad \textsc{U-Satisfy}\ \frac{\mathcal{S} \leftharpoondown_{p,r,\Psi'} \mathcal{S}' \ \wedge\ \psi \in \Psi'}{[\![\mathcal{S}]\!]_{\{p,\psi\}:\Psi} \leftsquigarrow [\![\mathcal{S}']\!]_{\Psi}}$$

$$\textsc{Request-Send}\ \frac{\mathcal{S} = \Gamma; \langle p, \mathbf{send}(\theta, e', S, p', \{v, \lambda\}){:}h, \theta, e, S\rangle \mid \Pi \ \wedge\ \mathcal{S} \not\leftharpoondown_{p,r,\Psi'}}{[\![\mathcal{S}]\!]_{\{p',\psi\}:\Psi} \leftsquigarrow [\![\mathcal{S}]\!]_{\{p',\mathsf{rec}(\lambda)\}:\{p,\psi\}:\Psi}}$$

$$\textsc{Request-Spawn}\ \frac{\mathcal{S} = \Gamma; \langle p, \mathbf{spawn}(\theta, e, S, p'){:}h, \theta, e, S\rangle \mid \Pi \ \wedge\ \mathcal{S} \not\leftharpoondown_{p,r,\Psi'}}{[\![\mathcal{S}]\!]_{\{p,\psi\}:\Psi} \leftsquigarrow [\![\mathcal{S}]\!]_{\{p',\mathsf{sp}\}:\{p,\psi\}:\Psi}}$$

$$\textsc{Satisfy-Spawn}\ \frac{}{[\![\Gamma; \langle p, [\,], \theta, e, S\rangle \mid \Pi]\!]_{\{p',\mathsf{sp}\}:\Psi} \leftsquigarrow [\![\Gamma; \langle p, [\,], \theta, e, S\rangle \mid \Pi]\!]_{\Psi}}$$

Figure 2.8: Rollback semantics

Note that all rollback requests are satisfied by a unique action in the past execution, thanks to the use of unique identifiers for processes and messages. This ensures that requests from the user are not ambiguous.

The requests satisfied by a transition are highlighting in the extended backwards semantic show in Fig. 2.7, where in a transition $s \rightleftharpoons_{p,r,\Psi'} s_1$ where Ψ' is the set of the requests that the transition satisfy.

Fig. 2.8 depicts the rollback semantics, whose steps execute selected transitions of the underlying backward semantics. Rules U-Act and U-Satisfy both lift single steps of the backward semantics. In the former, the topmost request $\{p, \psi\}$ is not satisfied (meaning that the last action performed by p is not the one we are looking for and we need to dig deeper in its history), hence it remains pending. In the latter, the request is satisfied hence it is removed. The Request-Send rule is used to add a request to undo the receiving of a message whose sending we want to undo, while the Request-Spawn rule is used to undo the actions of a process whose spawning we want to undo. Rule Satisfy-Spawn is fired when a process has reached its initial state (with an empty history), and the request $\{p, \mathsf{sp}\}$ can be removed. In this last case, the process p will actually be removed from the system when a request of the form $\{p', \mathsf{spawn}(p)\}$ is on top of the stack.

2.1.5 Replay Semantics

Previously we already argued that reversibility is an effective technique to debug programs, nonetheless, it may still be difficult to (re)create faulty executions. A bug can arise at any point in time, possibly very late in a program execution, hence reproducing it manually would be time consuming. Even worse, the occurrence of some bugs, known as Heisenbugs [43], depends on the outcome of some races in the program, hence they show up only for some results of these races. Reproducing the same computation is difficult and error-prone, since each wrong choice would require to go back and try another possible execution.

Ideally, one would like to be able to log a program execution showcasing a bug and then debug it by analyzing the logged computation. The log is useful for two reasons: first, once the faulty behavior is captured one can always reproduce it; second, once the bug has been solved one can re-try the same execution (provided that the concurrency structure of the code has not changed) to ensure that the problem has really been removed.

$$\textsc{Seq}\ \frac{\theta, e, S \xrightarrow{\tau} \theta', e', S'}{\Gamma; \langle p, \omega, h, \theta, e, S\rangle \mid \Pi \twoheadrightarrow_{p,\textsc{Seq},\{s\}} \Gamma; \langle p, \omega, \tau(\theta, e, S){:}h, \theta', e', S'\rangle \mid \Pi}$$

$$\textsc{Self}\ \frac{\theta, e, S \xrightarrow{\mathsf{self}(\kappa)} \theta', e', S'}{\Gamma; \langle p, \omega, h, \theta, e, S\rangle \mid \Pi \twoheadrightarrow_{p,\textsc{Self},\{s\}} \Gamma; \langle p, \omega, \mathbf{self}(\theta, e, S){:}h, (\theta', e'\{\kappa \mapsto p\}), S'\rangle \mid \Pi}$$

$$\textsc{Receive}\ \frac{\theta, e, S \xrightarrow{\mathsf{rec}(\kappa, \overline{cl_n})} \theta', e', S' \quad \mathsf{matchrec}(\overline{cl_n}\theta, v) = (\theta_i, e_i)}{\Gamma \cup (\{p, p', \{v, \lambda\}\}); \langle p, \mathbf{rec}(\lambda){:}\omega, h, \theta, e, S\rangle \mid \Pi \twoheadrightarrow_{p,\textsc{Receive},\{s,\mathsf{rec}(\lambda)\}} \Gamma; \langle p, \omega, \mathbf{rec}(\theta, e, S, p', \{v, \lambda\}){:}h, (\theta'\theta_i, e'\{\kappa \mapsto e_i\}), S'\rangle \mid \Pi}$$

$$\textsc{Spawn}\ \frac{\theta, e, S \xrightarrow{\mathsf{spawn}(\kappa, exs)} \theta', e', S' \quad p' \text{ is a fresh pid} \qquad \omega' = \mathcal{L}(d, p')}{\Gamma; \langle p, \mathbf{spawn}(p'){:}\omega, h, \theta, e, S\rangle \mid \Pi \twoheadrightarrow_{p,\textsc{Spawn},\{s,\mathsf{spawn}(p')\}} \Gamma; \langle p, \omega, \mathbf{spawn}(\theta, e, S, p'){:}h, (\theta', e'\{\kappa \mapsto p'\}), S'\rangle \mid \langle p', \omega', (), id, exs, ()\rangle \mid \Pi}$$

$$\textsc{Send}\ \frac{\theta, e, S \xrightarrow{\mathsf{send}(p', v)} \theta', e', S'}{\Gamma; \langle p, \mathbf{send}(\lambda){:}\omega, h, \theta, e, S\rangle \mid \Pi \twoheadrightarrow_{p,\textsc{Send},\{s,\mathsf{send}(\lambda)\}} \Gamma \cup \{(p, p', \{v, \lambda\})\}; \langle p, \omega, \mathbf{send}(\theta, e, S, p', \{v, \lambda\}){:}h, \theta', e', S'\rangle \mid \Pi}$$

Figure 2.9: Forward semantics with logs and histories: the standard primitives

To tackle this problem, we now introduce a logging semantics that creates a log from the execution of a given program and a causal-consistent replay semantics that replays an execution following the log. Similarly to the backward semantics, the causal-consistent replay semantics does not oblige the user to replay the execution following the same temporal order as the original one, but only to follow a causal-equivalent execution.

Relation $\hookrightarrow_{p,\ell}$ defines the logging semantics, which decorates the standard semantics with subscripts $_{p,\ell}$ denoting that item ℓ needs to be stored in the log of process p, as described below.

The rules for standard primitives can be obtained from the ones in Fig. 2.5 by adding subscripts $_{p,\ell}$ where p is the process executing the action and ℓ is $\mathbf{send}(\lambda)$, $\mathbf{rec}(\lambda)$, $\mathbf{spawn}(p')$, $\mathbf{seq}$, and $\mathbf{self}$ for rules Send, Receive, Spawn, Seq, and Self, respectively. See [38] for more details.

Definition 5 (Log)**.** *A log is a (finite) sequence of events $(\ell_1, \ell_2, \dots)$ where each ℓ_i is either $\mathbf{send}(\lambda)$, $\mathbf{rec}(\lambda)$, $\mathbf{spawn}(p)$, with λ a message identifier and p a pid. Logs are ranged over by ω. Given a derivation $d = (s_0 \hookrightarrow_{p_1,\ell_1} s_1 \hookrightarrow_{p_2,\ell_2} \dots \hookrightarrow_{p_n,\ell_n} s_n)$, $n \geq 0$, under the logging semantics, the log associated to a pid p in d, in symbols $\mathcal{L}(d, p)$, is inductively defined as follows:*

$$\mathcal{L}(d, p) = \begin{cases} () & \text{if } n = 0 \\ \ell_1 + \mathcal{L}(s_1 \hookrightarrow^* s_n, p) & \text{if } n > 0,\ p_1 = p \text{ and } \ell_1 \notin \{\mathbf{seq}, \mathbf{self}\} \\ \mathcal{L}(s_1 \hookrightarrow^* s_n, p) & \text{otherwise} \end{cases}$$

The log of d, written $\mathcal{L}(d)$, is defined as $\mathcal{L}(d) = \{(p, \mathcal{L}(d, p)) \mid p \text{ occurs in } d\}$. We sometimes call $\mathcal{L}(d)$ the global log of d to avoid confusion with $\mathcal{L}(d, p)$. Note that $\mathcal{L}(d, p) = \omega$ if $(p, \omega) \in \mathcal{L}(d)$ and $\mathcal{L}(d, p) = ()$ otherwise.

Now we can present the forward and backward semantics of Section 2.1.3 to be driven by a log. The rules of the forward semantics with logs and histories are depicted in Fig. 2.9. Processes in the forward semantics with logs and histories now have the generic shape $\langle p, \omega, h, \theta, e, S\rangle$, and the same for the corresponding backward semantics.

The forward semantics with logs and histories makes sure that the causal order of the

$$\textsc{U-Satisfy}\ \frac{\mathcal{S} \twoheadrightarrow_{p,r,\Psi'} \mathcal{S}' \ \wedge\ \psi \in \Psi'}{[\![\mathcal{S}]\!]_{\{p,\psi\}:\Psi} \rightsquigarrow [\![\mathcal{S}']\!]_{\Psi}} \qquad \textsc{U-Act}\ \frac{\mathcal{S} \twoheadrightarrow_{p,r,\Psi'} \mathcal{S}' \ \wedge\ \psi \notin \Psi'}{[\![\mathcal{S}]\!]_{\{p,\psi\}:\Psi} \rightsquigarrow [\![\mathcal{S}']\!]_{\{p,\psi\}:\Psi}}$$

$$\textsc{Spawn}\ \frac{\mathcal{S} = \Gamma; \Pi \ \wedge\ p \notin \Pi \ \wedge\ p' = \mathrm{parent}(\mathcal{L}(d), p)}{[\![\mathcal{S}]\!]_{\{p,\psi\}:\Psi} \rightsquigarrow [\![\mathcal{S}]\!]_{\{p',\mathsf{spawn}(p)\}:\{p,\psi\}:\Psi}}$$

$$\textsc{Request-Rec}\ \frac{\mathcal{S} = \Gamma; \langle p, \mathbf{rec}(\lambda){:}\omega, h, \theta, e, S\rangle \mid \Pi \ \wedge\ \mathcal{S} \not\twoheadrightarrow_{p,r,\Psi'} \ \wedge\ p' = \mathrm{sender}(\lambda)}{[\![\mathcal{S}]\!]_{\Psi} \rightsquigarrow [\![\mathcal{S}]\!]_{\{p',\mathsf{send}(\lambda)\}:\Psi}}$$

Figure 2.10: Replay semantics

execution present in the log is respected.

The forward semantics with logs and histories is paired with a backward semantics with logs and histories which consumes histories and recreates the log. It coincides with the backward semantics, but for the addition of logs. We refer to [38] for the semantics.

Formally, a system in replay mode is denoted as $[\![\mathcal{S}]\!]_{\{p,\psi\}}$, where $\{p, \psi\}$ is the request for a process p to perform ψ. The replay requests $\{p, \psi\}$ are analogous to the ones we presented in Section 2.1.4 for rollback.

Similarly to rollback, we write $[\![\mathcal{S}]\!]_{\Psi}$ where Ψ is the sequence of requests that need to be satisfied to redo the desired action; Ψ can be seen as a stack where the first element is the first request that will be satisfied by the replay semantics and once the stack is empty the required action has been redone. If the first request cannot be satisfied, then it means that some causes have not been replayed yet, so new requests, aimed at replaying these causes, will be pushed on top of the stack.

Fig. 2.10 depicts the replay semantics, built on top of the forward semantics with logs and histories. Rules U-Satisfy and U-Act are analogous to the corresponding ones in the rollback semantics. The first one applies when the executed forward step satisfies the request on top of the stack, which is thus removed. The second one applies when the target process can execute an action, but it will not satisfy any request. This means that the target action is in the future of the process. Rule Spawn has no counterpart in the rollback semantics. It is fired when the process p that needs to do the target action is not in Π, that is p has yet to be spawned. Hence, we add a request for the parent of p, say p', to spawn it. We use the auxiliary function parent to compute p' from the logs of the derivation. More precisely, $\mathrm{parent}(\mathcal{L}(d), p) = p'$ iff the history item spawning p occurs in $\mathcal{L}(d, p')$. Notably, p' may also not be in Π (hence we need to look for it in the global log $\mathcal{L}(d)$), such a case will be dealt with by recursive applications of rule Spawn. Rule Request-Rec is applied when the target process cannot execute (indicated by $\not\twoheadrightarrow_{p,r,\Psi'}$) because the message λ has not yet been sent. The the request $\{p', \mathsf{spawn}(p')\}$ is adding to the list of the request, we use the auxiliary functions $\mathrm{sender}(\lambda)$ to identify the sender p' of a message λ.

One could base the rollback on the backward semantics with logs and histories, thus re-creating logs while going backwards using rollback. This can be obtained by using in the premises of the rules of the rollback semantics the backward semantics with logs and histories instead of the backward semantics.

$$\textsc{Mutual}\ \frac{H \xrightarrow{(\theta,a)}_{\searrow} H' \qquad P \xrightarrow{(\theta,a)}_{\nearrow} P'}{H; P \xrightarrow{(\theta,a)} H'; P'} \qquad \textsc{Heap}\ \frac{H \xrightarrow{\epsilon}_{\searrow} H'}{H; P \xrightarrow{\epsilon} H'; P} \qquad \textsc{Program}\ \frac{P \xrightarrow{(\theta,\mathsf{pure})}_{\nearrow} P'}{H; P \xrightarrow{(\theta,\mathsf{pure})} H; P'}$$

Figure 2.11: MemModel's generic system model

2.2 Modular Meta-theory for Memory Consistency Models

In this section, we describe the MemModel framework, as outlined in [34], used for formalizing shared-memory programming languages with relaxed memory consistency models.

The core idea of this framework is to divide the semantics of a shared-memory system into two parts: the language semantics, which describes the syntax and semantics of the program, and the memory model, which describes the syntax and semantics of the heap.

The language semantics is represented as:

$$P \xrightarrow{p}_{\nearrow} P'$$

where P represents the pool of threads, each identified by an identifier θ. The label p, called program effects, corresponds to the actions that a thread can perform.

The heap semantics is represented as:

$$H \xrightarrow{h}_{\searrow} H'$$

where H represents the heap. The label h, called heap effects, corresponds to the actions that the heap can perform.

We can now present the system semantics depicted in Fig. 2.11, which merges the language and heap semantics into a complete system model. Rule Mutual is applied when a thread θ performs an action a that is visible to both the program and the heap; the step's effect is the pair (θ, a). If a step by thread θ has no effect on the heap, the step's effect is (θ, pure) and the system uses rule Program. The third type of effect is the empty effect ϵ, corresponding to steps taken by the heap that are invisible to the program; in this case, rule Heap is used.

We refer to [34] for a deeper discussion of the framework and its properties. In Sections 2.2.2 and 2.2.3, we present two examples of memory models described in [34].

In both examples, they consider a language with locks, where the possible operations that a thread may perform include: allocating a location r with initial value v; reading value v from location r; writing value v to location r; spawning a new thread with ID θ; acquiring a lock l; and releasing a lock l, described in Section 2.2.1.

A generic map type is used throughout the examples. Let $M : k \Rightarrow v$ denote a map M from keys k to values v. The empty map is represented by $[\,]$. A default value v_0 for all keys can be set with $[* \mapsto v_0]$. The value stored for key k in map M (if any) is denoted by $M(k)$. To update map M to hold value v for key k, we use $M[k \mapsto v]$. To remove the mapping for key k (if any) from map M, we use $M_{|k}$. The domain of a map M is $\mathsf{dom}(M)$.

2.2.1 Language Semantics

$$
\begin{array}{rrcl}
\text{Expressions} & e & ::= & () \mid r \mid l \mid \lambda x.e \mid x \mid e\ e \mid \mathsf{ref}(e) \mid !e \\
 & & & \mid e := e \mid \mathsf{spawn}(e) \mid \mathsf{acq}(e) \mid \mathsf{rel}(e) \\
\text{Values} & v & ::= & () \mid r \mid l \mid \lambda x.e \\
\text{Optional spawn} & \widehat{e} & ::= & \cdot \mid e \mid e; \widehat{e}
\end{array}
$$

Figure 2.12: Language syntax

$$
\textsc{Beta}\ \frac{}{(\lambda x.e)v \xhookrightarrow{\mathsf{pure}} e[x/v]; \cdot} \qquad
\textsc{Ref}\ \frac{}{\mathsf{ref}\ v \xhookrightarrow{\mathsf{ref}(r,v)} r; \cdot} \qquad
\textsc{Read}\ \frac{}{!r \xhookrightarrow{\mathsf{rd}(r,v)} v; \cdot} \qquad
\textsc{Write}\ \frac{}{r := v \xhookrightarrow{\mathsf{wr}(r,v)} (); \cdot}
$$

$$
\textsc{Acquire}\ \frac{}{\mathsf{acq}(l) \xhookrightarrow{\mathsf{acq}(l)} (); \cdot} \qquad
\textsc{Release}\ \frac{}{\mathsf{rel}(l) \xhookrightarrow{\mathsf{rel}(l)} (); \cdot} \qquad
\textsc{Spawn}\ \frac{}{\mathsf{spawn}\ (e) \xhookrightarrow{\mathsf{sp}(\theta)} (); e} \qquad
\textsc{App1}\ \frac{e_1 \overset{\beta}{\hookrightarrow} e_1'; \widehat{e}}{e_1\ e_2 \overset{\beta}{\hookrightarrow} e_1'\ e_2; \widehat{e}}
$$

$$
\textsc{App2}\ \frac{e_2 \overset{\beta}{\hookrightarrow} e_2'; \widehat{e}}{v\ e_2 \overset{\beta}{\hookrightarrow} v\ e_2'; \widehat{e}} \qquad
\beta = \mathsf{pure}, \mathsf{ref}(r,v), \mathsf{rd}(r,v), \mathsf{wr}(r,v), \mathsf{acq}(l), \mathsf{rel}(l), \mathsf{sp}(\theta)
$$

Figure 2.13: Expression Semantics

We consider a lambda calculus that includes threads, references, and locks. We present the language syntax in Figure 2.12, and Figure 2.13 shows the expression semantics.

The language (Fig. 2.12) is defined using expressions e, which include the empty expression, memory cells (r), locks (l), lambda abstractions, variables (x), function applications ($e\ e$), reference creations ($\mathsf{ref}(e)$), dereferencing ($!e$), assignments ($e := e$), thread creations ($\mathsf{spawn}(e)$), lock acquisitions ($\mathsf{acq}(e)$), and lock releases ($\mathsf{rel}(e)$). Values v can be empty values, reference values, locks, and lambda abstractions.

The language semantics (Fig. 2.13) is delineated by a transition system showing the possible transitions. Rule Beta shows the evaluation of a lambda expression applied to an argument. Rule Ref depicts reference creation, initializing a new reference cell with a given value. Rule Read represents reading from a reference cell, returning the stored value. Rule Write allows to modify a reference cell, replacing its current value with a new one. Rule Acquire models lock acquisition on a shared resource, potentially blocking the current thread until the lock becomes available. Rule Release models releasing a lock on a shared resource, making it available for other threads. Rule Spawn shows the creation of a new thread. Rules App1 and App2 model the evaluation of an expression in a function application.

In Fig. 2.14 we show the program semantics used by the model, where rule PrgmStep explains the evaluation of an action, different from the spawn, and rule PrgmSpawn creates a new thread with a new identifier θ'.

$$\textsc{PrgmStep}\ \frac{P(\theta) = e \qquad e \overset{\alpha}{\hookrightarrow} e'; \cdot}{P \xrightarrow{(\theta,\alpha)} P[\theta \mapsto e']} \qquad \textsc{PrgmSpawn}\ \frac{P(\theta) = e \qquad \theta' \notin \mathsf{dom}(P) \qquad e \xrightarrow{\mathsf{sp}(\theta')} e'; e''}{P \xrightarrow{(\theta,\mathsf{sp}(\theta'))} P[\theta \mapsto e'][\theta' \mapsto e'']}$$

$$\alpha = \mathsf{pure}, \mathsf{ref}(r,v), \mathsf{rd}(r,v), \mathsf{wr}(r,v), \mathsf{acq}(l), \mathsf{rel}(l)$$

Figure 2.14: Threads Semantics

2.2.2 Sequential Consistency Memory Model

In this section we present the concept of sequential consistency memory model, which guarantees global interleaving of memory operations consistent with each thread's program semantics. Fig. 2.15 depicts the semantics of the sequential-consistent heap and it is formed by two components: a store and a lock map, where store is a map from locations to values and a lock map is a map from locks to thread IDs. The Alloc rule is used to create a new location in the store, the Read rule retrieves the value of an existing element in the store, the Write rule modifies the value of an existing element in the store, the Acquire rule is used by a thread to acquire an available lock, and the Release rule is used to release a lock.

$$\textsc{Alloc}\ \frac{r \notin \mathsf{dom}(\sigma)}{(\sigma; L) \xrightarrow{(\theta,\mathsf{ref}(r,v))} (\sigma[r \mapsto v]; L)} \qquad \textsc{Read}\ \frac{\sigma(r) = v}{(\sigma; L) \xrightarrow{(\theta,\mathsf{rd}(r,v))} (\sigma; L)} \qquad \textsc{Write}\ \frac{}{(\sigma; L) \xrightarrow{(\theta,\mathsf{wr}(r,v))} (\sigma[r \mapsto v]; L)}$$

$$\textsc{Acquire}\ \frac{l \notin dom(L)}{(\sigma; L) \xrightarrow{(\theta,\mathsf{acq}(l))} (\sigma; L[l \mapsto \theta])} \qquad \textsc{Release}\ \frac{L(l) = \theta}{(\sigma; L) \xrightarrow{(\theta,\mathsf{rel}(l))} (\sigma; L_{|l})} \qquad \textsc{Spawn}\ \frac{}{(\sigma; L) \xrightarrow{(\theta,\mathsf{sp}(\theta'))} (\sigma; L)}$$

Figure 2.15: Sequential consistency semantics

We refer to [34] for a deeper discussion.

2.2.3 Write Buffer Memory Model

The second memory model presented is the write-buffering model, which implements the Partial Store Order (PSO) memory model [23]. This model relaxes the write/read and write/write program orders, allowing writes to be delayed such that they may actually commit after reads or writes to other locations by the same thread.

Fig. 2.16 depicts the heap semantics of the write-buffering memory models, and is formed by three components: a store, a lock map and a buffer, where the store and the lock map are the same of the causal consistent memory model while buffers are maps from thread ID/location pairs to lists of values:

$$\text{Queues } q ::= \cdot \mid q, v$$
$$\text{Buffers } B : (\theta, r) \Rightarrow q$$

$$\textsc{Alloc}\ \frac{r \notin \mathsf{dom}(\sigma)}{(\sigma; B; L) \xrightarrow{(\theta,\mathsf{ref}(r,v))} (\sigma[r \mapsto v]; B; L)} \qquad \textsc{ReadM}\ \frac{\sigma(r) = v \qquad B(\theta, r) = \cdot}{(\sigma; B; L) \xrightarrow{(\theta,\mathsf{rd}(r,v))} (\sigma; B; L)}$$

$$\textsc{ReadB}\ \frac{B(\theta, r) = q, v}{(\sigma; B; L) \xrightarrow{(\theta,\mathsf{rd}(r,v))} (\sigma; B; L)} \qquad \textsc{Write}\ \frac{B(\theta, r) = q}{(\sigma; B; L) \xrightarrow{(\theta,\mathsf{wr}(r,v))} (\sigma; B[(\theta, r) \mapsto q, v]; L)}$$

$$\textsc{Commit}\ \frac{B(\theta, r) = v, q}{(\sigma; B; L) \xrightarrow{\epsilon} (\sigma[r \mapsto v]; B[(\theta, r) \mapsto q]; L)} \qquad \textsc{Acquire}\ \frac{l \notin dom(L)}{(\sigma; B; L) \xrightarrow{(\theta,\mathsf{acq}(l))} (\sigma; B; L[l \mapsto \theta])}$$

$$\textsc{Release}\ \frac{L(l) = \theta \qquad \forall r. B(\theta, r) = \cdot}{(\sigma; B; L) \xrightarrow{(\theta,\mathsf{rel}(l))} (\sigma; B; L_{|l})} \qquad \textsc{Spawn}\ \frac{\forall r. B(\theta, r) = \cdot}{(\sigma; B; L) \xrightarrow{(\theta,\mathsf{sp}(\theta'))} (\sigma; B; L)}$$

Figure 2.16: Write-buffering semantics

Rules Alloc and Acquire are the same of the sequential consistent memory model. The Read rule has been split into two parts. Rule ReadB states that if the thread has values buffered for that location, the value returned by the read is the most recently buffered value. If the thread's buffer for that location is empty, it sees the store's current value for that location (ReadM). Rule Write buffers the value in the appropriate queue, without updating the global store. Rule Commit, which is an empty transition (seen only by the heap semantics), commits the oldest value for a thread ID/location pair by updating the store. The Release and Spawn forcing the heap to commit any buffered values for the thread before proceeding.

We refer to [34] for a deeper discussion.

2.3 Reversibility and its Properties

We introduce the approach to reversibility introduced in [35]. To study reversibility of programming languages in a setting as general as possible, we start from the notion of labeled transition system with Independence (LTSI). We define first labeled transition systems (LTSs). For these definitions we follow the notation used in [35].

Definition 6 (Labeled transition systems)**.** *A labeled transition system (LTS) is a tuple* $(\mathcal{S}, \Lambda, \rightarrow)$ *where:*

- $\mathcal{S}$ *is a set of states,*
- Λ *is a set of labels,*
- $\rightarrow$*, the labeled transition relation, is a subset of* $\mathcal{S} \times \Lambda \times \mathcal{S}$*.*

We say that there is a transition from state p to state q with label ℓ if $(p, \ell, q) \in \rightarrow$ and we denote it with $p \xrightarrow{\ell} q$.

Definition 7 (LTS with Independence)**.** *We say that* $(\mathcal{S}, \Lambda, \rightarrow, \iota)$ *is a* LTS with Independence *(LTSI) if* $(\mathcal{S}, \Lambda, \rightarrow)$ *is a LTS and* ι *is an irreflexive symmetric binary relation on transitions.*

In many scenarios, the idea of independence aligns with concurrency, however this correlation is not absolute. While concurrency suggests that transitions are independent because they occur in distinct processes, it is important to note that transitions within the same process can also be independent. The relation of independence ι is represented as a set of pairs of transitions. We write $p \xrightarrow{\ell_1} p' \ \iota \ q \xrightarrow{\ell_2} q'$ iff $(p \xrightarrow{\ell_1} p', q \xrightarrow{\ell_2} q') \in \iota$.

When defining a reversible causal-consistent semantics, the initial step is to define the backward semantics. This entails specifying how the system can revert from a given state to a previous state, undoing the effects of previous operations. Then we need to introduce the concept of a reversible LTS, which represents a labeled transition system that can execute transitions in backward direction.

Definition 8. *Given a LTS* $(\mathcal{S}, \Lambda, \rightarrow)$*, let the reverse LTS be* $(\mathcal{S}, \Lambda, \rightsquigarrow)$ *where* $p \xrightsquigarrow{\ell} q$ *iff* $q \xrightarrow{\ell} p$*. It is convenient to combine the two LTSs (forward and reverse) and define the combined LTS to be* $(\mathcal{S}, \Lambda, \rightarrow \cup \rightsquigarrow)$.

We, now, explore the fundamental properties that characterize reversibility according to [35]: Loop Lemma, Square property, Backward Transitions are Independent, Well-Foundedness, Causal equivalence and Causal consistency.

In order to present these properties, we need to introduce some notation, we indicate with $t : p \xrightarrow{\ell} p'$ a forward transition t from p to p' with a label ℓ, and with $\underline{t} : p' \xrightsquigarrow{\ell} p$ the *inverse* of t. Notably, we have $\underline{\underline{t}} = t$. Two transitions are co-initial if they start from the same state, co-final if they end in the same state. We define a computation (or trace) as a sequence of consecutive transitions. The empty computation is denoted by ϵ. We use d to range over computations. We define the inverse of a computation inductively as follows $\underline{d\ t} = \underline{t}\ \underline{d}$.

Definition 9 (Loop Lemma)**.** *Given a combined LTS L we say that L satisfies the Loop Lemma if whenever* $p \xrightarrow{\ell} p'$ *if and only if* $p' \xrightsquigarrow{\ell} p$.

This first property that we have presented, called Loop Lemma, states that each transition can be undone.

Definition 10 (Square Property (SP))**.** *Given a combined LTSI L we say that L satisfies the Square Property if whenever* $t : p \xrightarrow{\ell_1} p_1$*,* $u : p \xrightarrow{\ell_2} p_2$ *with* $t \ \iota \ u$ *then there are co-final transitions* $u' : p_1 \xrightarrow{\ell_2} p_3$ *and* $t' : p_2 \xrightarrow{\ell_1} p_3$

Intuitively, Square Property shows that independent transitions can be executed in any order. It can be seen as a safety check on the notion of independence.

Definition 11 (Backward Transitions are Independent (BTI))**.** *Given a combined LTSI L we say that L satisfies Backward Transitions are Independent if whenever* $t : p \xrightsquigarrow{\ell_1} q$ *and* $t' : p \xrightsquigarrow{\ell_2} q'$*,* $t \neq t'$ *then* $t \iota t'$.

Intuitively, BTI corresponds to the observation that two backward transitions are always independent.

Definition 12 (Well-Foundedness (WF))**.** *Given a combined LTS L we say that L satisfies Well-Foundedness if in L there is no infinite reverse computation, i.e. we do not have p_i (not necessarily distinct) such that $p_{i+1} \xrightarrow{\ell} p_i$ for all $i = 0, 1, \ldots$.*

Intuitively, Well-Foundedness requires backward computations to be finite.

Definition 13 (Causal equivalence)**.** *Let $\approx$ be the smallest equivalence on computations closed under composition and satisfying:*

1.(Swap) *if $t_1 = (p \xrightarrow{\ell_1} p_1)$, $t_2 = (p \xrightarrow{\ell_2} p_2)$ are independent ($t_1 \iota t_2$) and $t_3 = (p_1 \xrightarrow{\ell_2} p_3)$, $t_4 = (p_2 \xrightarrow{\ell_1} p_3)$ then $t_1 t_3 \approx t_2 t_4$;*

2.(Delete) *$t\underline{t} \approx \epsilon$ and $\underline{t}t \approx \epsilon$*

Intuitively, computations are causal equivalent if they only differ by swapping independent transitions and by adding do-undo or undo-redo pairs of transitions.

Definition 14 (Causal Consistency)**.** *Given a combined LTSI L, we say that L is causal consistent if for all the traces r and s co-initial and co-final then $r \approx s$.*

Intuitively, if co-initial computations are co-final then they have the same causal information and can reverse in the same ways: we want computations to reverse in the same ways iff they are causal equivalent.

In [35] the authors introduce a systematic approach to reversible computation in concurrent systems by defining a set of axioms and starting from them prove properties relenting with reversibility. The key result of our interest is that given a LTS where the Loop Lemma, Square property, Backward Transitions are Independent, and Well-Foundedness holds then it is causal consistent.

2.4 Deriving Reversible Semantics

In this section we summarize the framework that automatically derive a causal-consistent reversible semantics starting from a non-reversible one described in [33] and implemented in [44].

To apply the framework, the syntax must be divided into two levels: the lower level, composed of the entities and without any restrictions, and the upper level, which must follow this structure:

$$S ::= P \mid op_n(S_1, \ldots, S_n) \mid \mathbf{0}$$

where P is an entity of the lower level, $op_n(S_1, \ldots, S_n)$ is any n-ary composition operator of the lower level and $\mathbf{0}$ is the empty system. Among the operators we always assume a binary parallel operator $|$.

In [33] $\rightarrowtail$ denotes the relation defining the reduction semantics taken in input. The semantics taken in input needs to have the format described in Fig. 2.17, where rules S-Act and S-Opn are schemas and the semantics may contain any number of instances of them. Rule S-Act allows one to specify interactions between entities, and rule S-Opn describes the behaviour of the composition operator of the lower level. Rule Eqv exploits a structural congruence. Finally, rule Par is used to execute two systems in parallel, it is possible to see that it is an instance of

$$\textsc{S-Act}\ \frac{}{P_1 \mid \cdots \mid P_n \rightarrowtail T[Q_1, \ldots, Q_m]} \qquad \textsc{Eqv}\ \frac{S \equiv S' \qquad S \rightarrowtail S_1 \qquad S_1 \equiv S_1'}{S' \rightarrowtail S_1'}$$

$$\textsc{S-Opn}\ \frac{S_i \rightarrowtail S_i'}{op_n(S_0, \ldots S_i, \ldots, S_n) \rightarrowtail op_n(S_0, \ldots S_i', \ldots, S_n)} \qquad \textsc{Par}\ \frac{S \rightarrowtail S'}{S \mid S_1 \rightarrowtail S' \mid S_1}$$

Figure 2.17: Required structure of the semantics in input; S- rules are schemas

$$\textsc{F-S-Act}\ \frac{j_1, \ldots, j_m \text{ are fresh keys}}{\kappa_1 : P_1 \mid \cdots \mid \kappa_n : P_n \twoheadrightarrow T[j_1 : Q_1, \ldots, j_m : Q_m] \mid [\kappa_1 : P_1 \mid \cdots \mid \kappa_n : P_n; j_1 : \bullet_1, \ldots, j_m : \bullet_m]}$$

$$\textsc{F-S-Opn}\ \frac{R_i \twoheadrightarrow R_i' \qquad (\mathsf{keys}(R_i') \setminus \mathsf{keys}(R_i)) \cap (\mathsf{keys}(R_0, \ldots, R_{i-1}, R_{i+1}, \ldots, R_n))}{op_n(R_0, \ldots R_i, \ldots, R_n) \twoheadrightarrow op_n(R_0, \ldots R_i', \ldots, R_n)}$$

Figure 2.18: Forward rules of the uncontrolled reversible semantics

S-Opn. This is possible since a binary parallel operator is always assumed. We refer to [33] for more details about the rules, the syntax and the structure of the semantics taken in input.

In order to define the causal-consistent reversible extension of a given system, one first needs to extend the forward semantics to keep track of past states. First, the syntax of systems is updated as follows:

$$R ::= \kappa : P \mid op_n(R_1, \ldots, R_n) \mid \mathbf{0} \mid [R; C]$$

$$C ::= T[\kappa_1 : \bullet_1, \ldots, \kappa_m : \bullet_m]$$

In particular, it is possible see that the syntax has been updated with unique keys κ to distinguish identical entities, and with memories $[R; C]$ used to keep track of the parts of the system which have been changed by a computational step. R is the old configuration of the system, and C is a context that acts as link between R and the new configuration of the system.

The relation $\twoheadrightarrow$ defines the forward reversible semantics, and it is defined by the rules in Fig. 2.18, where the non-reversible reduction semantics is updated by decorating the entities with keys and by adding a memory information on each performed step, that connect the old configuration with the new one.

The backward relation is defined by $\rightsquigarrow$, and the semantics is defined by the rules depicted in Fig. 2.19. Let us note that every forward rule has a symmetric backward one. A backward rule can be fired if a memory $\mu = [R; C]$ and the entities tagged with the keys in C are available in the current system, the backward step restores the old configuration R.

The reversible semantics obtained through this approach captures causal dependencies in terms of resources produced and consumed. This because the memory each time entities are rewritten create a causal link. Additionally, the reversible semantics derived from this framework

B-S-Act

$$\frac{\mu = [\kappa_1 : P_1 \mid \cdots \mid \kappa_n : P_n; j_1 : \bullet_1, \ldots, j_m : \bullet_m]}{T[j_1 : Q_1, \ldots, j_m : Q_m] \mid \mu \rightsquigarrow \kappa_1 : P_1 \mid \cdots \mid \kappa_n : P_n}$$

B-S-Opn

$$\frac{R_i' \rightsquigarrow R_i}{op_n(R_0, \ldots R_i', \ldots, R_n) \rightsquigarrow op_n(R_0, \ldots R_i, \ldots, R_n)}$$

Figure 2.19: Backward rules of the uncontrolled reversible semantics

satisfies the properties of the Loop Lemma (see Def. 9), Square Property (SP, see Def. 10), Backward Transition are Independent (BTI, see Def. 11), and Well-Foundedness (WF, see Def. 12).

We refer to [33] for the formal proofs of causal-consistency and of the other properties.

Chapter 3

Reversibility in Erlang: Imperative Constructs

The CauDEr debugger [5], described in [39, 40, 41, 38], applies these ideas to provide a reversible debugger for the functional, concurrent and distributed fragment of the Erlang programming language [15]. CauDEr implements both causal-consistent rollback and causal-consistent replay.

In this chapter, we extend CauDEr and its underlying theory, including rollback and replay, by adding the support for some primitives that were not considered in the previous versions. These primitives, namely **register**, **unregister**, **whereis** and **registered**, provide imperative behaviors inside the Erlang language, whose core is functional. More precisely, they define a map linking process identifiers (pids) to names. They make it possible to add (**register**), delete (**unregister**) and read (**whereis** and **registered**) elements from the map. From the theoretical point of view, supporting these primitives is not trivial since they introduce causal dependencies that are different from those originating from the functional and concurrent fragment of Erlang considered in [39, 31, 40, 38]. In particular, read actions commute, but do not commute with add and delete actions. Such causal dependencies cannot be reliably represented in the general approach to derive reversible semantics for a given language presented in [33] and implemented in [44] because the approach in [33] considers a causal relation based on resources consumed and produced only, and does not support read operations. Similar dependencies are considered in [41] to model the set of nodes in an Erlang network, but this model does not include a delete operation, while we consider one. Similar dependencies are also used in [32] to study operations on shared tuple spaces in the framework of the coordination language Klaim [19], however they only access single tuples, while we also access multiple elements of the map at once or check for the absence of a given element of the map. Also, their work, which is in the context of an abstract calculus, has never been implemented and does not consider replay.

The Chapter is structured as follows. Section 3.1 describes the state of the art. Section 3.2 introduces and describes the imperative primitives. In Section 3.3 we extend the reversible semantics of Erlang to support the imperative primitives mentioned above. In Section 3.4 we prove the main properties of the reversible semantics, in particular its causal-consistency. In Sections 3.5 and 3.6 we describe, respectively, the extensions of causal-consistent rollback and causal-consistent replay. Finally, in Section 3.7 we outline our extension of CauDEr, and we put

it at work on an example.

3.1 State of the Art

While we have discussed many related works, in particular work on reversibility in Erlang at the beginning of the chapter, we mention here some additional related approaches. Indeed, reversibility of imperative languages with concurrency has been considered, e.g., in [45, 46]. There, however, actions are undone (mostly) in reverse order of completion, hence their approach does not fit causal-consistent reversibility. Generation of reversible code is also studied in the area of parallel simulation, see, e.g., [47], but their reversed code is sequential, and concurrency is added on top of it by the simulation algorithm. Also, this thread of research lacks theoretical results.

Beyond CauDEr, the only reversible debugger for actor systems we are aware of is Actoverse [48], which is for Akka-based applications. It provides many relevant features complementary to ours, such as a partial-order graphical representation of message exchanges that would nicely match our causal-consistent approach. On the other side, Actoverse has several limitations. For instance, its facilities to replay bugs, such as message-oriented breakpoints to force specific message interleavings and support for session replay, are more limited than our causal-consistent replay. Furthermore, Actoverse allows one to explore only some states of the computation, such as the ones corresponding to message sending and receiving.

Another interesting related work is [49], where an approach to record and replay for actor languages is introduced. While we concentrate on the theory, they focus on low-level features: dealing with I/O, producing compact logs, etc. Actually, we could adopt some of the ideas in [49] in order to produce more compact logs and thus reduce our instrumentation overhead.

Concerning our replay technique, previous replay in CauDEr was akin to replay techniques for message passing systems, while our extension to deal with imperative primitives is closer to approaches for shared memory systems. Our approach is currently working on single processors, but it makes no use of such an assumption, indeed it could work as well on multiprocessors, and it is closer to approaches used in multiprocessors. According to the taxonomy in a recent survey [50], our approach is a *complete record approach*, that is it keeps enough information to solve all the data races in the same way as in the original execution. The approaches in the literature we are aware of consider accesses to memory, which has a different structure than our map. However, the approaches closer to ours are the ones that instrument each single access to the memory, as in PinPlay [51].

While, as discussed above, CauDEr supports replay, it provides little help to actually generate and identify a trace producing a misbehavior. This may not be trivial since some misbehaviors only occur under some specific interleavings. Tools such as QuickCheck [52] and detectEr [53] allow one to isolate misbehaving traces which can then be analyzed inside CauDEr. One can increase the likelihood of finding misbehaviors which depend on the scheduling by generating rare schedulings using tools such as PULSE [54], which has been developed to find race conditions. Notably, while race conditions are a relevant class of bugs which can be analyzed using CauDEr, CauDEr is more general. E.g., the bug discussed in Section 3.7 does not depend on any race

condition. We remark that all the tools above target Erlang, making future integration a concrete possibility.

3.2 Syntax of the Imperative Primitives

In our extension, atoms and pids are central. In Erlang, a pid can be associated to an atom. Thus, one can refer to a process, e.g., when specifying the target of a send action, using the associated atom instead of the pid ($atom\,!\,expr$). On the one hand, an atom is more meaningful than a pid for a human. On the other hand, this allows one to decide which process plays a given role. E.g., if a process crashes another one can be registered under the same atom so that the replacement is transparent to other processes (provided that they use the atom to interact). All pairs $\langle atom, pid\rangle$ form a map, shared among the processes of the same node (we consider here a single node, we discuss in Section 3.7 how to deal with multiple nodes).

Our extension is based on the syntax in Fig. 2.1, but we add the following built-in functions (BIFs):

- **register**/2 (where /2 denotes the arity): given an atom a and a pid p, it inserts the pair $\langle a, p\rangle$ in the map and returns the atom **true**. If either the atom a or the pid p is already registered, an exception is raised.
- **unregister**/1: given an atom a, it removes the (unique) pair $\langle a, p\rangle$ from the map and returns **true** if the atom a is found, it raises an exception otherwise.
- **whereis**/1: given an atom, it returns the associated pid if it exists, the atom **undefined** otherwise.
- **registered**/0: it returns the list (possibly empty) of all the atoms in the map.

We remark that in this work we will not consider exception handling and propagation. We refer to [33] for a preliminary account on exception management in reversible Erlang.

3.3 Reversible Erlang with Imperative Primitives

Table 3.1 summarizes the semantics used, pointing out the corresponding arrow symbols and referencing where each semantics is defined. This table can serve as a guide to the reader to better understand the meaning and use of each symbol.

3.3.1 Standard Semantics of the Imperative Primitives

We now start extending the semantics of Erlang introduced in previous sections with support for the imperative primitives. According to the official documentation [15], the imperative BIFs discussed in Section 3.2 are implemented in Erlang using request and reply signals between the process and the manager of the map. However, the detailed behavior of such signals is not specified in the Erlang documentation, and even less the interplay between different signals. Thus, to simplify the modeling, we have opted to implement these BIFs as synchronous actions. Intuitively, this choice does not alter the possible behaviors since the behavior visible to Erlang

Semantics	Arrow	Reference
Expression semantics	$\rightarrow$	Fig. 2.2, 2.3 and 3.1
System semantics / Logging semantics	$\hookrightarrow$ or $\hookrightarrow_{pid,event}$	Fig. 3.3 and 3.10
Forward reversible semantics	$\rightharpoonup$ or $\rightharpoonup_{pid,rule,item}$	Fig. 2.5, 3.2 and 3.4
Backward reversible semantics	$\leftharpoondown$ or $\leftharpoondown_{pid,rule,item}$	Fig. 2.6, 3.5 and 3.6
Forward and backward reversible semantics	$\rightleftharpoons$ or $\rightleftharpoons_{pid,rule,item}$ $(=\rightharpoonup \cup \leftharpoondown)$	
Forward reversible semantics with logs and histories	$\twoheadrightarrow$	Fig. 3.11
Backward reversible semantics with logs and histories	$\twoheadleftarrow$	Fig. 3.12
Forward and backward rev. sem. with logs and histories	$\overset{\twoheadrightarrow}{\twoheadleftarrow}$ $(=\twoheadrightarrow \cup \twoheadleftarrow)$	
Rollback semantics	$\leftsquigarrow$	Fig. 3.7
Replay semantics	$\rightsquigarrow$	Fig. 3.13

Table 3.1: Summary of the semantics used

users is determined by the order in which the request messages are processed at the manager. This modeling choice is coherent with the outcomes of our tests on the Erlang implementation.

We begin by providing the updated definition of a *system* (the definition of a process is unchanged).

Definition 15 (System)**.** *A system is a tuple* $\Gamma; \Pi; \mathsf{M}$*. The global mailbox* Γ *and the pool of running reversible processes* Π *are as in Definition 2. The map* M *is a set of registered pairs of atom-pid of the form* $\{\langle a_1, p_1\rangle; \ldots; \langle a_n, p_n\rangle\}$*, where* a_i *are atoms and* p_i *are pids. Given an atom* a*,* M_a *is the set* $\{\langle a_1, p_1\rangle | \langle a_1, p_1\rangle \in \mathsf{M}$ *and* $a = a_1\}$*; given a pid* p*,* M_p *is the set* $\{\langle a_1, p_1\rangle | \langle a_1, p_1\rangle \in \mathsf{M}$ *and* $p = p_1\}$*.*

Sets M_a and M_p contain at most one element because they contain the tuples in M with, respectively, atom a and pid p, and in the map two elements can never have the same atom or pid.

We extend the semantics to deal with maps, hence we still have one level for expressions ($\rightarrow$) and one for systems ($\hookrightarrow$).

To simplify the presentation w.r.t. [40], in Fig. 3.1 we extend rule Op from Fig. 2.2 to deal also with built-in functions. To this end, we extend the operator eval to produce also the label for functions with side-effects. We define eval on them as:

- $\mathsf{eval}(\mathsf{self}) = (\kappa, \mathsf{self}(\kappa))$;
- $\mathsf{eval}(\mathsf{spawn}, \mathsf{fun}() \rightarrow exprs\ \mathsf{end}) = (\kappa, \mathsf{spawn}(\kappa, exprs))$;
- $\mathsf{eval}(\mathsf{spawn}, mod, f, [\overline{v_n}]) = (\kappa, \mathsf{spawn}(\kappa, mod : f(\overline{v_n})))$;
- $\mathsf{eval}(\mathsf{register}, atom, pid) = (\kappa, \mathsf{register}(\kappa, atom, pid))$;
- $\mathsf{eval}(\mathsf{unregister}, atom) = (\kappa, \mathsf{unregister}(\kappa, atom))$;
- $\mathsf{eval}(\mathsf{whereis}, atom) = (\kappa, \mathsf{whereis}(\kappa, atom))$;
- $\mathsf{eval}(\mathsf{registered}) = (\kappa, \mathsf{registered}(\kappa))$.

On sequential expressions eval returns (v, τ), with v the result of the evaluation.

$$\textsc{Op}\ \frac{\mathsf{eval}(op, v_1, \ldots, v_n) = (v, label)}{\theta, C[op\ (v_1, \ldots, v_n)], S \xrightarrow{label} \theta, C[v], S}$$

Figure 3.1: Semantics of function application, revised

$$\textsc{Send}\ \frac{\theta, e, S \xrightarrow{\mathsf{send}(p', v)} \theta', e', S' \quad \lambda \text{ is a fresh identifier}}{\Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \rightharpoonup \Gamma \cup \{(p, p', \{v, \lambda\})\}; \langle p, \mathbf{send}(\theta, e, S, \{v, \lambda\}){:}h, \theta', e', S'\rangle \mid \Pi; \mathrm{M}}$$

Figure 3.2: Forward reversible semantics: sample rule

Thanks to our extension, rule Op in Fig. 3.1 covers all function invocations, including BIFs with side effects, while in [39, 40] each such BIF requires a dedicated rule (like in Fig. 2.3). Furthermore, new BIFs with side effects can be added without changing the expression level, by defining the function eval so to produce the label and the expected output on the new BIFs.

The rules of the expression level are as in Fig. 2.2 and 2.3, but for replacing the Op rule in Fig. 2.2 with the Op rule in Fig. 3.1. Thanks to this change, in Fig. 2.3, we do not need to consider rules Spawn1, Spawn2, and Self because they are subsumed by the Op rule.

We now discuss the standard and the reversible semantics of the system level extended with imperative primitives. According to Definition 15, all systems now include the map, where we store all the registered pairs atom-pid. We highlight in blue the components of the rules related to the map.

Beyond adding the rules for the imperative primitives, we need to extend all the rules of the system level in Fig. 2.5 to include the map. The map has no effect on such rules, hence we just show in Fig. 3.2 below how rule Send is extended. The extension of the other rules is analogous.

Fig. 3.3 shows the rules defining the standard semantics of the system level of the imperative BIFs. The rules are divided into *write rules* (above the line), which modify the map, and *read rules* (below the line), that only read it. This has an impact on their concurrent behavior, as described later on.

Rule RegisterS defines the success case of the register BIF, which adds the tuple $\langle a, p'\rangle$ to the map. The register fails either when the atom a or the pid p' are already used, or when the pid p' refers to a dead process (this is checked by predicate isAlive, decribed below). The failing behavior is described by rule RegisterF. Similarly, for the unregister, the success case corresponds to rule UnregisterS, which removes from the map the (unique) pair atom-pid for a given atom a. The failure case, when there is no pid registered under the atom a, corresponds to rule UnregisterF. Both failure cases replace the current expression with ϵ and the current stack with [ ]. This denotes an uncaught exception (in this chapter we do not consider exception handling, see [33] for a reversible semantics of Erlang supporting it, but not imperative primitives). The predicate isAlive takes a pid p and the pool of running processes and controls that the process with pid p is alive ($\langle p, \theta, e, S\rangle$ with $e \neq \bot$).

Rules SendS and SendF define the behavior of send actions when the receiver is identified with an atom a. The former is fired when the atom a is registered in the map, which results in adding the triplet (p, p', m) to Γ. In the triplet, p is the pid of the sender, p' is the pid of

$$\textsc{RegisterS}\ \frac{\theta, e, S \xrightarrow{\mathsf{register}(\kappa,a,p')} \theta', e', S' \qquad \mathsf{M}_a = \emptyset \qquad \mathsf{M}_{p'} = \emptyset \qquad \mathsf{isAlive}(p', \Pi)}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi; \mathsf{M} \hookrightarrow \Gamma; \langle p, \theta', e'\{\kappa \to \mathsf{true}\}, S'\rangle \mid \Pi; \mathsf{M} \cup \{\langle a, p'\rangle\}}$$

$$\textsc{UnregisterS}\ \frac{\theta, e, S \xrightarrow{\mathsf{unregister}(\kappa,a)} \theta', e', S' \qquad \mathsf{M}_a = \{\langle a, p'\rangle\}}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi; \mathsf{M} \hookrightarrow \Gamma; \langle p, \theta', e'\{\kappa \to \mathsf{true}\}, S'\rangle \mid \Pi; \mathsf{M} \setminus \mathsf{M}_a}$$

$$\textsc{EndUn}\ \frac{e \text{ is a value } \vee\, e = \epsilon \qquad \mathsf{M}_p = \{\langle a, p\rangle\}}{\Gamma; \langle p, \theta, e, [\,]\rangle \mid \Pi; \mathsf{M} \hookrightarrow \Gamma; \langle p, \theta, \bot, [\,]\rangle \mid \Pi; \mathsf{M} \setminus \mathsf{M}_p}$$

$$\textsc{RegisterF}\ \frac{\theta, e, S \xrightarrow{\mathsf{register}(\kappa,a,p')} \theta', e', S' \qquad \mathsf{M}_a \neq \emptyset \vee \mathsf{M}_{p'} \neq \emptyset \vee \neg\ \mathsf{isAlive}(p', \Pi)}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi; \mathsf{M} \hookrightarrow \Gamma; \langle p, \theta, \epsilon, [\,]\rangle \mid \Pi; \mathsf{M}}$$

$$\textsc{UnregisterF}\ \frac{\theta, e, S \xrightarrow{\mathsf{unregister}(\kappa,a)} \theta', e', S' \qquad \mathsf{M}_a = \emptyset}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi; \mathsf{M} \hookrightarrow \Gamma; \langle p, \theta, \epsilon, [\,]\rangle \mid \Pi; \mathsf{M}}$$

$$\textsc{SendS}\ \frac{\theta, e, S \xrightarrow{\mathsf{send}(a,v)} \theta', e', S' \qquad \mathsf{M}_a = \{\langle a, p'\rangle\}}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi; \mathsf{M} \hookrightarrow \Gamma \cup \{(p, p', v)\}; \langle p, \theta', e', S'\rangle \mid \Pi; \mathsf{M}}$$

$$\textsc{SendF}\ \frac{\theta, e, S \xrightarrow{\mathsf{send}(a,v)} \theta', e', S' \qquad \mathsf{M}_a = \emptyset}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi; \mathsf{M} \hookrightarrow \Gamma; \langle p, \theta, \epsilon, [\,]\rangle \mid \Pi; \mathsf{M}}$$

$$\textsc{Whereis1}\ \frac{\theta, e, S \xrightarrow{\mathsf{whereis}(\kappa,a)} \theta', e', S' \qquad \mathsf{M}_a = \{\langle a, p'\rangle\}}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi; \mathsf{M} \hookrightarrow \Gamma; \langle p, \theta', e'\{\kappa \to p'\}, S'\rangle \mid \Pi; \mathsf{M}}$$

$$\textsc{Whereis2}\ \frac{\theta, e, S \xrightarrow{\mathsf{whereis}(\kappa,a)} \theta', e', S' \qquad \mathsf{M}_a = \emptyset}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi; \mathsf{M} \hookrightarrow \Gamma; \langle p, \theta', e'\{\kappa \to \mathsf{undefined}\}, S'\rangle \mid \Pi; \mathsf{M}}$$

$$\textsc{Registered}\ \frac{\theta, e, S \xrightarrow{\mathsf{registered}(\kappa)} \theta', e', S' \qquad \mathsf{registered}(\mathsf{M}) = \mathit{atoms}}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi; \mathsf{M} \hookrightarrow \Gamma; \langle p, \theta', e'\{\kappa \to \mathit{atoms}\}, S'\rangle \mid \Pi; \mathsf{M}}$$

$$\textsc{End}\ \frac{e \text{ is a value } \vee\, e = \epsilon \qquad \mathsf{M}_p = \emptyset}{\Gamma; \langle p, \theta, e, [\,]\rangle \mid \Pi; \mathsf{M} \hookrightarrow \Gamma; \langle p, \theta, \bot, [\,]\rangle \mid \Pi; \mathsf{M}}$$

Figure 3.3: System semantics of the imperative primitives

the receiver (obtained by searching the atom a in the map) and m is the message. Rule SendF instead is fired when the atom a is not registered in the map, resulting in an uncaught exception.

Rules Whereis1, Whereis2 and Registered define the behavior of the respective primitives; these rules read M without modifying it. Rule Registered uses the auxiliary function registered. We define it as: registered(M) $= [a_1, \ldots, a_n]$ if M $= \{\langle a_1, p_1\rangle, \ldots, \langle a_n, p_n\rangle\}$.

Finally, we have two rules dealing with process termination. If the pid p of the process is not registered in the map, rule End changes the expression to $\bot$, denoting a terminated process. Otherwise, rule EndUn applies, additionally removing the pid from the map. Note that the two rules could be merged in a single rule; we keep them separate since their causal behavior and their reversible semantics differ. This will be clarified in the following.

3.3.2 Reversible Semantics of the Imperative Primitives

The definition of the forward semantics poses a number of challenges, due to the need of balancing two conflicting requirements when defining the history information to be stored. On the one hand, we need to keep enough information to be able to define a corresponding backward semantics. This requires to understand when all the consequences of an action have been undone, and to restore the state prior to its execution. On the other hand, we need to avoid storing information allowing one to distinguish computations obtained by only swapping independent actions. This would invalidate the Square Property (Lemma 2 in Section 3.3.2), as we will discuss in Example 5. We first extend the definition of a system.

Definition 16 ((Reversible) system)**.** *A (reversible) system is a tuple* $\Gamma; \Pi; \mathrm{M}$. *The global mailbox* Γ *and the pool of running reversible processes* Π *are as in Definition 4. Each element of the map* M *is a quadruple* $\langle a, p, t, s\rangle$ *where* a *and* p *are as in Definition 15,* t *is the unique identifier for the quadruple and* s *can be either* $\top$ *or* $\bot$.

From now on, when discussing systems, we will implicitly refer to reversible ones, unless otherwise specified. Unique identifiers t are used to distinguish identical tuples registered in the map at different times. For example, if we have two successful pairs of register and unregister operations of the same tuple $\langle a, p\rangle$ of the map, without a unique identifier we would not know which unregister operation is connected to which register. This information is relevant since each instance of $\langle a, p\rangle$ generates a causal link between a register and the corresponding unregister. This justification is similar to the one for unique identifiers λ for messages, discussed in [31]. Notice that even if a map can contain at most one pair $\langle a, p\rangle$, identical elements can exist in the map at different times, as discussed above.

Tuples of the map whose last field is $\top$ match the ones in the standard semantics, we call them *alive* tuples. Those with $\bot$ are *ghost* tuples, namely previously alive tuples that have been removed from the map as a result of a past forward action. We will clarify in Example 2 why they are needed.

Given an atom a, M^a is the set $\{\langle a_1, p_1, t_1, \bot\rangle | \langle a_1, p_1, t_1, \bot\rangle \in \mathrm{M} \text{ and } a_1 = a\}$; similarly, given a pid p, $\mathrm{M}^p = \{\langle a_1, p_1, t_1, \bot\rangle | \langle a_1, p_1, t_1, \bot\rangle \in \mathrm{M} \text{ and } p_1 = p\}$. Dually, from now on, sets M_a and M_p include only alive tuples. We define function kill, which takes a map and sets to $\bot$ the last field of all its tuples.

$$\textsc{RegisterS}\ \frac{\theta, e, S \xrightarrow{\mathsf{register}(\kappa,a,p')} \theta', e', S' \qquad t \text{ fresh} \qquad \mathrm{M}_a = \emptyset \qquad \mathrm{M}_{p'} = \emptyset \qquad \mathsf{isAlive}(p', \Pi)}{\Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \rightharpoonup \Gamma; \langle p, \mathbf{regS}(\theta, e, S, \{\langle a, p', t, \top\rangle\}){:}h, \theta', e'\{\kappa \to \mathsf{true}\}, S'\rangle \mid \Pi; \mathrm{M} \cup \{\langle a, p', t, \top\rangle\}}$$

$$\textsc{UnregisterS}\ \frac{\theta, e, S \xrightarrow{\mathsf{unregister}(\kappa,a)} \theta', e', S' \qquad \mathrm{M}_a = \{\langle a, p', t, \top\rangle\}}{\Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \rightharpoonup \Gamma; \langle p, \mathbf{del}(\theta, e, S, \mathrm{M}_a, \mathrm{M}^a \cup \mathrm{M}^{p'}){:}h, \theta', e'\{\kappa \to \mathsf{true}\}, S'\rangle \mid \Pi; \mathrm{M} \setminus \mathrm{M}_a \cup \mathrm{kill}(\mathrm{M}_a)}$$

$$\textsc{EndUn}\ \frac{e \text{ is a value } \vee\, e = \epsilon \qquad \mathrm{M}_p = \{\langle a, p, t, \top\rangle\}}{\Gamma; \langle p, h, \theta, e, [\,]\rangle \mid \Pi; \mathrm{M} \rightharpoonup \Gamma; \langle p, \mathbf{del}(\theta, e, [\,], \mathrm{M}_p, \mathrm{M}^a \cup \mathrm{M}^p){:}h, \theta, \bot, [\,]\rangle \mid \Pi; \mathrm{M} \setminus \mathrm{M}_p \cup \mathrm{kill}(\mathrm{M}_p)}$$

$$\textsc{RegisterF}\ \frac{\theta, e, S \xrightarrow{\mathsf{register}(\kappa,a,p')} \theta', e', S' \qquad \mathrm{M}_a \neq \emptyset \vee \mathrm{M}_{p'} \neq \emptyset \vee \neg\ \mathsf{isAlive}(p', \Pi)}{\Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \rightharpoonup \Gamma; \langle p, \mathbf{readS}(\theta, e, S, \mathrm{M}_a \cup \mathrm{M}_{p'}){:}h, \theta, \epsilon, [\,]\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{UnregisterF}\ \frac{\theta, e, S \xrightarrow{\mathsf{unregister}(\kappa,a)} \theta', e', S' \qquad \mathrm{M}_a = \emptyset}{\Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \rightharpoonup \Gamma; \langle p, \mathbf{readF}(\theta, e, S, a, \mathrm{M}^a){:}h, \theta, \epsilon, [\,]\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{SendS}\ \frac{\theta, e, S \xrightarrow{\mathsf{send}(a,v)} \theta', e', S' \qquad \lambda \text{ fresh} \qquad \mathrm{M}_a = \{\langle a, p', t, \top\rangle\}}{\Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \rightharpoonup \Gamma \cup \{(p, p', \{v, \lambda\})\}; \langle p, \mathbf{sendS}(\theta, e, S, \{v, \lambda\}, \mathrm{M}_a){:}h, \theta', e', S'\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{SendF}\ \frac{\theta, e, S \xrightarrow{\mathsf{send}(a,v)} \theta', e', S' \qquad \mathrm{M}_a = \emptyset}{\Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \rightharpoonup \Gamma; \langle p, \mathbf{readF}(\theta, e, S, a, \mathrm{M}^a){:}h, \theta, \epsilon, [\,]\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{Whereis1}\ \frac{\theta, e, S \xrightarrow{\mathsf{whereis}(\kappa,a)} \theta', e', S' \qquad \mathrm{M}_a = \{\langle a, p', t, \top\rangle\}}{\Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \rightharpoonup \Gamma; \langle p, \mathbf{readS}(\theta, e, S, \mathrm{M}_a){:}h, \theta', e'\{\kappa \to p'\}, S'\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{Whereis2}\ \frac{\theta, e, S \xrightarrow{\mathsf{whereis}(\kappa,a)} \theta', e', S' \qquad \mathrm{M}_a = \emptyset}{\Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \rightharpoonup \Gamma; \langle p, \mathbf{readF}(\theta, e, S, a, \mathrm{M}^a){:}h, \theta', e'\{\kappa \to \mathsf{undefined}\}, S'\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{Registered}\ \frac{\theta, e, S \xrightarrow{\mathsf{registered}(\kappa)} \theta', e', S' \qquad \mathrm{registered}(\mathrm{M}) = atoms}{\Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \rightharpoonup \Gamma; \langle p, \mathbf{readM}(\theta, e, S, \mathrm{M}){:}h, \theta', e'\{\kappa \to atoms\}, S'\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{End}\ \frac{e \text{ is a value } \vee\, e = \epsilon \qquad \mathrm{M}_p = \emptyset}{\Gamma; \langle p, h, \theta, e, [\,]\rangle \mid \Pi; \mathrm{M} \rightharpoonup \Gamma; \langle p, \mathbf{readF}(\theta, e, [\,], p, \mathrm{M}^p){:}h, \theta, \bot, [\,]\rangle \mid \Pi; \mathrm{M}}$$

Figure 3.4: Forward reversible semantics of the imperative primitives

Forward reversible semantics The forward semantics of the sequential and concurrent primitives is as in Fig. 2.5, but for the introduction of the map M, as discussed in Fig. 3.2 in the sample case of rule SEND.

The forward semantics of the imperative primitives, defined in Fig. 3.4, is a conservative extension of the standard one in which each forward step also produces the corresponding history item. The following history items have been introduced to describe the imperative primitives: **regS**, **readS**, **readF**, **sendS**, **readM**, and **del**. Rules with common characteristics share the same history item, e.g., rule UNREGISTERS and rule ENDUN both remove a tuple from the map, hence both use del as a history item. Using the same history item is not problematic when going backwards, as discussed in the next paragraph.

All the new history items, like the old ones, carry the old state θ, e, S, thus allowing the backward computation to restore it. Furthermore, they carry some additional information to enable us to understand their causal dependencies:

- **regS**, created by rule REGISTERS, carries the tuple inserted in the map;
- **readS**, created by rules REGISTERF and WHEREIS1, carries the tuple(s) read from the map;
- **sendS**, created by rule SENDS, carries the tuple read from the map as well, but also the sent message;
- **readF**, created by rules UNREGISTERF, SENDF, WHEREIS2 and END, carries the atom or the pid which the rule tried to read and the ghost tuples for such atom or pid, if any;
- **readM**, created by rule REGISTERED, carries the whole map read by the rule;
- **del**, created by rules UNREGISTERS and ENDUN, carries the removed tuple and the ghost tuples on the same atom or pid.

The need for such information will be clearer when discussing the backward semantics as well as the concurrency model of the imperative primitives.

Let us now contrast the standard version of rule REGISTERS with its forward reversible version. We remind that the parts highlighted in red are the ones necessary to reverse the execution.

First, we observe that the forward reversible version deals with reversible processes, that is processes equipped with their own history. The map has been integrated with additional elements as well. More precisely, its tuples have been decorated with unique identifiers to distinguish identical pairs atom-pid added at different times, and with a symbol to signal whether a tuple is alive ($\top$) or not ($\bot$). Hence now the map also contains tuples not alive anymore. Then, when reducing, a tuple of the required kind is added to M and, finally, a new history item recording the former state of the process along with the inserted tuple is added to the history of the process. The backward reversible semantics uses this information to reverse the execution.

Similarly, the other forward reversible rules deal with reversible processes and the reversible map and they all produce history items while going forwards.

$$\overline{\textsc{Send}}\quad \Gamma \cup \{(p, p', \{v, \lambda\})\}; \langle p, \mathbf{send}(\theta, e, S, \{v, \lambda\}){:}h, \theta', e', S'\rangle \mid \Pi; \mathrm{M} \leftharpoonup \Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi; \mathrm{M}$$

Figure 3.5: Backward reversible semantics, sample rule

Backward reversible semantics The backward semantics of the sequential and concurrent primitives is as in Fig. 2.6, but for the introduction of the map M, as shown in Fig. 3.5 below in the sample case of rule $\overline{\textsc{Send}}$. As usual the map is in blue.

Fig. 3.6 presents the backward semantics of the imperative primitives. In previous works [41, 40, 31] each forward rule produced a different history item, hence there was one backward rule for each forward rule. Here, since different forward rules produce the same history item, some backward rules cover more than one forward rule. This is possible because the history item contains enough information to correctly reverse forward rules with similar effects. E.g., both rules RegisterF and Whereis1 read information from the map, and the history item tracks the read information. Hence, a single rule can exploit this information to check that the same read information is still available in the map.

Rule $\overline{\textsc{RegisterS}}$ undoes the corresponding forward action, which added a tuple to the map, removing it. To this end, rule $\overline{\textsc{RegisterS}}$ requires that the added tuple, uniquely identified by identifier t, is still in the map (ensuring that possible deletions of the same tuple have been undone) and, as a side condition, that no process performed a read operation on the tuple with unique identifier t. This last condition is checked by the predicate $\mathrm{readop}(t, \Pi)$, which scans the histories of processes in Π looking for such reads.

Rule $\overline{\textsc{Del}}$ undoes either rule UnregisterS or rule EndUn, turning a ghost tuple back into an alive one. Let us discuss its side conditions. The first two conditions require that in M there is no alive tuple on the same atom a or process p'. The third one ensures that no process performed a registered getting M, while the fourth that no process read a ghost tuple with identifier t. Finally, we require ghost tuples on both a and p' to be the same as when the corresponding forward action has been performed. The last condition ensures that rule $\overline{\textsc{Del}}$ will not commute with pairs of operations that add and then delete tuples on the same atom or pid, e.g., a pair register-unregister. This is needed to satisfy the properties described in Section 3.4, such as causal consistency.

Rule $\overline{\textsc{ReadS}}$ reverses rules Whereis1 and RegisterF. The only side condition requires that the tuple(s) read from the map by the forward rule must be alive. Rule $\overline{\textsc{SendS}}$ is analogous, but it also requires that the sent message is in Γ.

Rule $\overline{\textsc{ReadF}}$ undoes actions from rules UnregisterF, SendF, Whereis2, and End. As a side condition, it requires that no alive tuple matching ι - which is either a pid or an atom - exists and that the ghost tuples related to ι are the same as when the corresponding forward action triggered.

Rule $\overline{\textsc{ReadM}}$ is used to undo rule Registered. It requires that the map M_1 stored in the history is exactly the current map M.

$$\overline{\textsc{RegisterS}}\quad \Gamma;\langle p,\mathbf{regS}(\theta,e,S,\{\langle a,p',t,\top\rangle\}){:}h,\theta',e',S'\rangle \mid \Pi;\mathrm{M}\cup\{\langle a,p',t,\top\rangle\} \leftharpoonup \Gamma;\langle p,h,\theta,e,S\rangle \mid \Pi;\mathrm{M} \quad \text{if } \mathrm{readop}(t,\Pi)=\emptyset$$

$$\overline{\textsc{Del}}\quad \Gamma;\langle p,\mathbf{del}(\theta,e,S,\{\langle a,p',t,\top\rangle\},\mathrm{M}_1){:}h,\theta',e',S'\rangle \mid \Pi;\mathrm{M}\cup\{\langle a,p',t,\bot\rangle\} \leftharpoonup \Gamma;\langle p,h,\theta,e,S\rangle \mid \Pi;\mathrm{M}\cup\{\langle a,p',t,\top\rangle\} \quad \text{if } \mathrm{M}_a=\emptyset\wedge\mathrm{M}_{p'}=\emptyset\wedge\mathrm{readmap}(\mathrm{M}\cup\{\langle a,p',t,\bot\rangle\},\Pi)=\emptyset\wedge\mathrm{readghost}(t,\Pi)=\emptyset\wedge\mathrm{M}_1=\mathrm{M}^a\cup\mathrm{M}^{p'}$$

$$\overline{\textsc{ReadS}}\quad \Gamma;\langle p,\mathbf{readS}(\theta,e,S,\mathrm{M}_1){:}h,\theta',e',S'\rangle \mid \Pi;\mathrm{M} \leftharpoonup \Gamma;\langle p,h,\theta,e,S\rangle \mid \Pi;\mathrm{M} \quad \text{if } \mathrm{M}_1\subseteq\mathrm{M}$$

$$\overline{\textsc{SendS}}\quad \Gamma\cup\{(p,p',\{v,\lambda\})\};\langle p,\mathbf{sendS}(\theta,e,S,\{v,\lambda\},\mathrm{M}_1){:}h,\theta',e',S'\rangle \mid \Pi;\mathrm{M} \leftharpoonup \Gamma;\langle p,h,\theta,e,S\rangle \mid \Pi;\mathrm{M} \quad \text{if } \mathrm{M}_1\subseteq\mathrm{M}$$

$$\overline{\textsc{ReadF}}\quad \Gamma;\langle p,\mathbf{readF}(\theta,e,S,\iota,\mathrm{M}_1){:}h,\theta',e',S'\rangle \mid \Pi;\mathrm{M} \leftharpoonup \Gamma;\langle p,h,\theta,e,S\rangle \mid \Pi;\mathrm{M} \quad \text{if } \mathrm{M}_\iota=\emptyset\wedge\mathrm{M}_1=\mathrm{M}^\iota$$

$$\overline{\textsc{ReadM}}\quad \Gamma;\langle p,\mathbf{readM}(\theta,e,S,\mathrm{M}_1){:}h,\theta',e',S'\rangle \mid \Pi;\mathrm{M} \leftharpoonup \Gamma;\langle p,h,\theta,e,S\rangle \mid \Pi;\mathrm{M} \quad \text{if } \mathrm{M}_1=\mathrm{M}$$

Figure 3.6: Backward reversible semantics of the imperative primitives

3.4 Properties of the Reversible Semantics

Here we discuss some properties of the reversible semantics introduced in the previous section. Since most of the properties are related to causality, we need to study the concurrency model of the imperative primitives. Notably, this is not specific to reversibility and the same notation can be useful in other contexts, e.g., to find race conditions [55].

In order to present our study, we need to introduce some notation. We indicate with $\rightleftharpoons$ a forward or backward transition of the reversible semantics ($\rightleftharpoons \;=\; \rightharpoonup \cup \leftharpoonup$). We also decorate transition arrows with three pieces of information: the pid p of the process performing the action, the applied rule r, and the item k added or removed to/from the history. This notation allows us to extract relevant information from transitions. We indicate with $t : s \rightharpoonup_{p,r,k} s'$ a forward transition t from s to s', and with $\underline{t} = s' \leftharpoonup_{p,\overline{r},k} s$ the *inverse* of t. Notably, we have $\underline{\underline{t}} = t$. Two transitions are co-initial if they start from the same state, co-final if they end in the same state. We define a computation as a sequence of consecutive transitions. The empty computation is denoted by ϵ. We use d to range over computations. We define the inverse of a computation inductively as follows $\underline{d\ t} = \underline{t}\ \underline{d}$.

To define a notion of concurrency in Erlang which takes into account also the imperative primitives, we define for each history item k the set of resources (atoms and pids) read or written by the corresponding transition. The idea is that two transitions (including at least a forward one) are in conflict on the map if, according to Bernstein's conditions [56], they both access the same resource and at least one of the accesses is a write (performed by either rule RegisterS, UnregisterS, or EndUn).

Definition 17 (Resources read or written)**.** *We define functions* read(k) *and* write(k) *as follows:*

k	read(k)	write(k)
$\mathbf{regS}(\theta, e, S, \{\langle a, p, t, \top\rangle\})$	$\emptyset$	$\{a, p\}$
$\mathbf{del}(\theta, e, S, \{\langle a, p, t, \top\rangle\}, \mathrm{M})$	$\emptyset$	$\{a, p\}$
$\mathbf{readS}(\theta, e, S, \mathrm{M})$	$\{a \mid \mathrm{M}_a \neq \emptyset\} \cup \{p \mid \mathrm{M}_p \neq \emptyset\}$	$\emptyset$
$\mathbf{sendS}(\theta, e, S, \{v, \lambda\}, \{\langle a, p, t, \top\rangle\})$	$\{a, p\}$	$\emptyset$
$\mathbf{readF}(\theta, e, S, \iota, \mathrm{M})$	$\{\iota\}$	$\emptyset$
$\mathbf{readM}(\theta, e, S, \mathrm{M})$	$\{a \mid a \textit{ is an atom}\}$	$\emptyset$
any other item	$\emptyset$	$\emptyset$

A transition reads the map if read$(k) \neq \emptyset$ *and writes on it if* write$(k) \neq \emptyset$*; a transition is on the map if it either reads or writes on it.*

Functions read(k) and write(k) above given a history item k define, respectively, the set of resources read and written by the transitions producing (or consuming, in case of backward transitions) it. Intuitively, a transition writes a given atom a (resp. a given pid p) if it modifies the set of tuples containing the atom a (resp. pid p), and it reads a (resp. p) if it accesses such set of tuples without modifying it. For instance, the transitions defined using rule RegisterS (and creating item **regS**) add a tuple with an atom a and a pid p, hence they write on both resources. The reasoning is analogous for the transitions producing item **del**, which remove a tuple with the atom a and the pid p. The transitions producing item **readS** read one or two tuples, and access in read modality all the involved pids and atoms. The transitions producing item **sendS** just read the atom and pid of the accessed tuple. The transitions producing item **readF** access in read modality either an atom or a pid, as tracked in the history item. Finally, the transitions producing item **readM** exactly store the current map, and need to be in conflict with any transition writing on the map, even if it writes a tuple with an atom and a pid not previously used. Hence, we have chosen as read resources the set of all possible atoms, independently of whether they are currently used or not. We could additionally store all possible pids, but this will not impact the resulting notion of conflict, since each write access touches on at least an atom.

Definition 18 (Concurrent transitions)**.** *Two co-initial transitions,* $t_1 = (s \rightleftharpoons_{p_1, r_1, k_1} s_1)$ *and* $t_2 = (s \rightleftharpoons_{p_2, r_2, k_2} s_2)$*, are* in conflict *if one of these conditions hold:*

- *they both target the same process, i.e.,* $p_1 = p_2$*;*
- *one transition is backwards and undoes the send of some message* $\{v, \lambda\}$ *(via either rule* $\overline{\text{Send}}$ *or* $\overline{\text{SendS}}$*) and the other is forwards and receives the message* $\{v, \lambda\}$*;*
- *one is a forward transition performed by some process* p*, say* $p_1 = p$*, and the other one is a* $\overline{\text{Spawn}}$ *that undoes the creation of* p*;*
- *both the transitions are on the map, at least one is forwards, and either* read$(k_1) \cap$ write$(k_2) \neq \emptyset$*, or* read$(k_2) \cap$ write$(k_1) \neq \emptyset$*, or* write$(k_1) \cap$ write$(k_2) \neq \emptyset$*.*

Two co-initial transitions are concurrent if they are not in conflict.

Intuitively, concurrent transitions can be executed in any order (we will formalize this in Lemma 2). Note that co-initial backward transitions are never in conflict (indeed, each process has at most one backward transition enabled, determined by its most recent history item).

Example 1 (Conflicting registers). Consider a system s where two processes, say p_1 and p_2, try to register two different pids (say p' and p'', respectively) under the same atom a, and a is not already present in the map M (recall that an atom can be associated to one pid only).

$$s = \emptyset; \langle p_1, h_1, \theta_1, \mathsf{register(a, p')}, S_1\rangle \mid \langle p_2, h_2, \theta_2, \mathsf{register(a, p'')}, S_2\rangle; \emptyset$$

In this scenario the order in which the two actions are performed matters, because the first process to perform the action succeeds, while the second one is doomed to fail. Let us assume p_1 acts first.

$$\begin{aligned} s \;&\rightharpoonup\; \emptyset; \begin{array}{l}\langle p_1, \mathbf{regS}(\theta_1, \mathsf{register(a, p')}, S_1, \{\langle a, p', t, \top\rangle\}){:}h_1, \theta_1, \mathsf{true}, S_1\rangle \mid \\ \langle p_2, h_2, \theta_2, \mathsf{register(a, p'')}, S_2\rangle\end{array} ; \{\langle a, p', t, \top\rangle\} \\ &\rightharpoonup\; \emptyset; \begin{array}{l}\langle p_1, \mathbf{regS}(\theta_1, \mathsf{register(a, p')}, S_1, \{\langle a, p', t, \top\rangle\}){:}h_1, \theta_1, \mathsf{true}, S_1\rangle \mid \\ \langle p_2, \mathbf{readS}(\theta_2, \mathsf{register(a, p'')}, S_2, \{\langle a, p', t, \top\rangle\}){:}h_2, \theta_2, \epsilon, []\rangle\end{array} ; \{\langle a, p', t, \top\rangle\} = s' \end{aligned}$$

Notably, in the above computation, the register at the second step fails since we have $\mathrm{M}_a = \{\langle a, p', t, \top\rangle\} \neq \emptyset$. Dually, if we take the steps in the opposite order, we reach a system s'' of the form:

$$s'' = \quad \emptyset; \begin{array}{l}\langle p_1, \mathbf{readS}(\theta_1, \mathsf{register(a, p')}, S_1, \{\langle a, p'', t', \top\rangle\}){:}h_1, \theta_1, \epsilon, []\rangle \mid \\ \langle p_2, \mathbf{regS}(\theta_2, \mathsf{register(a, p'')}, S_2, \{\langle a, p'', t', \top\rangle\}){:}h_2, \theta_2, \mathsf{true}, S_2\rangle\end{array} ; \{\langle a, p'', t', \top\rangle\}$$

The two computations lead us to two systems: s' where p_1 has succeeded and p_2 failed (the history item created by the first transition is $k_1 = \mathbf{regS}(\theta_1, e_1, S_1, \{\langle a, p', t, \top\rangle\})$, and s'' where p_2 succeeded and p_1 failed (the history item created by the second transition is $k_2 = \mathbf{regS}(\theta_2, e_2, S_2, \{\langle a, p'', t', \top\rangle\})$). Clearly $s' \neq s''$. As we will see in Lemma 2, this is a consequence of the fact that the two transitions are in conflict, as visible since $\mathrm{write}(k_1) \cap \mathrm{write}(k_2) = \{a, p'\} \cap \{a, p''\} = \{a\} \neq \emptyset$. ◊

Example 2 (Register followed by delete).

$$s = \emptyset; \langle p_1, h_1, \theta_1, \mathsf{registered()}, S_1\rangle \mid \langle p_2, h_2, \theta_2, \mathsf{register(a, p)}, \mathsf{unregister(a)}, S_2\rangle; \emptyset$$

Consider the system s above where the process p_1 can do a registered operation, reading some map M. Another process, p_2, performs a (successful) register, obtaining as new map $\mathrm{M} \cup \{a, p, t, \top\}$, followed by a delete operation (e.g., unregister) of the same tuple. The map now becomes $\mathrm{M} \cup \{a, p, t, \bot\}$.

Let us now consider two of the possible computations. We start with p_1 executing first,

followed by p_2:

$$
\begin{array}{rcl}
s & \rightharpoonup & \emptyset;\ \langle p_1, \mathbf{readM}(\theta_1, \mathsf{registered}(), S_1, \emptyset){:}h_1, \theta_1, [], S_1\rangle \mid \\
& & \quad\ \langle p_2, h_2, \theta_2, \mathsf{register(a,\,p)}, \mathsf{unregister(a)}, S_2\rangle \ ;\emptyset \\
& \rightharpoonup & \emptyset;\ \langle p_1, \mathbf{readM}(\theta_1, \mathsf{registered}(), S_1, \emptyset){:}h_1, \theta_1, [], S_1\rangle \mid \\
& & \quad\ \langle p_2, \mathbf{regS}(\theta_2, \mathsf{register(a,\,p)}, \mathsf{unregister(a)}, S_2, \{\langle a,p,t,\top\rangle\}){:}h_2, \theta_2, \mathsf{true}, \mathsf{unregister(a)}, S_2\rangle \ ;\{\langle a,p,t,\top\rangle\} \\
& \rightharpoonup & \emptyset;\ \langle p_1, \mathbf{readM}(\theta_1, \mathsf{registered}(), S_1, \emptyset){:}h_1, \theta_1, [], S_1\rangle \mid \\
& & \quad\ \langle p_2, \begin{array}{l}\tau(\theta_2, \mathsf{true}, \mathsf{unregister(a)}, S_2){:} \\ \mathbf{regS}(\theta_2, \mathsf{register(a,\,p)}, \mathsf{unregister(a)}, S_2, \{\langle a,p,t,\top\rangle\}){:}h_2\end{array}, \theta_2, \mathsf{unregister(a)}, S_2\rangle \ ;\{\langle a,p,t,\top\rangle\} \\
& \rightharpoonup & \emptyset;\ \langle p_1, \mathbf{readM}(\theta_1, \mathsf{registered}(), S_1, \emptyset){:}h_1, \theta_1, [], S_1\rangle \mid \\
& & \quad\ \langle p_2, \begin{array}{l}\mathbf{del}(\theta_2, \mathsf{unregister(a)}, S_2, \{\langle a,p,t,\top\rangle\}, \emptyset){:}\tau(\theta_2, \mathsf{true}, \mathsf{unregister(a)}, S_2){:} \\ \mathbf{regS}(\theta_2, \mathsf{register(a,\,p)}, \mathsf{unregister(a)}, S_2, \{\langle a,p,t,\top\rangle\}){:}h_2\end{array}, \theta_2, \mathsf{true}, S_2\rangle \ ;\{\langle a,p,t,\bot\rangle\} = s'
\end{array}
$$

Let us now execute p_2 first, followed by p_1:

$$
\begin{array}{rcl}
s & \rightharpoonup & \emptyset;\ \langle p_1, h_1, \theta_1, \mathsf{registered}(), S_1\rangle \mid \\
& & \quad\ \langle p_2, \mathbf{regS}(\theta_2, \mathsf{register(a,\,p)}, \mathsf{unregister(a)}, S_2, \{\langle a,p,t,\top\rangle\}){:}h_2, \theta_2, \mathsf{true}, \mathsf{unregister(a)}, S_2\rangle \ ;\{\langle a,p,t,\top\rangle\} \\
& \rightharpoonup & \emptyset;\ \langle p_1, h_1, \theta_1, \mathsf{registered}(), S_1\rangle \mid \\
& & \quad\ \langle p_2, \begin{array}{l}\tau(\theta_2, \mathsf{true}, \mathsf{unregister(a)}, S_2){:} \\ \mathbf{regS}(\theta_2, \mathsf{register(a,\,p)}, \mathsf{unregister(a)}, S_2, \{\langle a,p,t,\top\rangle\}){:}h_2\end{array}, \theta_2, \mathsf{unregister(a)}, S_2\rangle \ ;\{\langle a,p,t,\top\rangle\} \\
& \rightharpoonup & \emptyset;\ \langle p_1, h_1, \theta_1, \mathsf{registered}(), S_1\rangle \mid \\
& & \quad\ \langle p_2, \begin{array}{l}\mathbf{del}(\theta_2, \mathsf{unregister(a)}, S_2, \{\langle a,p,t,\top\rangle\}, \emptyset){:}\tau(\theta_2, \mathsf{true}, \mathsf{unregister(a)}, S_2){:} \\ \mathbf{regS}(\theta_2, \mathsf{register(a,\,p)}, \mathsf{unregister(a)}, S_2, \{\langle a,p,t,\top\rangle\}){:}h_2\end{array}, \theta_2, \mathsf{true}, S_2\rangle \ ;\{\langle a,p,t,\bot\rangle\} \\
& \rightharpoonup & \emptyset;\ \langle p_1, \mathbf{readM}(\theta_1, \mathsf{registered}(), S_1, \{\langle a,p,t,\bot\rangle\}){:}h_1, \theta_1, [], S_1\rangle \mid \\
& & \quad\ \langle p_2, \begin{array}{l}\mathbf{del}(\theta_2, \mathsf{unregister(a)}, S_2, \{\langle a,p,t,\top\rangle\}, \emptyset){:}\tau(\theta_2, \mathsf{true}, \mathsf{unregister(a)}, S_2){:} \\ \mathbf{regS}(\theta_2, \mathsf{register(a,\,p)}, \mathsf{unregister(a)}, S_2, \{\langle a,p,t,\top\rangle\}){:}h_2\end{array}, \theta_2, \mathsf{true}, S_2\rangle \ ;\{\langle a,p,t,\bot\rangle\} = s''
\end{array}
$$

Under the standard semantics, executing first p_1 and then p_2 or vice versa would lead to the same system. Indeed, p_2 is the same in s' and s'', while the only difference in p_1 is that in the history in s'' there is a ghost tuple $\langle a,p,t,\bot\rangle$, which is not present in s'. If we were not using ghost tuples, the histories of p_1 and p_2 would be the same regardless of the order of the operations. However, we want to distinguish the two computations above, since, in particular, the delete operation can be immediately undone only in s'. Indeed, if we undo the delete operation in s'', then undoing and redoing the registered would result in reading a different map, violating the Loop Lemma (cf. Lemma 1). This behavior is ruled out in our semantics by the side condition $\mathrm{readmap}(\mathrm{M}\cup\{\langle a,p',t,\bot\rangle\},\Pi)=\emptyset$ in rule $\overline{\text{DEL}}$, requiring that no process has read the current map (including its ghost tuples). Hence the registered transition can not be concurrent to either of register or unregister, otherwise they would commute because of the Square Property (cf. Lemma 2).

We get a similar behavior also if we consider, instead of the registered operation, any other read operation involving the added tuple. ◊

We can now discuss some relevant properties of the reversible semantics. As standard (see, e.g., [31] and the notion of consistency in [30]) we consider only reachable systems, namely systems obtained from a single process with empty history (and empty Γ and M) via some computation.

Definition 19 (Reachable system)**.** *A system is* initial *if it is composed of a single process, and this process has an empty history; furthermore the global mailbox and the map are both empty. A system s is* reachable *if there exists an initial system s_0 and a derivation $s_0 \rightharpoonup^* s$ using the rules of the semantics.*

The first property, called the Loop Lemma [13, Lemma 6], states that each transition can be undone.

Lemma 1 (Loop Lemma)**.** *For every pair of reachable systems, s_1 and s_2, we have $s_1 \rightharpoonup s_2$ iff $s_2 \leftharpoondown s_1$.*

Proof. The proof that a forward transition can be undone follows by rule inspection. The other direction relies on the restriction to reachable systems: consider the process undoing the action. Since the system is reachable, restoring the history item would put us back in a state where the undone action can be performed again (if the system would not be reachable the history item would be arbitrary, hence there would not be such a guarantee), as desired. Again, this can be proved by rule inspection. We show below two sample cases, the others are similar:

Case Send: consider the system

$$s_2 = \Gamma \cup \{(p, p', \{v, \lambda\})\}; \langle p, \mathbf{send}(\theta', e', S', \{v, \lambda\}){:}h, \theta, e, S\rangle \mid \Pi; \mathrm{M}$$

and consider transition $s_2 \leftharpoondown s_1$, where

$$s_1 = \Gamma; \langle p, h, \theta', e', S'\rangle \mid \Pi; \mathrm{M}$$

Given that the system s_2 is reachable, the history item $\mathbf{send}(\theta', e', S', \{v, \lambda\})$ has been created by a forward transition derived by rule Send, hence from its hypothesis $\theta', e', S' \xrightarrow{\mathsf{send}(p', v)} \theta, e, S$ (notice that processes are sequential, hence the resulting configuration should be θ, e, S). We can thus apply rule Send to derive $s_1 \rightharpoonup s_2$ where

$$s_2 = \Gamma \cup \{(p, p', \{v, \lambda\})\}; \langle p, \mathbf{send}(\theta', e', S', \{v, \lambda\}){:}h, \theta, e, S\rangle \mid \Pi; \mathrm{M}$$

Case Register: consider the system

$$s_2 = \Gamma; \langle p, \mathbf{regS}(\theta', e', S', \{\langle a, p', t, \top\rangle\}){:}h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \cup \{\langle a, p', t, \top\rangle\}$$

and consider transition $s_2 \leftharpoondown s_1$, where

$$s_1 = \Gamma; \langle p, h, \theta', e', S'\rangle; \mathrm{M}$$

Given that the system s_2 is reachable, the history item $\mathbf{regS}(\theta', e', S', \{\langle a, p', t, \top\rangle\})$ has been derived by an application of rule RegisterS, hence from its hypothesis

$\theta', e', S' \xrightarrow{\mathsf{register}(\kappa,a,p')} \theta, e, S$. Since s_2 is reachable and has map $\mathrm{M} \cup \{\langle a, p', t, \top\rangle\}$ we know that $\mathrm{M}_a = \emptyset$ and $\mathrm{M}_{p'} = \emptyset$. Furthermore, thanks to the hypothesis of rule RegisterS, process p' was alive when the transition has been executed. It is still alive, otherwise the system would have executed either rule EndUn or End, and this is not possible. Rule EndUn could not have been executed, otherwise operation kill would have killed the tuple $\langle a, p', t, \top\rangle$, but we need it alive to apply rule $\overline{\text{RegisterS}}$. Rule End could not have been applied since it requires that the process p' is not registered. We can thus apply rule RegisterS to derive $s_1 \rightharpoonup s_2$ where

$$s_2 = \Gamma; \langle p, \mathbf{regS}(\theta', e', S', \{\langle a, p', t, \top\rangle\}){:}h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \cup \{\langle a, p', t, \top\rangle\}$$

□

Thanks to the Loop Lemma, $\underline{t}$ exists iff t exists.

We show now that the restriction to reachable systems is needed to prove the Loop Lemma, in particular to show that a backward transition can always be undone by a forward one.

Example 3 (Loop Lemma, counterexample for unreachable systems). Consider the system:

$$s_1 = \emptyset; \langle p, \mathbf{regS}(\theta', 1, S', \{\langle a, p', t, \top\rangle\}){:}h, \theta, e, S\rangle; \{\langle a, p', t, \top\rangle\}$$

This system is unreachable because the expression to restore (second element of the item **regS**) is 1 and not the predecessor of e (a register expression). Indeed, consider transition $s_1 \leftharpoondown s_2$, obtained by rule $\overline{\text{RegisterS}}$, where:

$$s_2 = \emptyset; \langle p, h, \theta', 1, S'\rangle; \emptyset$$

Now it is not possible to apply rule RegisterS because the expression to evaluate is not a register. Then the transition $s_2 \rightharpoonup s_1$ is not derivable. Hence, reachability of systems is needed to prove the Loop Lemma.

◊

Example 4 (Loop Lemma, further counterexample for unreachable systems). Consider the system:

$$s_1 = \emptyset; \langle p, \mathbf{readS}(\theta', \mathsf{whereis(a)}, S', \{\langle a, p', t, \top\rangle\}){:}h, \theta, e, S\rangle; \{\langle a, p', t, \top\rangle, \langle a, p'', t', \top\rangle\}$$

This system is unreachable because the map $(\{\langle a, p', t, \top\rangle, \langle a, p'', t', \top\rangle\})$ contains two alive tuples with the same atom, but this is not possible in Erlang. Indeed, consider transition $s_1 \leftharpoondown s_2$, obtained by rule $\overline{\text{Whereis1}}$, where:

$$s_2 = \emptyset; \langle p, h, \theta', \mathsf{whereis(a)}, \mathsf{S}'\rangle; \{\langle a, p', t, \top\rangle, \langle a, p'', t', \top\rangle\}$$

Transition $s_1 \leftharpoondown s_2$ undid the read of the tuple $\langle a, p', t, \top\rangle$; but now it is not possible to apply rule Whereis1 because the side condition $\mathsf{M}_a = \{\langle a, p', t, \top\rangle\}$ is not satisfied. Indeed, the map contains two alive tuples with the same atom a. This is possible only since s_1 is not reachable,

since in reachable systems there is at most one alive tuple per atom. This example shows that reachability is not only relevant to ensure that the correct expression is restored, but also to ensure a consistent structure of the system. ◊

The next lemma shows that concurrent transitions can be executed in any order. It can be seen as a safety check on the notion of concurrency.

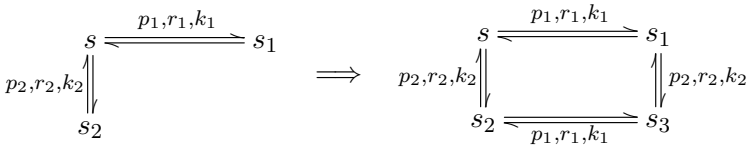
Lemma 2 (Square Property)**.** *Given two co-initial concurrent transitions*

$$t_1 = (s \rightleftharpoons_{p_1,r_1,k_1} s_1) \qquad \textit{and} \qquad t_2 = (s \rightleftharpoons_{p_2,r_2,k_2} s_2)$$

there exist two transitions

$$t_2/t_1 = (s_1 \rightleftharpoons_{p_2,r_2,k_2} s_3) \qquad \textit{and} \qquad t_1/t_2 = (s_2 \rightleftharpoons_{p_1,r_1,k_1} s_3)$$

Graphically:



Proof. We distinguish the following cases depending on the applied rules:

1. Two forward transitions. We have the following subcases:

   - the two transitions are not on the map: we can prove that by applying rule r_2 to p_1 in s_1 and rule r_1 to p_2 in s_2 we have two transitions t_1/t_2 and t_2/t_1 which are co-final, as desired;
   - one of the two transitions is on the map and the other one is not: we can apply the same reasoning as above;
   - both transitions are on the map: we have a case analysis on the applied rules. We show a few examples, the others are similar.
     - if $r_1 \in \{\textsc{UnregisterS}, \textsc{EndUn}\}$, $k_1 = \mathbf{del}(\theta_1, e_1, S_1, \mathrm{M}_1, \mathrm{M}')$ and $r_2 \in \{\textsc{UnregisterS}, \textsc{EndUn}\}$, $k_2 = \mathbf{del}(\theta_2, e_2, S_2, \mathrm{M}_2, \mathrm{M}'')$ we have that both $\mathrm{M}_1 = \{\langle a_1, p_1, t_1, \top\rangle\}$ and $\mathrm{M}_2 = \{\langle a_2, p_2, t_2, \top\rangle\}$ are in the map M of s and by definition of concurrent transitions (Definition 18) we also have that $a_1 \neq a_2$ and $p_1 \neq p_2$. Then applying k_1 followed by k_2 or vice-versa leads us to the same system because

       $$(\mathrm{M} \setminus \mathrm{M}_1 \cup \mathrm{kill}(\mathrm{M}_1)) \setminus \mathrm{M}_2 \cup \mathrm{kill}(\mathrm{M}_2) = (\mathrm{M} \setminus \mathrm{M}_2 \cup \mathrm{kill}(\mathrm{M}_2)) \setminus \mathrm{M}_1 \cup \mathrm{kill}(\mathrm{M}_1)$$

       The only side effect of each rule is to kill a tuple from the system's map.
     - in the other cases we can apply a similar reasoning, more cases are shown in Appendix A.

2. One forward transition and one backward transition. We have the following subcases:

   - the two transitions are both on the map, then we have the following subcases:

  - r_1 = RegisterS, where $k_1 = \mathbf{regS}(\_, \_, \_, \mathrm{M}_1)$ with $\mathrm{M}_1 = \{\langle a_1, p_1, t_1, \top\rangle\}$, and $r_2 = \overline{\mathrm{Del}}$, where $k_2 = \mathbf{del}(\_, \_, \_, \mathrm{M}_2, \_)$ with $\mathrm{M}_2 = \{\langle a_2, p_2, t_2, \top\rangle\}$. By Definition 18, we know that $a_1 \neq a_2$ and $p_1 \neq p_2$ (because otherwise they would be in conflict), then the map M of s contains the tuple $\langle a_2, p_2, t_2, \bot\rangle$ (otherwise it would not be possible to apply rule r_2) and we can see that the applications of r_1 and r_2 commute because

$$(\mathrm{M} \cup \mathrm{M}_1) \setminus \mathrm{kill}(\mathrm{M}_2) \cup \mathrm{M}_2 = (\mathrm{M} \setminus \mathrm{kill}(\mathrm{M}_2) \cup \mathrm{M}_2) \cup \mathrm{M}_1$$

  - in the other cases we can apply a similar reasoning, more cases are shown in Appendix A.

- the two transitions are not both on the map: the claim follows easily (see [39, Lemma 13] and [41, Lemma 3.1]).

3. Two backward transitions. We have a case analysis on the two rules. We show below a few examples related to the map, we refer to [39, Lemma 13] and [41, Lemma 3.1] for additional examples related to the other constructs:

   - $r_1 = \overline{\mathrm{RegisterS}}$, where $k_1 = \mathbf{regS}(\_, \_, \_, \mathrm{M}_1)$ with $\mathrm{M}_1 = \{\langle a_1, p_1, t_1, \top\rangle\}$, and $r_2 = \overline{\mathrm{Del}}$, where $k_2 = \mathbf{del}(\_, \_, \_, \mathrm{M}_2, \_)$ with $\mathrm{M}_2 = \{\langle a_2, p_2, t_2, \top\rangle\}$. The map M of s contains the tuple $\langle a_2, p_2, t_2, \bot\rangle$ (otherwise it would not possible to apply rule r_2) and the tuple $\langle a_1, p_1, t_1, \top\rangle$ (otherwise it would not possible to apply rule r_1). Also, we know that $t_1 \neq t_2$ since identifiers are unique. Now we can see that the applications of rules r_1 and r_2 commute because

$$(\mathrm{M} \setminus \mathrm{M}_1) \setminus \mathrm{kill}(\mathrm{M}_2) \cup \mathrm{M}_2 = (\mathrm{M} \setminus \mathrm{kill}(\mathrm{M}_2) \cup \mathrm{M}_2) \setminus \mathrm{M}_1$$

   - in the other cases we can apply a similar reasoning, more cases are shown in Appendix A. □

The next example shows that in order to ensure that the Square Property holds the semantics needs to be carefully crafted, in particular one should avoid to store information allowing to distinguish the order of execution of concurrent transitions.

Example 5 (Information carried by the register history item). If the history item of the register would contain the whole map, it would be impossible to swap the register action with an unregister action even if on a tuple with different pid and atom, because of the Square Property (Lemma 2). Indeed, the Square Property requires to reach the same system after two concurrent transitions are executed, regardless of their order. If we save the whole map in the history item of the register, we would reach two different systems:

- if we execute the register operation first, the saved map would include the tuple that the unregister operation will delete;

- if we execute the unregister operation first, the map saved by the register would not contain the deleted tuple, but only its ghost. ◊

We now want to prove that our semantics enjoys causal-consistency [13, 35], which essentially states that we store the correct amount of causal and history information.

Definition 20 (Causal equivalence)**.** *Let $\asymp$ be the smallest equivalence on computations closed under composition and satisfying:*

1. *if $t_1 = (s \rightleftharpoons_{p_1,r_1,k_1} s_1)$ and $t_2 = (s \rightleftharpoons_{p_2,r_2,k_2} s_2)$ are concurrent and $t_3 = (s_1 \rightleftharpoons_{p_2,r_2,k_2} s_3)$, $t_4 = (s_2 \rightleftharpoons_{p_1,r_1,k_1} s_3)$ then $t_1 t_3 \asymp t_2 t_4$;*

2. *$t\underline{t} \asymp \epsilon$ and $\underline{t}t \asymp \epsilon$*

Intuitively, computations are causal equivalent if they only differ by swapping concurrent transitions and by adding do-undo or undo-redo pairs of transitions.

Definition 21 (Causal Consistency)**.** *Two co-initial computations are co-final iff they are causal equivalent.*

Intuitively, if co-initial computations are co-final then they have the same causal information and can reverse in the same ways: we want computations to reverse in the same ways iff they are causal equivalent.

To prove that our semantics enjoys Causal Consistency, we rely on the theory developed in [35]. It considers a transition system with forward and backward transitions which satisfies the Loop Leamma and has a notion of independence. The latter is concurrency in our case. The theory allows one to reduce the proof that our semantics enjoys Causal Consistency and other relevant properties to the validity of five axioms: Square Property (SP), Backward Transitions are Independent (BTI), Well-Foundedness (WF), Co-initial Propagation of Independence (CPI) and Co-initial Independence Respects Event (CIRE). We proved SP in Lemma 2, BTI corresponds to the observation that two backward transitions are always concurrent (see Definition 18), and WF requires backward computations to be finite. WF holds since each backward transition consumes a history item, of which there are a finite number. CPI and CIRE hold thanks to [35, Prop. 5.4] because the notion of concurrency is defined in terms of transition labels only. Hence, Causal Consistency follows from [35, Prop. 3.6] thanks to axioms BTI, SP and WF. We obtain as well a number of other properties (a list can be found in [35, Table 1]), including various forms of causal safety and causal liveness (thanks to axioms CPI and CIRE), that intuitively say that a transition can be undone iff its consequences have been undone. One of these properties, particularly relevant for us since we will exploit it in the next sections, is the Parabolic Lemma, stated below (proved thanks to axioms BTI and SP). It essentially states that any computation can be decomposed into a backward one followed by a forward one.

Lemma 3 (Parabolic Lemma [35, Prop. 3.4])**.** *For any computation d there are forward only computations d', d'' such that $d \approx \underline{d'}d''$ and $\mid d' \mid + \mid d'' \mid \leq \mid d \mid$, where $|s|$ is the number of transitions in the computation s.*

We refer to [35] for precise definitions and further discussion on the other properties.

3.5 Rollback Semantics

We extend the rollback semantics (Section 2.1.4), enabling a debugger to undo a past action and its consequences automatically. This feature is useful for debugging, allowing users to trace causal links backwards from a misbehavior to its causes. We mostly follow the approach in [41, 40]. We however extract information for rollback from history items (cf. Definition 22) instead of writing it explicitly in the semantics as in previous approaches, better highlighting the relation between the rollback and the backward semantics. More importantly, we extend rollback with support for imperative primitives.

A rollback is formalized as a sequence of steps derived using the backward semantics, driven by the target action we want to undo. We denote a system in the rollback mode with $[\![\mathcal{S}]\!]_{\Psi}$, where Ψ is the sequence of undo requests that need to be satisfied; a request have the form $\{p, \psi\}$. We consider the same requests described in Section 2.1.4, and we add also the requests considering the imperative primitives:

- $\{p, \mathsf{regS}(t)\}$: the register of the tuple with the unique identifier t;
- $\{p, \mathsf{del}(t)\}$: the deletion (due to either unregister or the termination of the process execution) of the tuple with the unique identifier t.

For example, a request $\{p, \mathsf{regS}(t)\}$ asks to undo the register executed by the process with the pid p that inserted the tuple $\langle \_, \_, t, \top \rangle$ into the map.

We can compute the requests satisfied by a transition by abstracting the information contained in the corresponding history item. Remember that in a transition $s \rightleftharpoons_{p,r,k} s_1$ we have that k is the history item created/removed by the transition.

Definition 22 (Satisfied requests)**.** *Given a transition* $t = (s \rightleftharpoons_{p,r,k} s_1)$ *the set* $\mathrm{sat}(k)$ *of requests satisfied by* t *is defined as* $\{s\} \cup \mathrm{abst}(k)$ *where* $\mathrm{abst}(k)$ *is defined as follows:*

$$
\begin{aligned}
\mathrm{abst}(\tau(\_,\_,\_)) &= \emptyset \\
\mathrm{abst}(\mathbf{send}(\_,\_,\_,\_,\{\_,\lambda\})) &= \{\mathsf{send}(\lambda)\} \\
\mathrm{abst}(\mathbf{rec}(\_,\_,\_,\_,\{\_,\lambda\})) &= \{\mathsf{rec}(\lambda)\} \\
\mathrm{abst}(\mathbf{spawn}(\_,\_,\_,p')) &= \{\mathsf{spawn}(p')\} \\
\mathrm{abst}(\mathbf{self}(\_,\_,\_)) &= \emptyset \\
\mathrm{abst}(\mathbf{regS}(\_,\_,\_,\{\langle \_,\_,t,\_\rangle\})) &= \{\mathsf{regS}(t)\} \\
\mathrm{abst}(\mathbf{del}(\_,\_,\_,\{\langle \_,\_,t,\_\rangle\},\_)) &= \{\mathsf{del}(t)\} \\
\mathrm{abst}(\mathbf{readS}(\_,\_,\_,\_)) &= \emptyset \\
\mathrm{abst}(\mathbf{sendS}(\_,\_,\_,\{\_,\lambda\},\langle \_,\_,t,\_\rangle)) &= \{\mathsf{send}(\lambda)\} \\
\mathrm{abst}(\mathbf{readF}(\_,\_,\_,\_,\_)) &= \emptyset \\
\mathrm{abst}(\mathbf{readM}(\_,\_,\_,\_)) &= \emptyset
\end{aligned}
$$

Fig. 3.7 depicts the rollback semantics, whose steps execute selected transitions of the underlying backward semantics (the rules are described in Section 2.1.4 and they are shown

$$\textsc{U-Act}\ \frac{\mathcal{S} \leftharpoondown_{p,r,k} \mathcal{S}' \ \wedge\ \psi \notin \mathrm{sat}(k)}{[\![\mathcal{S}]\!]_{\{p,\psi\}:\Psi} \rightsquigarrow [\![\mathcal{S}']\!]_{\{p,\psi\}:\Psi}} \qquad \textsc{U-Satisfy}\ \frac{\mathcal{S} \leftharpoondown_{p,r,k} \mathcal{S}' \ \wedge\ \psi \in \mathrm{sat}(k)}{[\![\mathcal{S}]\!]_{\{p,\psi\}:\Psi} \rightsquigarrow [\![\mathcal{S}']\!]_{\Psi}}$$

$$\textsc{Request}\ \frac{\mathcal{S} = \Gamma; \langle p, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \ \wedge\ \mathcal{S} \not\leftharpoondown_{p,r,k} \ \wedge\ \mathrm{bkDep}(h, \mathcal{S}) = \{p', \psi'\}}{[\![\mathcal{S}]\!]_{\{p,\psi\}:\Psi} \rightsquigarrow [\![\mathcal{S}']\!]_{\{p',\psi'\}:\{p,\psi\}:\Psi}}$$

$$\textsc{Satisfy-Spawn}\ \frac{}{[\![\Gamma; \langle p, [\,], \theta, e, S\rangle \mid \Pi]\!]_{\{p',\mathsf{sp}\}:\Psi} \rightsquigarrow [\![\Gamma; \langle p, [\,], \theta, e, S\rangle \mid \Pi]\!]_{\Psi}}$$

Figure 3.7: Rollback semantics

$\mathrm{bkDep}(\mathbf{send}(\_,\_,\_,p',\{v,\lambda\}){:}h,\_;\_;\_)$	$= \{p', \mathsf{rec}(\lambda)\}$	
$\mathrm{bkDep}(\mathbf{sendS}(\_,\_,\_,\{v,\lambda\},\_){:}h,\_;\langle p',h',\_,\_,\_\rangle \mid \Pi;\_)$	$= \{p', \mathsf{rec}(\lambda)\}$	if $\mathbf{rec}(\_,\_,\_,\_,\{v,\lambda\}) \in h'$
$\mathrm{bkDep}(\mathbf{sendS}(\_,\_,\_,\_,\{\langle\_,\_,t,\top\rangle\}){:}h,\_;\Pi;\mathrm{M}')$	$= \{p', \mathsf{del}(t)\}$	if $\langle\_,\_,t,\top\rangle \notin \mathrm{M}' \wedge p' = \mathrm{getHP}(\mathsf{del}, t, \Pi)$
$\mathrm{bkDep}(\mathbf{spawn}(\_,\_,\_,p'){:}h,\_;\Pi;\_;\_)$	$= \{p', \mathsf{sp}\}$	if $p' \in \Pi$
$\mathrm{bkDep}(\mathbf{readS}(\_,\_,\_,\mathrm{M} \cup \{\langle\_,\_,t,\top\rangle\}){:}h,\_;\Pi;\mathrm{M}')$	$= \{p', \mathsf{del}(t)\}$	if $\langle\_,\_,t,\top\rangle \notin \mathrm{M}' \wedge p' = \mathrm{getHP}(\mathsf{del}, t, \Pi)$
$\mathrm{bkDep}(\mathbf{readF}(\_,\_,\_,a,\mathrm{M}){:}h,\_;\Pi;\mathrm{M}' \cup \{\langle a,\_,t,\_\rangle\})$	$= \{p', \mathsf{regS}(t)\}$	if $\langle a,\_,t,\_\rangle \notin \mathrm{M} \wedge p' = \mathrm{getHP}(\mathsf{regS}, t, \Pi)$
$\mathrm{bkDep}(\mathbf{readF}(\_,\_,\_,p,\mathrm{M}){:}h,\_;\Pi;\mathrm{M}' \cup \{\langle\_,p,t,\_\rangle\})$	$= \{p', \mathsf{regS}(t)\}$	if $\langle\_,p,t,\_\rangle \notin \mathrm{M} \wedge p' = \mathrm{getHP}(\mathsf{regS}, t, \Pi)$
$\mathrm{bkDep}(\mathbf{readM}(\_,\_,\_,\mathrm{M}){:}h,\_;\Pi;\mathrm{M}' \cup \{\langle\_,\_,t,\top\rangle\})$	$= \{p', \mathsf{regS}(t)\}$	if $\langle\_,\_,t,\top\rangle \notin \mathrm{M} \wedge p' = \mathrm{getHP}(\mathsf{regS}, t, \Pi)$
$\mathrm{bkDep}(\mathbf{readM}(\_,\_,\_,\mathrm{M}){:}h,\_;\Pi;\mathrm{M}' \cup \{\langle\_,\_,t,\bot\rangle\})$	$= \{p', \mathsf{del}(t)\}$	if $\langle\_,\_,t,\bot\rangle \notin \mathrm{M} \wedge p' = \mathrm{getHP}(\mathsf{del}, t, \Pi)$
$\mathrm{bkDep}(\mathbf{regS}(\_,\_,\_,\{\langle\_,\_,t,\top\rangle\}){:}h,\_;\Pi;\mathrm{M}')$	$= \{p', \mathsf{del}(t)\}$	if $\langle\_,\_,t,\top\rangle \notin \mathrm{M}' \wedge p' = \mathrm{getHP}(\mathsf{del}, t, \Pi)$
$\mathrm{bkDep}(\mathbf{regS}(\_,\_,\_,\{\langle\_,\_,t,\top\rangle\}){:}h,\_;\Pi;\_)$	$= \{p', s\}$	if $p' = \mathrm{getHP}(\mathsf{read}, t, \Pi)$
$\mathrm{bkDep}(\mathbf{del}(\_,\_,\_,\{\langle a,\_,t,\top\rangle\},\mathrm{M}){:}h,\_;\Pi;\mathrm{M}' \cup \{\langle a,\_,t_a,\_\rangle\})$	$= \{p', \mathsf{regS}(t_a)\}$	if $\langle a,\_,t_a,\_\rangle \notin \mathrm{M} \wedge p' = \mathrm{getHP}(\mathsf{regS}, t_a, \Pi)$
$\mathrm{bkDep}(\mathbf{del}(\_,\_,\_,\{\langle\_,p,t,\top\rangle\},\mathrm{M}){:}h,\_;\Pi;\mathrm{M}' \cup \{\langle\_,p,t_p,\_\rangle\})$	$= \{p', \mathsf{regS}(t_p)\}$	if $\langle\_,p,t_p,\_\rangle \notin \mathrm{M} \wedge p' = \mathrm{getHP}(\mathsf{regS}, t_p, \Pi)$
$\mathrm{bkDep}(\mathbf{del}(\_,\_,\_,\{\langle\_,\_,t,\top\rangle\},\_){:}h,\_;\Pi;\_)$	$= \{p', s\}$	if $p' = \mathrm{getHP}(\mathsf{rghost}, t, \Pi)$

Figure 3.8: Backward dependencies operator

Fig. 2.8). It is possible to see that previously there was a rule for each request (Request-Send and Request-Spawn). Now in Fig. 3.7 we consider only one rule Request for satisfy each request and it is fired when the process p targeted by the topmost request $\{p, \psi\}$ has no backward transition enabled (indicated by $\not\leftharpoondown_{p,r,k}$). This means that the process p is blocked because of some dependency on an action executed by another process. The operator bkDep (defined in Fig. 3.8) computes a new rollback request $\{p', \psi'\}$, aimed at solving the dependency. The request $\{p', \psi'\}$ is then pushed on Ψ. Hence, the dependency will be solved before coming back to the request $\{p, \psi\}$.

If there are multiple dependencies to solve, we add them one by one. Examples are given below, when describing in detail the bkDep operator. Adding all the dependencies at once would be more complex, since by resolving one dependency, we could also resolve some deeper ones; in this way, we would need an additional check to avoid starting a computation to solve a dependency that no longer exists.

Adding dependencies one by one avoids the problem, so the bkDep operator (defined in Fig. 3.8) non-deterministically selects one of them. The order in which dependencies are resolved is not relevant. Let us discuss the cases of the definition:

send: a send cannot be undone since the message sent has already been received by another process p', so a request has to be made to p' to undo the receipt of the (unique) message identified by λ. That cover the rule Request-Send.

sendS: this case is similar to the one above, but it considers messages sent using rule SendS.

$$
\begin{array}{lll}
\mathrm{getHP}(\mathsf{del}, t, \langle p, h, \_, \_, \_\rangle \mid \Pi) & = p & \text{if } \mathbf{del}(\_, \_, \_, \{\_, \_, t, \_\}, \_) \in h \\
\mathrm{getHP}(\mathsf{regS}, t, \langle p, h, \_, \_, \_\rangle \mid \Pi) & = p & \text{if } \mathbf{regS}(\_, \_, \_, \{\_, \_, t, \_\}) \in h \\
\mathrm{getHP}(\mathsf{read}, t, \langle p, h, \_, \_, \_\rangle \mid \Pi) & = p & \text{if } \begin{cases} \mathbf{readS}(\_, \_, \_, \mathrm{M} \cup \{\langle \_, \_, t, \top\rangle\}) \in h \\ \vee\ \mathbf{sendS}(\_, \_, \_, \_, \{\langle \_, \_, t, \top\rangle\}) \in h \\ \vee\ \mathbf{readM}(\_, \_, \_, \mathrm{M}' \cup \{\langle \_, \_, t, \top\rangle\}) \in h \end{cases} \\
\mathrm{getHP}(\mathsf{rghost}, t, \langle p, h, \_, \_, \_\rangle \mid \Pi) & = p & \text{if } \begin{cases} \mathbf{readF}(\_, \_, \_, \_, \mathrm{M} \cup \{\langle \_, \_, t, \bot\rangle\}) \in h \\ \vee\ \mathbf{readM}(\_, \_, \_, \mathrm{M}' \cup \{\langle \_, \_, t, \bot\rangle\}) \in h \end{cases}
\end{array}
$$

Figure 3.9: Get pid operator

This is a case where the bkDep operator is nondeterministic. In fact, a dependency is possible both on the received message as for rule SEND, as specified by the first clause, as well as on the atom a, as specified by the second clause. Indeed, if the tuple is no longer alive in M we need to ask the process p' that killed it to undo the delete. This is obtained by the $\{p', \mathsf{del}(t)\}$ request, where p' is the pid of the process that deleted the tuple with the unique identifier t. The pid p' is computed using the auxiliary operator getHP, defined in Fig. 3.9. Operator getHP takes a descriptor of a memory item, a unique identifier t and a pool of processes, and finds the pid of a process whose history contains an operation on the tuple with the identifier t compatible with the descriptor. Notice that, while for the sendS request the pid that fits is unique, in general many pids may fit the descriptor (e.g., many processes may have read the same tuple), hence the operator non deterministically selects one.

spawn: before undoing the spawn of a process p', it is necessary to undo all the operations executed by p', hence bkDep returns a request of a step back of p'. If p' has executed more than one step, the same case of bkDep will be triggered again by a new application of rule REQUEST. This avoids introducing an auxiliary dependency target. That cover the rule REQUEST-SPAWN.

readS: in this case it is necessary that the tuple t read is still in the map. If it is not then it means that it has been deleted by some successive operation, hence we need to undo the deletion of t. As before, the pid p' of the process that executed the delete of t is retrieved by means of the auxiliary operator getHP.

readF: for readF we have two cases, depending on whether the failed read concerned an atom a or a pid p. In both the cases we have to undo all the registers on the same atom a or pid p which are not in the map carried by the history item. This last check is needed since there may be ghosts on the same atom or pid which preceded the readF, and should not be undone. Ghosts of tuples created after the readF need to be removed instead. The operator getHP returns the pid of the process that executed the register primitive.

readM: in this case we need to restore exactly the map M read by the registered primitive. In the system's map we may have in addition both further alive tuples, and in this case we need to undo their creation via a register operation, and further dead tuples, and in this case we need to undo their deletion (indeed the corresponding alive tuple may be in M).

regS: in the case of the register of a tuple t it is necessary to undo all the operations on t, namely the deletion of t (if it is no longer alive in the system's map) as well as the read operations on t. In this last case we use as request just to undo a step. This is enough since if the dependency is not solved then the same condition will trigger again. As usual, the pid p' of the target process is obtained using the operator getHP.

del: in the case of the deletion of a tuple t we may have as dependencies: (i) a tuple on the same atom or pid, whose register needs to be undone before undoing the deletion, unless it is a pre-existing ghost (which is in M); (ii) a read operation on the ghost of t (undone using requests for undoing single steps till needed).

We now present an example to clarify how rollback requests are managed.

Example 6. We consider the system in rollback mode $[\![\mathcal{S}]\!]_{\{p_1,\mathsf{regS}(t)\}}$, where:

$$\mathcal{S} = \{(p_2,p,\{v,\lambda\})\}; \begin{array}{l} \mid \langle p_1, \mathbf{regS}(\theta_1, \mathsf{register(a,p)}, S_1, \{\langle a,p,t,\top\rangle\}){:}h_1, \theta_1', e_1', S_1'\rangle \\ \mid \langle p_2, \mathbf{sendS}(\theta_2, a \mathbin{!} v, S_2, \{v,\lambda\}, \{\langle a,p,t,\top\rangle\}){:}h_2, \theta_2', e_2', S_2'\rangle \\ \mid \langle p_3, \mathbf{del}(\theta_3, \mathsf{unregister(a)}, S_3, \{\langle a,p,t,\top\rangle\}, \emptyset){:}h_3, \theta_3', e_3', S_3'\rangle \end{array} ; \{\{a,p,t,\bot\}\}$$

and the rollback request $\{p_1, \mathsf{regS}(t)\}$ asks to undo the register executed by process p_1 on the tuple with identifier t.

We cannot directly answer the request, since rule $\overline{\text{RegisterS}}$ is not applicable to system $\mathcal{S}$ ($\mathcal{S} \not\rightarrow_{p_1,r,k}$). Indeed, two conditions fail: first, the map does not contain an alive tuple with the identifier t (the corresponding tuple is dead), second, the side condition $\text{readop}(t,\Pi) = \emptyset$ fails since the process p_2 has read the tuple with identifier t. As a result, we can only apply rule Request, which adds the request $\{p_3, \mathsf{del}(t)\}$ (computed using the case analysis in Fig. 3.8, where the pid p_3 of the target process is computed using the operator getHP in Fig. 3.9). Now, the system becomes $[\![\mathcal{S}]\!]_{\{p_3,\mathsf{del}(t)\}:\{p_1,\mathsf{regS}(t)\}}$, where it is possible to apply rule U-Satisfy satisfying the request $\{p_3, \mathsf{del}(t)\}$. As a result, we can take the step:

$$\mathcal{S} = \{(p_2,p,\{v,\lambda\})\}; \begin{array}{l} \mid \langle p_1, \mathbf{regS}(\theta_1, \mathsf{register(a,p)}, S_1, \{\langle a,p,t,\top\rangle\}){:}h_1, \theta_1', e_1', S_1'\rangle \\ \mid \langle p_2, \mathbf{sendS}(\theta_2, a \mathbin{!} v, S_2, \{v,\lambda\}, \{\langle a,p,t,\top\rangle\}){:}h_2, \theta_2', e_2', S_2'\rangle \\ \mid \langle p_3, \mathbf{del}(\theta_3, \mathsf{unregister(a)}, S_3, \{\langle a,p,t,\top\rangle\}, \emptyset){:}h_3, \theta_3', e_3', S_3'\rangle \end{array} ; \{\{a,p,t,\bot\}\}$$

$$\rightharpoonup \{(p_2,p,\{v,\lambda\})\}; \begin{array}{l} \mid \langle p_1, \mathbf{regS}(\theta_1, \mathsf{register(a,p)}, S_1, \{\langle a,p,t,\top\rangle\}){:}h_1, \theta_1', e_1', S_1'\rangle \\ \mid \langle p_2, \mathbf{sendS}(\theta_2, a \mathbin{!} v, S_2, \{v,\lambda\}, \{\langle a,p,t,\top\rangle\}){:}h_2, \theta_2', e_2', S_2'\rangle \\ \mid \langle p_3, h_3, \theta_3, \mathsf{unregister(a)}, S_3\rangle \end{array} ; \{\{a,p,t,\top\}\} = \mathcal{S}'$$

and we get $[\![\mathcal{S}']\!]_{\{p_1,\mathsf{regS}(t)\}}$.

As before, we can only apply rule Request, since rule $\overline{\text{RegisterS}}$ is not applicable to system $\mathcal{S}'$. As a result, we add the request $\{p_2, s\}$ to undo the read of the tuple with the identifier t (computed again using the case analysis in Fig. 3.14 using the operator getHP in Fig. 3.15).

Now the system is $[\![\mathcal{S}']\!]_{\{p_2,s\}:\{p_1,\mathsf{regS}(t)\}}$ and it is possible to apply rule U-Satisfy to execute

the step:

$$\mathcal{S}' = \{(p_2,p,\{v,\lambda\})\};\ \begin{array}{l}|\ \langle p_1, \mathbf{regS}(\theta_1,\mathsf{register(a,p)},S_1,\{\langle a,p,t,\top\rangle\}){:}h_1,\theta_1',e_1',S_1'\rangle\\ |\ \langle p_2, \mathbf{sendS}(\theta_2,a\ !\ v,S_2,\{v,\lambda\},\{\langle a,p,t,\top\rangle\}){:}h_2,\theta_2',e_2',S_2'\rangle\\ |\ \langle p_3,h_3,\theta_3,\mathsf{unregister(a)},S_3\rangle\end{array};\{\{a,p,t,\top\}\}$$

$$\leftharpoondown\quad \emptyset;\ \begin{array}{l}|\ \langle p_1, \mathbf{regS}(\theta_1,\mathsf{register(a,p)},S_1,\{\langle a,p,t,\top\rangle\}){:}h_1,\theta_1',e_1',S_1'\rangle\\ |\ \langle p_2,h_2,\theta_2,a\ !\ v,S_2\rangle\\ |\ \langle p_3,h_3,\theta_3,\mathsf{unregister(a)},S_3\rangle\end{array};\{\{a,p,t,\top\}\} = \mathcal{S}''$$

The resulting system is $[\![\mathcal{S}'']\!]_{\{p_2,\mathsf{regS}(t)\}}$, where it is possible to apply rule U-SATISFY satisfying the last request, leading to $\mathcal{S}'''$:

$$\mathcal{S}'' = \emptyset;\ \begin{array}{l}|\ \langle p_1, \mathbf{regS}(\theta_1,\mathsf{register(a,p)},S_1,\{\langle a,p,t,\top\rangle\}){:}h_1,\theta_1',e_1',S_1'\rangle\\ |\ \langle p_2,h_2,\theta_2,a\ !\ v,S_2\rangle\\ |\ \langle p_3,h_3,\theta_3,\mathsf{unregister(a)},S_3\rangle\end{array};\{\{a,p,t,\top\}\}$$

$$\leftharpoondown\quad \emptyset;\ \begin{array}{l}|\ \langle p_1,h_1,\theta_1,\mathsf{register(a,p)},S_1\rangle\\ |\ \langle p_2,h_2,\theta_2,a\ !\ v,S_2\rangle\\ |\ \langle p_3,h_3,\theta_3,\mathsf{unregister(a)},S_3\rangle\end{array};\emptyset = \mathcal{S}'''$$

Since there are no further pending requests, the rollback is completed. ◊

Now let us focus on the properties of the rollback semantics and of the relation $\leftsquigarrow$, which can be seen as a *controlled* version of the uncontrolled reversible semantics ($\leftharpoondown$). Indeed, an execution of the rollback operator corresponds to a backward derivation.

Theorem 1 (Soundness of rollback)**.** *If* $[\![\mathcal{S}]\!]_\Psi \leftsquigarrow^* [\![\mathcal{S}']\!]_{\Psi'}$ *then* $\mathcal{S} \leftharpoondown^* \mathcal{S}'$.

Proof. The rollback semantics is either executing backward steps using the backward semantics or executing administrative steps (i.e., pushing new requests on top of Ψ via rule REQUEST), which do not alter the underlying system. □

Our controlled semantics is not only causal-consistent (since it is based on the uncontrolled one) but also *minimal*. This means that in order to satisfy the oldest request on Ψ, and consequently every other request in Ψ, we undo the least number of steps which allow us to satisfy such requests. Notice that minimality is a relevant property for a reversible debugger, since executions can be very long and minimality allows the user, in quest for a bug, to focus only on the subsystem(s) that may contain the bug, disregarding the other subsystems. This simplifies the debugging activity.

In order to prove minimality we follow the approach in [38], extending it so to cope with the imperative primitives. First, we need to restrict our attention to those rollback requests which require us to rollback transitions that are in the past of the selected process. The existence of a derivation satisfying the request can be easily checked in finite time, where the time depends on the length of the process' history.

We now re-call some auxiliary notation. First, we provide a notion of projection from controlled systems to uncontrolled ones:

$$\text{uctrl}(\llbracket \mathcal{S} \rrbracket_{\Psi}) = \mathcal{S}$$

The notion of projection trivially extends to derivations. Given a derivation $d = s_1 \hookrightarrow^* s_n$ we define init(d) as s_1 and final(d) as s_n. We proceed now to prove that controlled derivations are finite.

Lemma 4. *Let d be a derivation under the rollback semantics. Then d is finite.*

Proof. First note that uctrl(d) is finite. Indeed, the number of transitions in uctrl(d) is bounded by the length of histories. In addition to lifting the uncontrolled steps, the controlled semantics also takes some administrative steps. If we show that between each pair of uncontrolled steps there is a finite amount of administrative steps, the thesis follows. Let us consider the rollback semantics. The administrative steps are taken by rule REQUEST, and for each of them the target process p' is computed using the bkDep operator.

This operator can create a finite number of requests because it inspects the histories of all the processes looking for dependencies only if the system is blocked. The number of dependencies is finite because it depends on the messages, alive tuples, and ghost tuples (that are all finite). Then the thesis follows. □

A main stepping stone to show minimality is proving that all transitions executed by the rollback semantics are consequences of the target action we want to undo. The concept of consequence is formalized using the happened-before relation [57], extended to cover both message passing and imperative primitives.

Definition 23 (Happened-before)**.** *Given a derivation d and two forward transitions, $t_1 = (s_1 \rightharpoonup_{p_1,r_1,k_1} s_1')$ and $t_2 = (s_2 \rightharpoonup_{p_2,r_2,k_2} s_2')$ in d, we say that t_1 happened before t_2, written as $t_1 \triangleright t_2$, if one of the following conditions holds:*

- *they both target the same process, i.e., $p_1 = p_2$ and t_1 occurs before t_2 in d;*
- *t_1 spawns a process p, i.e., $r_1 =$ SPAWN, and t_2 is performed by the process p;*
- *t_1 sends a message λ, and t_2 receives the same message λ;*
- *they both are on the map and either $\text{read}(k_1) \cap \text{write}(k_2) \neq \emptyset$, or $\text{read}(k_2) \cap \text{write}(k_1) \neq \emptyset$, or $\text{write}(k_1) \cap \text{write}(k_2) \neq \emptyset$ and t_1 occurs before t_2 in d.*

Furthermore, if $t_1 \triangleright t_2$ and $t_2 \triangleright t_3$, then $t_1 \triangleright t_3$ (transitivity). Two transitions t_1 and t_2 are independent if $t_1 \not\triangleright t_2$ and $t_2 \not\triangleright t_1$.

Independence is related to our notion of concurrency.

Lemma 5 (Independence vs concurrency)**.** *Two consecutive transitions $t_1 = (s_1 \rightharpoonup_{p_1,r_1,k_1} s_2)$ and $t_2 = (s_2 \rightharpoonup_{p_2,r_2,k_2} s_3)$ are independent iff $\underline{t_1}$ and t_2 are concurrent.*

Proof. This can be seen by contrasting the definition of concurrent transitions (Definition 18) and of happened-before relation, using the Loop Lemma (Lemma 1) to ensure the existence of inverse transitions. □

The next result is a version of the Square Lemma for consecutive, independent transitions.

Lemma 6 (Switching Lemma)**.** *Given two transitions of the form*

$$t_1 = (s_1 \rightleftharpoons_{p_1,r_1,k_1} s_2) \qquad \textit{and} \qquad t_2 = (s_2 \rightleftharpoons_{p_2,r_2,k_2} s_3)$$

such that $\underline{t}_1$ *and* t_2 *are concurrent, there exist a system* s_4 *and two transitions*

$$t_1' = (s_4 \rightleftharpoons_{p_1,r_1,k_1} s_3) \qquad \textit{and} \qquad t_2' = (s_1 \rightleftharpoons_{p_2,r_2,k_2} s_4)$$

Proof. The proof from [38, Lemma 2.9] applies. It relies on the Loop Lemma (Lemma 1) and the Square Lemma (Lemma 2). □

We can now show that all the uncontrolled transitions executed as a consequence of a rollback request depend on the action that needs to be undone. Notice that transitions executed during rollback are backwards, while the happened-before relation is defined on forward transitions, hence we need to consider the reverse transitions.

Theorem 2. *Take a controlled system in rollback mode* $c = [\![\mathcal{S}]\!]_{\{p,\psi\}}$ *and a maximal derivation* d *with* $\mathrm{init}(d) = c$*. Let us call* t *the last transition in* $\mathrm{uctrl}(d)$*. We have that* t *satisfies* $\{p, \psi\}$*, and for each transition* t' *in* $\mathrm{uctrl}(d)$*,* $\underline{t} \triangleright \underline{t'}$*.*

Proof. We adapt the proof from [38, Theorem 5.4], which is based on the following observations:

- a derivation under the rollback semantics only terminates when the request at the bottom of the stack is removed, and this is always the original request $\{p, \psi\}$;
- the controlled semantics executes steps from the backward semantics, and when it gets stuck a new request is generated using rule Request.

Both observations follow directly from rule inspection. The two observations combined ensure that t satisfies $\{p, \psi\}$. The relation $\underline{t} \triangleright \underline{t'}$ follows by transitivity, noticing that two transitions taken by the same process are in the relation, and that when a transition of the process p is not enabled and operator bkDep is invoked, the target computed by bkDep corresponds to a transition which is a consequence of the blocked one. □

The two results below are needed to prove minimality.

Lemma 7 (Shortening Lemma)**.** *Let* d_1 *and* d_2 *be co-initial and co-final derivations, such that* d_2 *is a forward derivation while* d_1 *contains at least one backward transition. Then, there exists a forward derivation* d_1' *of length strictly less than that of* d_1 *such that* $d_1' \approx d_1$ *.*

Proof. We prove this lemma by induction on the length of d_1. Thanks to the Parabolic Lemma (Lemma 3) there exist a backward derivation $\underline{d}$ and a forward derivation d' such that $d_1 \approx \underline{d}; d'$. Furthermore, $\underline{d}; d'$ is not longer than d_1. Let $\underline{t}t_0$ be the only two successive transitions in $\underline{d}; d'$ with opposite direction. Note that $\underline{t}$ is backwards, hence it removes an element from the history of some process. We will show below that there is in d' a transition $\tilde{t}$ which adds the same element to the history of the same process as above.

Moreover, we can swap $\tilde{t}$ with all the transitions between $\underline{t}$ and $\tilde{t}$, in order to obtain a derivation in which $\underline{t}$ and the swapped version of $\tilde{t}$ are adjacent. To do the swapping we apply the Switching Lemma (Lemma 6), since for all transitions t' in between, we have that t' and the (possibly swapped version of) $\tilde{t}$ are concurrent (this is proved below too). Since the swapped version of $\tilde{t}$ undoes the effect of $\underline{t}$, when adjacent to it, it coincides with t. Hence, we can remove both of them using $\approx$. The resulting derivation is strictly shorter, thus the claim follows by inductive hypothesis.

Let us now prove the results used above. Thanks to the Loop Lemma (Lemma 1) we have the derivations above iff we have two forward derivations which are co-initial (with s_2 as initial state) and co-final: $d; d_2$ and d'. Since the first transition of $d; d_2$ is $t : s_2 \rightharpoonup_{p_1,r_1,k_1} s_1$, which adds some item k_1 to the history of p_1 and such an item is never removed (since the derivation is forwards), then the same item k_1 has to be added also by a transition in d', otherwise the two derivations cannot be co-final. The earliest transition in d' adding item k_1 is exactly $\tilde{t}$.

Let us now justify that for each transition t' before $\tilde{t}$ in d' we have that $\underline{t'}$ and (the possibly swapped version of) $\tilde{t}$ are concurrent. First, t' is a forward transition and it should be applied to a process which is different from p_1, otherwise the item k_1 would be added in the wrong position in the history of p_1.

We consider the following cases:

- If t' applies rule Spawn to create a process p, then $\tilde{t}$ should not apply to process p since the process p_1 to which $\tilde{t}$ applies already existed before t', since p_1 was involved in the action at the beginning of d.
- If t' applies rule Send to send a message to some process p, then $\tilde{t}$ cannot receive the same message since the received message necessarily existed before (after the corresponding $\overline{\text{Receive}}$ has been performed).
- If t' applies rule RegisterS to create a tuple with unique identifier u, then $\tilde{t}$ cannot access a tuple with the same unique identifier u, since the tuple with identifier u' accessed by $\tilde{t}$ necessarily existed before, since it was involved in the transition at the beginning of d.
- If t' applies rule UnregisterS or rule EndUn to delete the tuple $\langle a, p', u, \top \rangle$, then $\tilde{t}$ cannot apply a read or register to an atom a or a pid p' since the tuple read or created necessarily existed before.
- If t' applies rule Registered to read the map M, then $\tilde{t}$ cannot modify M (delete or create a tuple) since the map M is necessarily the same as before.
- If t' applies a read rule, then t' and $\tilde{t}$ are concurrent for the same reason as for the rule Registered.
- If t' applies some other rule, then t' and $\tilde{t}$ are clearly concurrent.

In all the cases we have that $\underline{t'}$ and $\tilde{t}$ are concurrent, hence the thesis follows. □

Proposition 1 (Backward confluence)**.** *Let $\mathcal{S}$ be a system in the reversible semantics. If $\mathcal{S} \rightleftharpoons^* \mathcal{S}_1$ and $\mathcal{S} \rightleftharpoons^* \mathcal{S}_2$ then there exists $\mathcal{S}'$ such that $\mathcal{S}_1 \leftharpoondown^* \mathcal{S}'$ and $\mathcal{S}_2 \leftharpoondown^* \mathcal{S}'$.*

Proof. The proof from [38, Proposition 5.5] applies. It relies on the Loop Lemma (Lemma 1) and the Parabolic Lemma (Lemma 3). □

We can finally prove minimality.

Theorem 3 (Minimality)**.** *Let d be a controlled derivation such that* $\text{init}(d) = [\![\mathcal{S}]\!]_{\{p,\psi\}}$. *Derivation* $\text{uctrl}(d)$ *has minimal length among all uncontrolled derivations d' with* $\text{init}(d') = \mathcal{S}$ *including at least one transition satisfying* $\{p, \psi\}$.

Proof. The proof from [38, Theorem 5.6] applies. It relies on Proposition 1, the Shortening lemma (Lemma 7) and Theorem 2. □

3.6 Replay Semantics

In this section we extend the replay semantics (described in Section 2.1.5), as we see previously we have before to define a logging semantics that creates a log from the execution of a given program and a causal-consistent replay semantics that replays an execution following the log. Relation $\hookrightarrow_{p,\ell}$ defines the logging semantics, which decorates the standard semantics with subscripts $_{p,\ell}$ denoting that item ℓ needs to be stored in the log of process p, as described below. The rules of the logging semantics for the imperative primitives are available in Fig. 3.10. Log events are required to keep track of the causal links between actions of different processes; here we consider the following five kinds:

- $\mathbf{reg}(a, p, t, \mathrm{M})$, which logs the register of a tuple $\langle a, p, t, \top \rangle$, together with the map M containing the dead tuples on the atom a or the process p present in the map when the register was executed. This is needed to ensure that the transition would not be swapped with other operations on the map involving the same atom a or pid p. See Example 7 for more details.
- $\mathbf{del}(t, \mathrm{M})$, where t is the unique identifier of the deleted tuple and M is the set of dead tuples on the same atom or pid.
- $\mathbf{sendS}(\lambda, \mathrm{M})$, where λ is the unique identifier of the message sent and M is the alive tuple read to find the target process of the send.
- $\mathbf{rd}(\mathrm{M})$, where M is the part of the system's map involved in the read.
- $\mathbf{rdF}(\iota, \mathrm{M})$, where ι is the atom or pid that was not available, and M is the set of dead tuples on the atom or pid ι.
- $\mathbf{rdM}(\mathrm{M})$, where M is the system's map read by the registered primitive.

Now we extend the Definition 5 to covers also the imperative primitives log events.

Definition 24 (Log)**.** *A log is a (finite) sequence of events $(\ell_1, \ell_2, \dots)$ where each ℓ_i is either* $\mathbf{send}(\lambda)$, $\mathbf{rec}(\lambda)$, $\mathbf{spawn}(p)$, $\mathbf{sendS}(\lambda, \mathrm{M})$, $\mathbf{reg}(a, p, t, \mathrm{M})$, $\mathbf{del}(t, \mathrm{M})$, $\mathbf{rd}(\mathrm{M})$, $\mathbf{rdF}(\iota, \mathrm{M})$ *or* $\mathbf{rdM}(\mathrm{M})$, *with λ a message identifier, p a pid, t a tuple identifier, a an atom, ι either an atom or a pid, and* M *a set of tuples. Logs are ranged over by ω. Given a derivation*

$$\textsc{RegisterS}\ \frac{\theta, e, S \xrightarrow{\mathsf{register}(\kappa,a,p')} \theta', e', S' \qquad \mathrm{M}_a = \emptyset \qquad \mathrm{M}_{p'} = \emptyset \qquad \mathsf{isAlive}(p', \Pi)}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi; \mathrm{M} \hookrightarrow_{p,\mathbf{reg}(a,p',t,\mathrm{M}^a \cup \mathrm{M}^{p'})} \Gamma; \langle p, \theta', e'\{\kappa \to \mathsf{true}\}, S'\rangle \mid \Pi; \mathrm{M} \cup \{\langle a, p', t, \top\rangle\}}$$

$$\textsc{UnregisterS}\ \frac{\theta, e, S \xrightarrow{\mathsf{unregister}(\kappa,a)} \theta', e', S' \qquad \mathrm{M}_a = \{\langle a, p', t, \top\rangle\}}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi; \mathrm{M} \hookrightarrow_{p,\mathbf{del}(t,\mathrm{M}^a \cup \mathrm{M}^{p'})} \Gamma; \langle p, \theta', e'\{\kappa \to \mathsf{true}\}, S'\rangle \mid \Pi; \mathrm{M} \setminus \mathrm{M}_a \cup \mathrm{kill}(\mathrm{M}_a)}$$

$$\textsc{EndUn}\ \frac{e \text{ is a value } \vee\, e = \epsilon \qquad \mathrm{M}_p = \{\langle a, p, t, \top\rangle\}}{\Gamma; \langle p, \theta, e, [\,]\rangle \mid \Pi; \mathrm{M} \hookrightarrow_{p,\mathbf{del}(t,\mathrm{M}^a \cup \mathrm{M}^{p'})} \Gamma; \langle p, \theta, \bot, [\,]\rangle \mid \Pi; \mathrm{M} \setminus \mathrm{M}_p \cup \mathrm{kill}(\mathrm{M}_p)}$$

$$\textsc{RegisterF}\ \frac{\theta, e, S \xrightarrow{\mathsf{register}(\kappa,a,p')} \theta', e', S' \qquad \mathrm{M}_a \neq \emptyset \vee \mathrm{M}_{p'} \neq \emptyset \vee \neg\ \mathsf{isAlive}(p', \Pi)}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi; \mathrm{M} \hookrightarrow_{p,\mathbf{rd}(\mathrm{M}_a \cup \mathrm{M}_{p'})} \Gamma; \langle p, \theta, \epsilon, [\,]\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{UnregisterF}\ \frac{\theta, e, S \xrightarrow{\mathsf{unregister}(\kappa,a)} \theta', e', S' \qquad \mathrm{M}_a = \emptyset}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi; \mathrm{M} \hookrightarrow_{p,\mathbf{rdF}(a,\mathrm{M}^a)} \Gamma; \langle p, \theta, \epsilon, [\,]\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{SendS}\ \frac{\theta, e, S \xrightarrow{\mathsf{send}(a,v)} \theta', e', S' \qquad \mathrm{M}_a = \{\langle a, p', t, \top\rangle\}}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi; \mathrm{M} \hookrightarrow_{p,\mathbf{sendS}(\lambda,\mathrm{M}_a)} \Gamma \cup \{(p, p', \{v\})\}; \langle p, \theta', e', S'\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{SendF}\ \frac{\theta, e, S \xrightarrow{\mathsf{send}(a,v)} \theta', e', S' \quad \mathrm{M}_a = \emptyset}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi; \mathrm{M} \hookrightarrow_{p,\mathbf{rdF}(a,\mathrm{M}^a)} \Gamma; \langle p, \theta, \epsilon, [\,]\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{Whereis1}\ \frac{\theta, e, S \xrightarrow{\mathsf{whereis}(\kappa,a)} \theta', e', S' \qquad \mathrm{M}_a = \{\langle a, p', t, \top\rangle\}}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi; \mathrm{M} \hookrightarrow_{p,\mathbf{rd}(\mathrm{M}_a)} \Gamma; \langle p, \theta', e'\{\kappa \to p'\}, S'\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{Whereis2}\ \frac{\theta, e, S \xrightarrow{\mathsf{whereis}(\kappa,a)} \theta', e', S' \qquad \mathrm{M}_a = \emptyset}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi; \mathrm{M} \hookrightarrow_{p,\mathbf{rdF}(a,\mathrm{M}^a)} \Gamma; \langle p, \theta', e'\{\kappa \to \mathsf{undefined}\}, S'\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{Registered}\ \frac{\theta, e, S \xrightarrow{\mathsf{registered}(\kappa)} \theta', e', S' \qquad \mathrm{registered}(\mathrm{M}) = atoms}{\Gamma; \langle p, \theta, e, S\rangle \mid \Pi; \mathrm{M} \hookrightarrow_{p,\mathbf{rdM}(\mathrm{M})} \Gamma; \langle p, \theta', e'\{\kappa \to atoms\}, S'\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{End}\ \frac{e \text{ is a value } \vee\, e = \epsilon \qquad \mathrm{M}_p = \emptyset}{\Gamma; \langle p, \theta, e, [\,]\rangle \mid \Pi; \mathrm{M} \hookrightarrow_{p,\mathbf{rdF}(p,\mathrm{M}^p)} \Gamma; \langle p, \theta, \bot, [\,]\rangle \mid \Pi; \mathrm{M}}$$

Figure 3.10: Logging semantics of the imperative primitives

$d = (s_0 \hookrightarrow_{p_1,\ell_1} s_1 \hookrightarrow_{p_2,\ell_2} \ldots \hookrightarrow_{p_n,\ell_n} s_n)$, $n \geq 0$, under the logging semantics, the log associated to a pid p in d, in symbols $\mathcal{L}(d,p)$, is inductively defined as follows:

$$\mathcal{L}(d,p) = \begin{cases} () & \text{if } n = 0 \\ \ell_1 + \mathcal{L}(s_1 \hookrightarrow^* s_n, p) & \text{if } n > 0,\ p_1 = p \text{ and } \ell_1 \notin \{\mathbf{seq}, \mathbf{self}\} \\ \mathcal{L}(s_1 \hookrightarrow^* s_n, p) & \text{otherwise} \end{cases}$$

The log of d, written $\mathcal{L}(d)$, is defined as $\mathcal{L}(d) = \{(p, \mathcal{L}(d,p)) \mid p \text{ occurs in } d\}$. We sometimes call $\mathcal{L}(d)$ the global log of d to avoid confusion with $\mathcal{L}(d,p)$. Note that $\mathcal{L}(d,p) = \omega$ if $(p,\omega) \in \mathcal{L}(d)$ and $\mathcal{L}(d,p) = ()$ otherwise.

We can now extend the forward and backward semantics of Section 3.3 to be driven by a log. The forward semantics with logs and histories computes forwards using the log to replay a computation which is causally equivalent to the one captured using the logging semantics (cf. Lemma 10). This semantics consumes the log and produces the history. Dually, the backward semantics with logs and histories consumes the history and re-creates the log. The rules of the forward semantics with logs and histories for the imperative primitives are depicted in Fig. 3.11. Processes in the forward semantics with logs and histories now have the generic shape $\langle p, \omega, h, \theta, e, S\rangle$, and the same for the corresponding backward semantics.

The forward semantics with logs and histories makes sure that the causal order of the execution present in the log is respected. In other words, it ensures that non-deterministic choices are resolved in the same way as in the original computation. This is fundamental to ensure that a faulty behavior recorded in the original execution shows up also in the replay (cf. Theorem 5).

Let us now discuss the rules in Fig. 3.11. Rules RegisterS, UnregisterS, EndUn behave as their forward counterparts but also ensure, through the condition $\mathrm{M}^a \cup \mathrm{M}^{p'} = \mathrm{M}'$, that the part of the map concerning the atom a and the pid p' is exactly the same as it was when the log was recorded.

Moreover, rule RegisterS permanently modifies the map (if the added tuple is removed it would not disappear but it would become a ghost, and if it is re-added it would have a different unique identifier), hence we need to ensure that no process still needs to read the map as it currently is (at least as far as tuples or processes it is interested in are concerned). We ensure this property via the side condition $\mathrm{noRead}(a, p', t, \Pi, \mathcal{L})$. Predicate $\mathrm{noRead}(a, p', t, \Pi, \mathcal{L})$ holds if there is no item in the logs of the processes in Π, or in the processes to be spawned from them, which requires to read tuples on the atom a or the process p' (where a registered read any atom or pid) without reading a tuple with unique identifier t. Instead, the rules UnregisterS and EndUn ensure, through the predicate $\mathrm{noReadt}(\mathrm{M}, \Pi, \mathcal{L})$, that in the logs of the processes in Π (or spawned from them) there are no read of the alive tuples in M. All the other rules, i.e., the ones below the line, read the map but do not modify it, so the only additional check required, w.r.t. the forward reversible rules, is to verify that the map is the same as when the log was recorded.

Example 7. Consider a system composed of four processes, p_1, p_2, p_3, p_4, that have executed, respectively, a Registered, RegisterS, SendS and an UnregisterS, in that specific order,

$$\textsc{RegisterS}\ \frac{\theta, e, S \xrightarrow{\mathsf{register}(\kappa,a,p')} \theta', e', S' \qquad \mathrm{M}_a = \emptyset \qquad \mathrm{M}_{p'} = \emptyset \qquad \mathsf{isAlive}(p', \Pi) \qquad \mathrm{M}^a \cup \mathrm{M}^{p'} = \mathrm{M}' \qquad \mathrm{noRead}(a, p', t, \Pi, \mathcal{L})}{\Gamma; \langle p, \mathbf{reg}(a, p', t, \mathrm{M}'){:}\omega, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \twoheadrightarrow \Gamma; \langle p, \omega, \mathbf{regS}(\theta, e, S, \{\langle a, p', t, \top\rangle\}){:}h, \theta', e'\{\kappa \to \mathsf{true}\}, S'\rangle \mid \Pi; \mathrm{M} \cup \{\langle a, p', t, \top\rangle\}}$$

$$\textsc{UnregisterS}\ \frac{\theta, e, S \xrightarrow{\mathsf{unregister}(\kappa,a)} \theta', e', S' \qquad \mathrm{M}_a = \{\langle a, p', t, \top\rangle\} \qquad \mathrm{M}^a \cup \mathrm{M}^{p'} = \mathrm{M}' \qquad \mathrm{noReadt}(t, \Pi, \mathcal{L})}{\Gamma; \langle p, \mathbf{del}(t, \mathrm{M}'){:}\omega, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \twoheadrightarrow \Gamma; \langle p, \omega, \mathbf{del}(\theta, e, S, \mathrm{M}_a, \mathrm{M}^a \cup \mathrm{M}^{p'}){:}h, \theta', e'\{\kappa \to \mathsf{true}\}, S'\rangle \mid \Pi; \mathrm{M} \setminus \mathrm{M}_a \cup \mathrm{kill}(\mathrm{M}_a)}$$

$$\textsc{EndUn}\ \frac{e \text{ is a value } \vee\, e = \epsilon \qquad \mathrm{M}_p = \{\langle a, p, t, \top\rangle\} \qquad \mathrm{M}^a \cup \mathrm{M}^p = \mathrm{M}' \qquad \mathrm{noReadt}(t, \Pi, \mathcal{L})}{\Gamma; \langle p, \mathbf{del}(t, \mathrm{M}'){:}\omega, h, \theta, e, [\,]\rangle \mid \Pi; \mathrm{M} \twoheadrightarrow \Gamma; \langle p, \omega, \mathbf{del}(\theta, e, [\,], \mathrm{M}_p, \mathrm{M}^a \cup \mathrm{M}^p){:}h, \theta, \bot, [\,]\rangle \mid \Pi; \mathrm{M} \setminus \mathrm{M}_p \cup \mathrm{kill}(\mathrm{M}_p)}$$

$$\textsc{RegisterF}\ \frac{\theta, e, S \xrightarrow{\mathsf{register}(\kappa,a,p')} \theta', e', S' \qquad \mathrm{M}_a \neq \emptyset \vee \mathrm{M}_{p'} \neq \emptyset \vee \neg\ \mathsf{isAlive}(p', \Pi) \qquad \mathrm{M}_a \cup \mathrm{M}_{p'} = \mathrm{M}'}{\Gamma; \langle p, \mathbf{rd}(\mathrm{M}'){:}\omega, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \twoheadrightarrow \Gamma; \langle p, \omega, \mathbf{readS}(\theta, e, S, \mathrm{M}_a \cup \mathrm{M}_{p'}){:}h, \theta, \epsilon, [\,]\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{UnregisterF}\ \frac{\theta, e, S \xrightarrow{\mathsf{unregister}(\kappa,a)} \theta', e', S' \qquad \mathrm{M}_a = \emptyset \qquad \mathrm{M}^a = \mathrm{M}'}{\Gamma; \langle p, \mathbf{rdF}(a, \mathrm{M}'){:}\omega, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \twoheadrightarrow \Gamma; \langle p, \omega, \mathbf{readF}(\theta, e, S, a, \mathrm{M}^a){:}h, \theta, \epsilon, [\,]\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{SendS}\ \frac{\theta, e, S \xrightarrow{\mathsf{send}(a,v)} \theta', e', S' \qquad \mathrm{M}_a = \{\langle a, p', t, \top\rangle\} = \mathrm{M}'}{\Gamma; \langle p, \mathbf{sendS}(\lambda, \mathrm{M}'){:}\omega, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \twoheadrightarrow \Gamma \cup \{(p, p', \{v, \lambda\})\}; \langle p, \omega, \mathbf{sendS}(\theta, e, S, \{v, \lambda\}, \mathrm{M}_a){:}h, \theta', e', S'\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{SendF}\ \frac{\theta, e, S \xrightarrow{\mathsf{send}(a,v)} \theta', e', S' \qquad \mathrm{M}_a = \emptyset \qquad \mathrm{M}^a = \mathrm{M}'}{\Gamma; \langle p, \mathbf{rdF}(a, \mathrm{M}'){:}\omega, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \twoheadrightarrow \Gamma; \langle p, \omega, \mathbf{readF}(\theta, e, S, a, \mathrm{M}^a){:}h, \theta, \epsilon, [\,]\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{Whereis1}\ \frac{\theta, e, S \xrightarrow{\mathsf{whereis}(\kappa,a)} \theta', e', S' \qquad \mathrm{M}_a = \{\langle a, p', t, \top\rangle\} = \mathrm{M}'}{\Gamma; \langle p, \mathbf{rd}(\mathrm{M}'){:}\omega, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \twoheadrightarrow \Gamma; \langle p, \omega, \mathbf{readS}(\theta, e, S, \mathrm{M}_a){:}h, \theta', e'\{\kappa \to p'\}, S'\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{Whereis2}\ \frac{\theta, e, S \xrightarrow{\mathsf{whereis}(\kappa,a)} \theta', e', S' \qquad \mathrm{M}_a = \emptyset \qquad \mathrm{M}^a = \mathrm{M}'}{\Gamma; \langle p, \mathbf{rdF}(a, \mathrm{M}'){:}\omega, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \twoheadrightarrow \Gamma; \langle p, \omega, \mathbf{readF}(\theta, e, S, a, \mathrm{M}^a){:}h, \theta', e'\{\kappa \to \mathsf{undefined}\}, S'\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{Registered}\ \frac{\theta, e, S \xrightarrow{\mathsf{registered}(\kappa)} \theta', e', S' \qquad \mathsf{registered}(\mathrm{M}) = atoms \qquad \mathrm{M} = \mathrm{M}'}{\Gamma; \langle p, \mathbf{rdM}(\mathrm{M}'){:}\omega, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \twoheadrightarrow \Gamma; \langle p, \omega, \mathbf{readM}(\theta, e, S, \mathrm{M}){:}h, \theta', e'\{\kappa \to atoms\}, S'\rangle \mid \Pi; \mathrm{M}}$$

$$\textsc{End}\ \frac{e \text{ is a value } \vee\, e = \epsilon \qquad \mathrm{M}_p = \emptyset \qquad \mathrm{M}^p = \mathrm{M}'}{\Gamma; \langle p, \mathbf{rdF}(p, \mathrm{M}'){:}\omega, h, \theta, e, [\,]\rangle \mid \Pi; \mathrm{M} \twoheadrightarrow \Gamma; \langle p, \omega, \mathbf{readF}(\theta, e, [\,], p, \mathrm{M}^p){:}h, \theta, \bot, [\,]\rangle \mid \Pi; \mathrm{M}}$$

Figure 3.11: Forward semantics with logs and histories: the imperative primitives

and all targeting the same atom a. The log of the computation, computed using the logging semantics, is as follows:

$$\mathcal{L}(d) = \{\{p_1, \mathbf{rdM}(\emptyset)\}, \{p_2, \mathbf{reg}(a,p,t,\emptyset)\}, \{p_3, \mathbf{sendS}(\lambda, \{\langle a,p,t,\top\rangle\})\}, \{p_4, \mathbf{del}(t,\emptyset)\}, \dots\}$$

By using the forward semantics with logs and histories we can derive only the following computation:

$$\begin{array}{rl}
 & \emptyset;\ \begin{array}{l}\langle p_1, \mathbf{rdM}(\emptyset){:}\omega_1, h_1, \theta_1, \mathsf{registered}(), S_1\rangle \\ \mid \langle p_2, \mathbf{reg}(a,p,t,\emptyset){:}\omega_2, h_2, \theta_2, \mathsf{register(a,p)}, S_2\rangle \\ \mid \langle p_3, \mathbf{sendS}(\lambda, \langle a,p,t,\top\rangle){:}\omega_3, h_3, \theta_3, a\ !\ v, S_3\rangle \\ \mid \langle p_4, \mathbf{del}(t,\emptyset){:}\omega_4, h_4, \theta_4, \mathsf{unregister(a)}, S_4\rangle\end{array}\ ;\emptyset \\
\twoheadrightarrow & \emptyset;\ \begin{array}{l}\langle p_1, \omega_1, \mathbf{readM}(\theta_1, \mathsf{registered}(), S_1, \emptyset){:}h_1, \theta_1', e_1', S_1'\rangle \\ \mid \langle p_2, \mathbf{reg}(a,p,t,\emptyset){:}\omega_2, h_2, \theta_2, \mathsf{register(a,p)}, S_2\rangle \\ \mid \langle p_3, \mathbf{sendS}(\lambda, \langle a,p,t,\top\rangle){:}\omega_3, h_3, \theta_3, a\ !\ v, S_3\rangle \\ \mid \langle p_4, \mathbf{del}(t,\emptyset){:}\omega_4, h_4, \theta_4, \mathsf{unregister(a)}, S_4\rangle\end{array}\ ;\emptyset \\
\twoheadrightarrow & \emptyset;\ \begin{array}{l}\langle p_1, \omega_1, \mathbf{readM}(\theta_1, \mathsf{registered}(), S_1, \emptyset){:}h_1, \theta_1', e_1', S_1'\rangle \\ \mid \langle p_2, \omega_2, \mathbf{regS}(\theta_2, \mathsf{register(a,p)}, S_2, \{\langle a,p,t,\top\rangle\}){:}h_2, \theta_2', e_2', S_2'\rangle \\ \mid \langle p_3, \mathbf{sendS}(\lambda, \langle a,p,t,\top\rangle){:}\omega_3, h_3, \theta_3, a\ !\ v, S_3\rangle \\ \mid \langle p_4, \mathbf{del}(t,\emptyset){:}\omega_4, h_4, \theta_4, \mathsf{unregister(a)}, S_4\rangle\end{array}\ ;\{\{a,p,t,\top\}\} \\
\twoheadrightarrow & \{(p_3,p,\{v,\lambda\})\};\ \begin{array}{l}\langle p_1, \omega_1, \mathbf{readM}(\theta_1, \mathsf{registered}(), S_1, \emptyset){:}h_1, \theta_1', e_1', S_1'\rangle \\ \mid \langle p_2, \omega_2, \mathbf{regS}(\theta_2, \mathsf{register(a,p)}, S_2, \{\langle a,p,t,\top\rangle\}){:}h_2, \theta_2', e_2', S_2'\rangle \\ \mid \langle p_3, \omega_3, \mathbf{sendS}(\theta_3, a\ !\ v, S_3, \{v,\lambda\}, \{\langle a,p,t,\top\rangle\}){:}h_3, \theta_3', e_3', S_3'\rangle \\ \mid \langle p_4, \mathbf{del}(t,\emptyset){:}\omega_4, h_4, \theta_4, \mathsf{unregister(a)}, S_4\rangle\end{array}\ ;\{\{a,p,t,\top\}\} \\
\twoheadrightarrow & \{(p_3,p,\{v,\lambda\})\};\ \begin{array}{l}\langle p_1, \omega_1, \mathbf{readM}(\theta_1, \mathsf{registered}(), S_1, \emptyset){:}h_1, \theta_1', e_1', S_1'\rangle \\ \mid \langle p_2, \omega_2, \mathbf{regS}(\theta_2, \mathsf{register(a,p)}, S_2, \{\langle a,p,t,\top\rangle\}){:}h_2, \theta_2', e_2', S_2'\rangle \\ \mid \langle p_3, \omega_3, \mathbf{sendS}(\theta_3, a\ !\ v, S_3, \{v,\lambda\}, \{\langle a,p,t,\top\rangle\}){:}h_3, \theta_3', e_3', S_3'\rangle \\ \mid \langle p_4, \omega_4, \mathbf{del}(\theta_4, \mathsf{unregister(a)}, S_4, \{\langle a,p,t,\bot\rangle\}, \emptyset){:}h_4, \theta_4', e_4', S_4'\rangle\end{array}\ ;\{\{a,p,t,\bot\}\}
\end{array}$$

Let us analyze the mechanisms that ensure that the causal order provided in the log is respected. The registered needs to be executed first, since for register the condition $\mathrm{noRead}(a,p,t,\Pi,\mathcal{L})$ fails, while the send and the unregister cannot trigger since the required tuple is not in the map. For this last reason, after the registered operation, only the register can trigger. At this point, we need to execute the send. Indeed, the unregister is not enabled since the side condition $\mathrm{noReadt}(t,\Pi,\mathcal{L})$ fails due to the send in the log of p_3. Now, it is finally possible to execute the unregister. In this example, even if each primitive is executed by a different process, thanks to the log the execution above is the only possible one. In general, given a log, multiple executions could be possible. This would happen if some actions were concurrent.

$\Diamond$

The forward semantics with logs and histories is paired with a backward semantics with logs and histories which consumes histories and recreates the log. It coincides with the backward semantics, but for the addition of logs. We report in Fig. 3.12 the semantics of the imperative

$\overline{\text{RegisterS}}$ $\Gamma; \langle p, \omega, \mathbf{regS}(\theta, e, S, \{\langle a, p', t, \top\rangle\}){:}h, \theta', e', S'\rangle \mid \Pi; \mathrm{M} \cup \{\langle a, p', t, \top\rangle\} \twoheadleftarrow \Gamma; \langle p, \mathbf{reg}(a, p', t, \mathrm{M}^a \cup \mathrm{M}^{p'}){:}\omega, h, \theta, e, S\rangle \mid \Pi; \mathrm{M}$ if $\mathrm{readop}(t, \Pi) = \emptyset$

$\overline{\text{Del}}$ $\Gamma; \langle p, \omega, \mathbf{del}(\theta, e, S, \{\langle a, p', t, \top\rangle\}, \mathrm{M}_1){:}h, \theta', e', S'\rangle \mid \Pi; \mathrm{M} \cup \{\langle a, p', t, \bot\rangle\} \twoheadleftarrow \Gamma; \langle p, \mathbf{del}(t, \mathrm{M}_1){:}\omega, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \cup \{\langle a, p', t, \top\rangle\}$ if $\mathrm{M}_a = \emptyset \wedge \mathrm{M}_{p'} = \emptyset \wedge \mathrm{readmap}(\mathrm{M} \cup \{\langle a, p', t, \bot\rangle\}, \Pi) = \emptyset \wedge \mathrm{readghost}(t, \Pi) = \emptyset \wedge \mathrm{M}_1 = \mathrm{M}^a \cup \mathrm{M}^{p'}$

$\overline{\text{ReadS}}$ $\Gamma; \langle p, \omega, \mathbf{readS}(\theta, e, S, \mathrm{M}_1){:}h, \theta', e', S'\rangle \mid \Pi; \mathrm{M} \twoheadleftarrow \Gamma; \langle p, \mathbf{rd}(\mathrm{M}_1){:}\omega, h, \theta, e, S\rangle \mid \Pi; \mathrm{M}$ if $\mathrm{M}_1 \subseteq \mathrm{M}$

$\overline{\text{SendS}}$ $\Gamma \cup \{(p, p', \{v, \lambda\})\}; \langle p, \omega, \mathbf{sendS}(\theta, e, S, \{v, \lambda\}, \mathrm{M}_1){:}h, \theta', e', S'\rangle \mid \Pi; \mathrm{M} \twoheadleftarrow \Gamma; \langle p, \mathbf{sendS}(\lambda, \mathrm{M}_1){:}\omega, h, \theta, e, S\rangle \mid \Pi; \mathrm{M}$ if $\mathrm{M}_1 \subseteq \mathrm{M}$

$\overline{\text{ReadF}}$ $\Gamma; \langle p, \omega, \mathbf{readF}(\theta, e, S, \iota, \mathrm{M}_1){:}h, \theta', e', S'\rangle \mid \Pi; \mathrm{M} \twoheadleftarrow \Gamma; \langle p, \mathbf{rdF}(\iota, \mathrm{M}_1){:}\omega, h, \theta, e, S\rangle \mid \Pi; \mathrm{M}$ if $\mathrm{M}_\iota = \emptyset \wedge \mathrm{M}_1 = \mathrm{M}^\iota$

$\overline{\text{ReadM}}$ $\Gamma; \langle p, \omega, \mathbf{readM}(\theta, e, S, \mathrm{M}_1){:}h, \theta', e', S'\rangle \mid \Pi; \mathrm{M} \twoheadleftarrow \Gamma; \langle p, \mathbf{rdM}(\mathrm{M}_1){:}\omega, h, \theta, e, S\rangle \mid \Pi; \mathrm{M}$ if $\mathrm{M}_1 = \mathrm{M}$

Figure 3.12: Backward semantics with logs and histories: the imperative primitives

$$\text{U-Satisfy}\ \frac{\mathcal{S} \twoheadrightarrow_{p,r,k} \mathcal{S}' \ \wedge\ \psi \in \mathrm{sat}(k)}{[\![\mathcal{S}]\!]_{\{p,\psi\}:\Psi} \rightsquigarrow [\![\mathcal{S}']\!]_{\Psi}} \qquad \text{U-Act}\ \frac{\mathcal{S} \twoheadrightarrow_{p,r,k} \mathcal{S}' \ \wedge\ \psi \notin \mathrm{sat}(k)}{[\![\mathcal{S}]\!]_{\{p,\psi\}:\Psi} \rightsquigarrow [\![\mathcal{S}']\!]_{\{p,\psi\}:\Psi}}$$

$$\text{Spawn}\ \frac{\mathcal{S} = \Gamma; \Pi; \mathrm{M} \ \wedge\ p \notin \Pi \ \wedge\ p' = \mathrm{parent}(\mathcal{L}(d), p)}{[\![\mathcal{S}]\!]_{\{p,\psi\}:\Psi} \rightsquigarrow [\![\mathcal{S}']\!]_{\{p',\mathsf{spawn}(p)\}:\{p,\psi\}:\Psi}}$$

$$\text{Request}\ \frac{\mathcal{S} = \Gamma; \langle p, \omega, h, \theta, e, S\rangle \mid \Pi; \mathrm{M} \ \wedge\ \mathcal{S} \not\twoheadrightarrow_{p,r,k} \ \wedge\ \mathrm{fwDep}(\omega, \mathcal{S}) = \{p', \psi'\}}{[\![\mathcal{S}]\!]_{\{p,\psi\}:\Psi} \rightsquigarrow [\![\mathcal{S}']\!]_{\{p',\psi'\}:\{p,\psi\}:\Psi}}$$

Figure 3.13: Replay semantics

primitives, which extends the one in Fig. 3.6. We refer to [38] for the semantics of the other constructs.

In Section 3.5 we exploited the backward semantics to define a rollback operator, permitting to undo an action far in the past, together with all and only its consequences. Dually, here we define a *replay semantics*, allowing one to redo a future action, including all and only its causes. Replaying the causes is needed to ensure that we reach consistent systems; e.g., if a process p needs to redo a receive of some message with the unique identifier λ, we first need to ensure that the message with the unique identifier λ has been sent, otherwise we would reach a system where p has received a message that was never sent.

Formally, a system in replay mode is denoted as $[\![\mathcal{S}]\!]_{\{p,\psi\}}$, where $\{p, \psi\}$ is the request for a process p to perform ψ. The replay requests $\{p, \psi\}$ are analogous to the ones we presented in Section 3.5 for rollback. For instance, a replay request $\{p, \mathsf{reg}(t)\}$ asks to redo the register made by a process p that inserted a tuple $\langle \_, \_, t, \top\rangle$ in M.

Similarly to rollback, we write $[\![\mathcal{S}]\!]_{\Psi}$ where Ψ is the sequence of requests that need to be satisfied to redo the desired action; Ψ can be seen as a stack where the first element is the first request that will be satisfied by the replay semantics and once the stack is empty the required action has been redone. If the first request cannot be satisfied, then it means that some causes have not been replayed yet, so new requests, aimed at replaying these causes, will be pushed on top of the stack.

Fig. 3.13 depicts the replay semantics, built on top of the forward semantics with logs and histories. The rules are described in Section 2.1.5 and they are shown Fig. 2.10. It is possible to

$$
\begin{array}{lll}
\text{fwDep}(\mathbf{rec}(\lambda){:}\omega, \Gamma; \langle p', \omega', \_, \_, \_, \_\rangle \mid \Pi; \_) & = \{p', \mathsf{send}(\lambda)\} & \text{if } \mathbf{send}(\lambda) \in \omega' \vee \mathbf{sendS}(\lambda, \_) \in \omega' \\
\text{fwDep}(\mathbf{sendS}(\_, \{\langle \_, \_, t, \_\rangle\}){:}\omega, \_; \Pi; \mathrm{M}) & = \{p', \mathsf{regS}(t)\} & \text{if } \{\langle \_, \_, t, \top\rangle\} \notin \mathrm{M} \wedge p' = \text{getLP}(\mathsf{regS}, t, \Pi) \\
\text{fwDep}(\mathbf{rd}(\mathrm{M}' \cup \{\langle \_, \_, t, \top\rangle\}){:}\omega, \_; \Pi; \mathrm{M}) & = \{p', \mathsf{regS}(t)\} & \text{if } \{\langle \_, \_, t, \top\rangle\} \notin \mathrm{M} \wedge p' = \text{getLP}(\mathsf{regS}, t, \Pi) \\
\text{fwDep}(\mathbf{rdF}(\_, \mathrm{M}' \cup \{\langle \_, \_, t, \bot\rangle\}){:}\omega, \_; \Pi; \mathrm{M}) & = \{p', \mathsf{del}(t)\} & \text{if } \{\langle \_, \_, t, \bot\rangle\} \notin \mathrm{M} \wedge p' = \text{getLP}(\mathsf{del}, t, \Pi) \\
\text{fwDep}(\mathbf{rdM}(\mathrm{M}' \cup \{\langle \_, \_, t, \top\rangle\}){:}\omega, \_; \Pi; \mathrm{M}) & = \{p', \mathsf{regS}(t)\} & \text{if } \{\langle \_, \_, t, \top\rangle\} \notin \mathrm{M} \wedge p' = \text{getLP}(\mathsf{regS}, t, \Pi) \\
\text{fwDep}(\mathbf{rdM}(\mathrm{M}' \cup \{\langle \_, \_, t, \bot\rangle\}){:}\omega, \_; \Pi; \mathrm{M}) & = \{p', \mathsf{del}(t)\} & \text{if } \{\langle \_, \_, t, \bot\rangle\} \notin \mathrm{M} \wedge p' = \text{getLP}(\mathsf{del}, t, \Pi) \\
\text{fwDep}(\mathbf{reg}(\_, \_, t, \mathrm{M}' \cup \{\langle \_, \_, t', \bot\rangle\}){:}\omega, \_; \Pi; \mathrm{M}) & = \{p', \mathsf{del}(t')\} & \text{if } \{\langle \_, \_, t', \bot\rangle\} \notin \mathrm{M} \wedge p' = \text{getLP}(\mathsf{del}, t', \Pi) \\
\text{fwDep}(\mathbf{reg}(a, p, t, \_){:}\omega, \_; \Pi; \_) & = \{p', s\} & \text{if } p' = \text{getLP}(\mathsf{rghost}, a, p, t, \Pi) \\
\text{fwDep}(\mathbf{del}(t, \_){:}\omega, \_; \Pi; \mathrm{M}) & = \{p', \mathsf{regS}(t)\} & \text{if } \{\langle \_, \_, t, \top\rangle\} \notin \mathrm{M} \wedge p' = \text{getLP}(\mathsf{regS}, t, \Pi) \\
\text{fwDep}(\mathbf{del}(t, \mathrm{M}' \cup \langle \_, \_, t', \bot\rangle){:}\omega, \_; \Pi; \mathrm{M}) & = \{p', \mathsf{del}(t')\} & \text{if } \{\langle \_, \_, t', \bot\rangle\} \notin \mathrm{M} \wedge p' = \text{getLP}(\mathsf{del}, t', \Pi) \\
\text{fwDep}(\mathbf{del}(t, \_){:}\omega, \_; \Pi; \mathrm{M} \cup \{\langle \_, \_, t, \top\rangle\}) & = \{p', s\} & \text{if } p' = \text{getLP}(\mathsf{read}, t, \Pi)
\end{array}
$$

Figure 3.14: Forward dependencies operator

$$
\begin{array}{lll}
\text{getLP}(\mathsf{del}, t, \langle p, \omega, \_, \_, \_, \_\rangle \mid \Pi) & = p & \text{if } \mathbf{del}(\{\langle \_, \_, t, \_\rangle\}, \_) \in \omega \\
\text{getLP}(\mathsf{regS}, t, \langle p, \omega, \_, \_, \_, \_\rangle \mid \Pi) & = p & \text{if } \mathbf{reg}(\{\langle \_, \_, t, \_\rangle\}, \_) \in \omega \\
\text{getLP}(\mathsf{read}, t, \langle p, \omega, \_, \_, \_, \_\rangle \mid \Pi) & = p & \text{if } \begin{cases} \mathbf{rd}(\mathrm{M} \cup \{\langle \_, \_, t, \top\rangle\}) \in \omega \\ \vee\ \mathbf{rdM}(\mathrm{M} \cup \{\langle \_, \_, t, \top\rangle\}) \in \omega \\ \vee\ \mathbf{sendS}(\_, \mathrm{M}' \cup \{\langle \_, \_, t, \top\rangle\}) \in \omega \end{cases} \\
\text{getLP}(\mathsf{rghost}, a, p', t, \langle p, \omega, \_, \_, \_, \_\rangle \mid \Pi) & = p & \text{if } \begin{cases} (\mathbf{rdF}(a, \mathrm{M}) \in \omega \wedge \langle \_, \_, t, \_\rangle \notin \mathrm{M}) \\ \vee\ (\mathbf{rdF}(p', \mathrm{M}') \in \omega \wedge \langle \_, \_, t, \_\rangle \notin \mathrm{M}') \\ \vee\ (\mathbf{rdM}(\mathrm{M}'') \in \omega \wedge \langle \_, \_, t, \_\rangle \notin \mathrm{M}'') \end{cases}
\end{array}
$$

Figure 3.15: Get pid operator

see that previously there was a rule for replay the receive request (Request-Rec). Now in Fig. 3.7 we consider only one rule Request similar to its counterpart in the rollback semantics. It is applied when the target process cannot execute because some causes have yet to be replayed (indicated by $\not\twoheadrightarrow_{p,r,k}$)., for e.g., a SendS cannot be replayed if the target atom has yet to be registered. With the help of the operator fwDep (defined in Fig. 3.14) the rule computes a new request, aimed at solving the dependency. It is then pushed on Ψ.

The operator fwDep uses the getLP operator to find the pid of a process which is the target of the dependency. Note that function getHP looks for dependencies inside histories while getLP looks for dependencies inside logs. In Fig. 3.14, we define the fwDep operator. Let us discuss the different cases:

rec: a receive cannot be executed since the message received has yet to be sent by some other process p', so a request has to be made to p' to execute the send (via either rule Send or rule SendS) of the (unique) message identified by λ.

sendS: a send of a message using an atom cannot be executed since the atom has yet to be registered by some other process p', so a request has to be made to the process p' to execute the register of the (unique) tuple identified by t.

rd: in this case we need to restore the portion of the map (one or two tuples) read by some *read* operation. These are alive tuples which have not been created yet, hence we need to create them via a register operation.

rdF: in this case we need to restore the ghost tuples in the portion of the map read by the

operation, hence we need to perform the deletion of the tuples which are not present in the system map.

rdM: in this case we need to restore the map as read by the registered primitive. In the read map we may have both alive tuples which have not been created yet, and in this case we need to create them via a register operation, and dead tuples which are not present, and in this case we need to perform their deletion.

reg: in the case of the register of a tuple t it is necessary to execute all the operations on tuples in M′ (carried by the log), namely deletion of a tuple (if it is no more in the system's map), read operations on tuples and read operations on the whole map. In this case we use as request just to take a step, this is enough since if the dependency is not solved the same condition will trigger again. This avoids introducing an auxiliary dependency target. As usual, the pid p' is obtained from the operator getLP.

del: in the case of deletion of a tuple t we may have as dependencies: (i) a tuple on the same atom or pid, whose register needs to be executed before doing the deletion, unless it is a pre-existing ghost (which is not in M); (ii) a read operation on the live tuple of t (done using requests for doing single steps till needed) (iii) a read operation on the whole map.

The pid p' that identifies the process that needs to satisfy the request is computed using the auxiliary operator getLP, defined in Fig. 3.15. In particular, operator getLP takes a descriptor of a log item, a unique identifier t and a pool of processes, and finds the pid of a process which will do an operation compatible with the descriptor on the tuple with unique identifier t. Notice that more than one pid may fit the descriptor (e.g., many processes may read the same tuple), hence the operator non-deterministically selects one.

One could base the rollback on the backward semantics with logs and histories, thus re-creating logs while going backwards using rollback. This can be obtained by using in the premises of the rules of the rollback semantics (cf. Fig. 3.7) the backward semantics with logs and histories instead of the backward semantics.

We now present an example to clarify how replay requests are managed.

Example 8. We consider the same system as Example 7, with the same log, computed using the logging semantics. The log is hence as follows:

$$\mathcal{L}(d) = \{\{p_1, \mathbf{rdM}(\emptyset)\}, \{p_2, \mathbf{reg}(a,p,t,\emptyset)\}, \{p_3, \mathbf{sendS}(\lambda, \{\langle a,p,t,\top\rangle\})\}, \{p_4, \mathbf{del}(t,\emptyset)\}, \dots\}$$

We consider the replay request $\{p_3, \mathsf{send}(\lambda)\}$, asking to replay the send of the message with identifier λ traced in the log of p_3. The system in replay mode is $[\![\mathcal{S}]\!]_{\{p,\mathsf{send}(\lambda)\}}$, where:

$$\mathcal{S} = \emptyset;\ \begin{array}{l}\langle p_1, \mathbf{rdM}(\emptyset){:}\omega_1, h_1, \theta_1, \mathsf{registered}(), S_1\rangle \\ \mid \langle p_2, \mathbf{reg}(a,p,t,\emptyset){:}\omega_2, h_2, \theta_2, \mathsf{register}(\mathsf{a},\mathsf{p}), S_2\rangle \\ \mid \langle p_3, \mathbf{sendS}(\lambda, \langle a,p,t,\top\rangle){:}\omega_3, h_3, \theta_3, a\ !\ v, S_3\rangle \\ \mid \langle p_4, \mathbf{del}(t,\emptyset){:}\omega_4, h_4, \theta_4, \mathsf{unregister}(\mathsf{a}), S_4\rangle\end{array};\emptyset$$

We cannot immediately replay the requested send, since rule SendS is not applicable to system $\mathcal{S}$ ($\mathcal{S} \not\twoheadrightarrow_{p_3,r,k}$). Indeed, the map does not contain the element of the map used by the send.

As a result, we can only apply rule Request, which adds the request $\{p_2, \mathsf{regS}(t)\}$ (computed using the case analysis in Fig. 3.14, where the pid of the target process p_2 is computed using the operator getLP in Fig. 3.15). Now, the system becomes $[\![\mathcal{S}]\!]_{\{p_2,\mathsf{regS}(t)\}:\{p_3,\mathsf{send}(\lambda)\}}$. Again, it is not possible to apply rule RegisterS to satisfy the request $\{p_2, \mathsf{regS}(t)\}(\mathcal{S} \not\twoheadrightarrow_{p_2,r,k})$, hence we need a second application of rule Request. The newly added request is now $\{p_1, s\}$.

Now the system is $[\![\mathcal{S}]\!]_{\{p_1,s\}:\{p_2,\mathsf{regS}(t)\}:\{p_3,\mathsf{send}(\lambda)\}}$, where it is possible to apply rule U-Satisfy to execute the step:

$$\mathcal{S} = \emptyset;\ \begin{array}{l}\langle p_1, \mathbf{rdM}(\emptyset){:}\omega_1, h_1, \theta_1, \mathsf{registered}(), S_1\rangle \\ \mid \langle p_2, \mathbf{reg}(a,p,t,\emptyset){:}\omega_2, h_2, \theta_2, \mathsf{register}(\mathsf{a},\mathsf{p}), S_2\rangle \\ \mid \langle p_3, \mathbf{sendS}(\lambda, \langle a,p,t,\top\rangle){:}\omega_3, h_3, \theta_3, a\ !\ v, S_3\rangle \\ \mid \langle p_4, \mathbf{del}(t,\emptyset){:}\omega_4, h_4, \theta_4, \mathsf{unregister}(\mathsf{a}), S_4\rangle\end{array};\emptyset$$

$$\twoheadrightarrow \emptyset;\ \begin{array}{l}\langle p_1, \omega_1, \mathbf{readM}(\theta_1, \mathsf{registered}(), S_1, \emptyset){:}h_1, \theta_1', e_1', S_1'\rangle \\ \mid \langle p_2, \mathbf{reg}(a,p,t,\emptyset){:}\omega_2, h_2, \theta_2, \mathsf{register}(\mathsf{a},\mathsf{p}), S_2\rangle \\ \mid \langle p_3, \mathbf{sendS}(\lambda, \langle a,p,t,\top\rangle){:}\omega_3, h_3, \theta_3, a\ !\ v, S_3\rangle \\ \mid \langle p_4, \mathbf{del}(t,\emptyset){:}\omega_4, h_4, \theta_4, \mathsf{unregister}(\mathsf{a}), S_4\rangle\end{array};\emptyset \quad = \mathcal{S}'$$

The system is now $[\![\mathcal{S}']\!]_{\{p_2,\mathsf{regS}(t)\}:\{p_3,\mathsf{send}(\lambda)\}}$, where it is possible to apply again rule U-Satisfy to execute the step:

$$\mathcal{S}' = \emptyset;\ \begin{array}{l}\langle p_1, \omega_1, \mathbf{readM}(\theta_1, \mathsf{registered}(), S_1, \emptyset){:}h_1, \theta_1', e_1', S_1'\rangle \\ \mid \langle p_2, \mathbf{reg}(a,p,t,\emptyset){:}\omega_2, h_2, \theta_2, \mathsf{register}(\mathsf{a},\mathsf{p}), S_2\rangle \\ \mid \langle p_3, \mathbf{sendS}(\lambda, \langle a,p,t,\top\rangle){:}\omega_3, h_3, \theta_3, a\ !\ v, S_3\rangle \\ \mid \langle p_4, \mathbf{del}(t,\emptyset){:}\omega_4, h_4, \theta_4, \mathsf{unregister}(\mathsf{a}), S_4\rangle\end{array};\emptyset$$

$$\twoheadrightarrow \emptyset;\ \begin{array}{l}\langle p_1, \omega_1, \mathbf{readM}(\theta_1, \mathsf{registered}(), S_1, \emptyset){:}h_1, \theta_1', e_1', S_1'\rangle \\ \mid \langle p_2, \omega_2, \mathbf{regS}(\theta_2, \mathsf{register}(\mathsf{a},\mathsf{p}), S_2, \{\langle a,p,t,\top\rangle\}){:}h_2, \theta_2', e_2', S_2'\rangle \\ \mid \langle p_3, \mathbf{sendS}(\lambda, \langle a,p,t,\top\rangle){:}\omega_3, h_3, \theta_3, a\ !\ v, S_3\rangle \\ \mid \langle p_4, \mathbf{del}(t,\emptyset){:}\omega_4, h_4, \theta_4, \mathsf{unregister}(\mathsf{a}), S_4\rangle\end{array};\{\{a,p,t,\top\}\} \quad = \mathcal{S}''$$

The system is now $[\![\mathcal{S}'']\!]_{\{p_3,\mathsf{send}(\lambda)\}}$ and a final application of rule U-Satisfy satisfies the last request:

$$\mathcal{S}'' = \emptyset;\ \begin{array}{l}\langle p_1, \omega_1, \mathbf{readM}(\theta_1, \mathsf{registered}(), S_1, \emptyset){:}h_1, \theta_1', e_1', S_1'\rangle \\ \mid \langle p_2, \omega_2, \mathbf{regS}(\theta_2, \mathsf{register}(\mathsf{a},\mathsf{p}), S_2, \{\langle a,p,t,\top\rangle\}){:}h_2, \theta_2', e_2', S_2'\rangle \\ \mid \langle p_3, \mathbf{sendS}(\lambda, \langle a,p,t,\top\rangle){:}\omega_3, h_3, \theta_3, a\ !\ v, S_3\rangle \\ \mid \langle p_4, \mathbf{del}(t,\emptyset){:}\omega_4, h_4, \theta_4, \mathsf{unregister}(\mathsf{a}), S_4\rangle\end{array};\{\{a,p,t,\top\}\}$$

$$\twoheadrightarrow \{(p_3,p,\{v,\lambda\})\};\ \begin{array}{l}\langle p_1, \omega_1, \mathbf{readM}(\theta_1, \mathsf{registered}(), S_1, \emptyset){:}h_1, \theta_1', e_1', S_1'\rangle \\ \mid \langle p_2, \omega_2, \mathbf{regS}(\theta_2, \mathsf{register}(\mathsf{a},\mathsf{p}), S_2, \{\langle a,p,t,\top\rangle\}){:}h_2, \theta_2', e_2', S_2'\rangle \\ \mid \langle p_3, \omega_3, \mathbf{sendS}(\theta_3, a\ !\ v, S_3, \{v,\lambda\}, \{\langle a,p,t,\top\rangle\}){:}h_3, \theta_3', e_3', S_3'\rangle \\ \mid \langle p_4, \mathbf{del}(t,\emptyset){:}\omega_4, h_4, \theta_4, \mathsf{unregister}(\mathsf{a}), S_4\rangle\end{array};\{\{a,p,t,\top\}\} \quad = \mathcal{S}'''$$

Since there are no more pending requests the replay ends.

◊

Now, let us focus on the properties of the replay semantics and of the relation $\rightsquigarrow$, which can be seen as a controlled version of the forward semantics with logs and histories ($\twoheadrightarrow$). Indeed, an execution of the replay operator corresponds to a forward derivation.

Theorem 4 (Soundness of replay)**.** *If* $[\![\mathcal{S}]\!]_\Psi \rightsquigarrow^* [\![\mathcal{S}']\!]_{\Psi'}$ *then* $\mathcal{S} \twoheadrightarrow^* \mathcal{S}'$.

Proof. The replay semantics is either executing forward steps using the forward semantics or executing administrative steps (i.e., pushing new requests on top of Ψ via rule Request or rule Spawn), which do not alter the underlying system. □

We want to define a system taking a log from the logging semantics.

Definition 25. *Let* $d = (s_1 \hookrightarrow^* s_2)$ *be a derivation under the logging semantics, with* $s_1 = \Gamma; \langle p_1, \theta_1, e_1, S_1\rangle \mid \cdots \mid \langle p_n, \theta_n, e_n, S_n\rangle; \mathrm{M}$. *The system corresponding to* s_1 *in the reversible semantics with logs and histories is defined as follows:*

$$\mathrm{addLog}(\mathcal{L}(d), s_1) = \Gamma; \langle p_1, \mathcal{L}(d, p_1), (), \theta_1, e_1, S_1\rangle \mid \cdots \mid \langle p_n, \mathcal{L}(d, p_n), (), \theta_n, e_n, S_n\rangle; \mathrm{M}$$

Conversely, given a system $s = \Gamma; \langle p_1, \omega_1, h_1, \theta_1, e_1, S_1\rangle \mid \cdots \mid \langle p_n, \omega_n, h_n, \theta_n, e_n, S_n\rangle; \mathrm{M}$ *in the reversible semantics with logs and histories, we let* $\mathsf{Proj}(s)$ *be the system obtained from* s *by removing both logs and histories, i.e.,* $\mathsf{Proj}(s) = \Gamma; \langle p_1, \theta_1, e_1, S_1\rangle \mid \cdots \mid \langle p_n, \theta_n, e_n, S_n\rangle; \mathrm{M}$. *It is extended to derivations in the obvious way: given a derivation* d *of the form* $s_1 \hookrightarrow_{p_1,\ell_1} \ldots \hookrightarrow_{p_n,\ell_n} s_n$, *we let* $\mathsf{Proj}(d)$ *be* $\mathsf{Proj}(s_1) \hookrightarrow_{p_1,\ell_1} \ldots \hookrightarrow_{p_n,\ell_n} \mathsf{Proj}(s_n)$.

Definition 26 (Well-initialized)**.** *A controlled system* $c = [\![s]\!]_{(\{p,\psi\})}$ *is* well-initialized *iff there exist a derivation* d *under the logging semantics, a system* $s_0 = \mathrm{addLog}(\mathcal{L}(d), \mathrm{init}(d))$, *an uncontrolled derivation* $s_0 \overset{\twoheadrightarrow}{\twoheadleftarrow}^* s$, *and an uncontrolled forward derivation from* s *satisfying* $\{p, \psi\}$. *A controlled derivation* d *is well-initialized iff* $\mathrm{init}(d)$ *is well-initialized.*

In order to study the properties of replay, we need to restrict our attention to requests that ask to replay transitions which are in the future of the process. The existence of a derivation satisfying the request can be efficiently checked. For replay requests $\{p, s\}$ it is enough to check that the process p can perform a step, for other replay requests it is enough to check the process log. We can now show that controlled derivations are finite.

Lemma 8. *Let* d *a well-initialized derivation. Then* d *is finite.*

Proof. The proof from [38, Lemma 5.3] applies. □

We want to compare different derivations using the same log. However, they may differ in how many actions they execute after the last action in the log. Indeed, steps derived using rules Seq or Self do not appear in the log. Thus, we restrict our attention to fully-logged derivations, which block just after the last action in the log.

Definition 27 (Fully-logged derivation)**.** *A derivation d under the forward semantics with logs and histories is* fully-logged *if, for each process p in* final(d)*, the log is empty and the last element in the history (if any) is a logged transition (i.e., one which is not labeled with* **seq** *nor* **self***).*

The next lemma shows that if the log is fixed, the behavior of each process is also fixed (but not the alternation between them).

Lemma 9 (Local determinism for the replay semantics)**.** *Let d_1, d_2 be co-initial fully-logged derivations under the forward semantics with logs and histories. Then, for each pid p occurring in d_1, d_2, we have $S_p^1 = S_p^2$ where S_p^1 (resp. S_p^2) is the ordered sequence of configurations $\langle p, \omega, h, \theta, e, S\rangle$ occurring in d_1 (resp. d_2), with consecutive equal elements collapsed.*

Proof. The proof from [38, Lemma 4.21] applies. □

Now, we can show that any reachable state in the replay semantics corresponds to a state reachable in the logging semantics:

Lemma 10. *Let d, d' be fully-logged derivations under the logging and the replay semantics with logs and histories, respectively. Let* $\mathrm{init}(d') = \mathrm{addLog}(\mathcal{L}(d), \mathrm{init}(d))$*. Then there exists a finite forward computation $d'' \approx d'$ such that $d = \mathsf{Proj}(d'')$.*

Proof. The proof from [38, Lemma 4.20] applies. □

Now it is possible to prove that if a behavior occurs in the logged derivation then the replay derivation also exhibits the same behavior (and vice-versa), hence replay is correct and complete. Notably, correctness and completeness do not depend on the scheduling.

Theorem 5 (Correctness and completeness)**.** *Let d be a fully-logged derivation under the logging semantics. Let d' be any fully-logged derivation under the replay semantics with logs and histories such that* $\mathrm{init}(d') = \mathrm{addLog}(\mathcal{L}(d), \mathrm{init}(d))$*. Then:*

- *there is a system $\Gamma; \Pi; \mathrm{M}$ in d with a configuration $\langle p, \theta, e, S\rangle$ in Π iff there is a system $\Gamma'; \Pi'; \mathrm{M}'$ in d' with a configuration $\langle p, \theta, e, S\rangle$ in $\mathsf{Proj}(\Gamma'; \Pi'; \mathrm{M}')$;*
- *there is a system $\Gamma; \Pi; \mathrm{M}$ in d with a message $(p, p', \{v, \lambda\})$ in Γ iff there is a system $\Gamma'; \Pi'; \mathrm{M}'$ in d' with a message $(p, p', \{v, \lambda\})$ in Γ'.*
- *there is a system $\Gamma; \Pi; \mathrm{M}$ in d with a tuple $\langle a, p, t, s\rangle$ in M iff there is a system $\Gamma'; \Pi'; \mathrm{M}'$ in d' with a tuple $\langle a, p, t, s\rangle$ in M'.*

Proof. From Lemma 10 there exists a forward derivation $d'' \approx d'$ such that $\mathsf{Proj}(d'') = d$. Trivially, the thesis holds for d''. Item (1) follows by applying local determinism (Lemma 9) to derivations d'' and d'. Concerning item (2), every message in Γ should either be in $\mathrm{init}(d)$ or must be sent by a transition of d. In the first case, the claim follows since $\mathsf{Proj}(\mathrm{init}(d')) = \mathrm{init}(d)$. In the second case, there should be a send action in d producing it, and the thesis follows from item (1), observing that since the derivation d' is fully-logged the send action is replayed. Concerning item (3), every tuple in M should either be in $\mathrm{init}(d)$ or must be registered and/or deleted by transitions of d. In the first case, the claim follows since $\mathsf{Proj}(\mathrm{init}(d')) = \mathrm{init}(d)$. In the second

case, there should be a register and/or a delete action in d producing/removing it, and the thesis follows from item (1), observing that since the derivation d' is fully-logged the register and/or delete actions are replayed. □

Our controlled semantics is not only causal-consistent but also *minimal*. This means that in order to satisfy the oldest request on Ψ, and consequently every other request in Ψ, the replay executes the minimal amount of actions needed to satisfy the replay request. Now we extend function uctrl() to also project systems in replay mode to uncontrolled ones:

$$\text{uctrl}(\llbracket \mathcal{S} \rrbracket_{\Psi}) = \mathcal{S}$$

The extended notion of projection extends to derivations as well.

As for rollback, we need some auxiliary results.

Theorem 6. *For each well-initialized controlled system* $c = \llbracket \mathcal{S} \rrbracket_{\{p,\psi\}}$ *consider a maximal derivation* d *with* $\text{init}(d) = c$*. Let us call* t *the last transition in* $\text{uctrl}(d)$*. We have that* t *satisfies* $\{p, \psi\}$*, and for each transition* t' *in* $\text{uctrl}(d)$*,* $t' \triangleright t$*.*

Proof. The proof from [38, Theorem 5.4] applies. □

Proposition 2 (Forward confluence)**.** *Let* $\mathcal{S}$ *be a system in the uncontrolled reversible semantics. If* $\mathcal{S} \rightrightarrows^* \mathcal{S}_1$ *and* $\mathcal{S} \rightrightarrows^* \mathcal{S}_2$ *then there exists* $\mathcal{S}'$ *such that* $\mathcal{S}_1 \twoheadrightarrow^* \mathcal{S}'$ *and* $\mathcal{S}_2 \twoheadrightarrow^* \mathcal{S}'$

Proof. The proof from [38, Proposition 5.5] applies. It relies on the Loop Lemma (Lemma 1) and the Parabolic Lemma (Lemma 3). □

We can finally prove minimality.

Theorem 7 (Minimality)**.** *Let* d *be a well-initialized controlled derivation such that* $\text{init}(d) = \llbracket \mathcal{S} \rrbracket_{\{p,\psi\}}$*. Derivation* $\text{uctrl}(d)$ *has the minimal length among all uncontrolled derivations* d' *with* $\text{init}(d') = \mathcal{S}$ *including at least one transition satisfying* $\{p, \psi\}$*.*

Proof. The proof from [38, Theorem 5.6] applies. It relies on Proposition 2, the Shortening Lemma (Lemma 7) and Theorem 6. □

We now show the concrete impact of the result above.

Example 9. Consider the computation in Example 8, which needs to replay the send operation. It consists of three steps. Theorem 7 above guarantees that this is the minimal length of a computation replaying the send. Indeed, both the other executed transitions (the read and the register) are causes of the send, hence they need to be executed first in any uncontrolled computation. Notably, the delete operation, which is not a cause of the send, is not executed.

$$\textsc{RegisterS}\ \frac{\theta, e, S \xrightarrow{\mathsf{register}(\kappa,a,p')} \theta', e', S' \qquad t \text{ fresh} \qquad \mathrm{M}_a = \emptyset \qquad \mathrm{M}_{p'} = \emptyset \qquad \mathsf{isAlive}(p', \Pi)}{\Gamma; \langle nid, p, h, \theta, e, S\rangle \mid \Pi; \{\langle nid, \mathrm{M}\rangle\} \cup \mathcal{M}; \Omega \rightharpoonup \Gamma; \langle nid, p, \mathbf{regS}(\theta, e, S, \{\langle a, p', t, \top\rangle\}){:}h, \theta', e'\{\kappa \to \mathsf{true}\}, S'\rangle \mid \Pi; \{\langle nid, \mathrm{M} \cup \{\langle a, p', t, \top\rangle\}\}\} \cup \mathcal{M}; \Omega}$$

$$\textsc{RegisterS}\ \frac{}{\Gamma; \langle nid, p, \mathbf{regS}(\theta, e, S, \{\langle a, p', t, \top\rangle\}){:}h, \theta', e', S'\rangle \mid \Pi; \{\langle nid, \mathrm{M} \cup \{\langle a, p', t, \top\rangle\}\}\} \cup \mathcal{M}; \Omega \leftharpoonup \Gamma; \langle nid, p, h, \theta, e, S\rangle \mid \Pi; \{\langle nid, \mathrm{M}\rangle\} \cup \mathcal{M}; \Omega} \quad \text{if } \mathsf{readop}(t, \Pi) = \emptyset$$

Figure 3.16: Distributed semantics, sample rules

3.7 CauDEr with Imperative Primitives

We exploited the theory presented above to extend CauDEr [5, 41, 39, 40], a Causal-consistent reversible Debugger for Erlang. CauDEr is written in Erlang and provides a graphical interface for user interaction. Previously CauDEr supported only the sequential, concurrent and distributed fragment of Erlang, and we now have added support for the imperative primitives. The updated code can be found at [58].

While the theory discussed so far does not consider distribution, we have extended the version of CauDEr supporting distribution [41], where systems can be composed of multiple nodes. This is rendered by adding to each process the *node identifier* nid of the node where it is running. Also, the description of the system now includes the set of identifiers Ω of the nodes in the network. As far as the imperative primitives are concerned, the only difference is that each node has its own map, shared only among its processes. Thus, we replace the map M with a set of pairs $\langle nid, \mathrm{M}\rangle$, denoted as $\mathcal{M}$. Each pair associates a map M to a node identifier *nid*. The imperative primitives we consider work only on the map of the node where the process is running. As an example, Fig. 3.16 shows how to update rule RegisterS. Note that the *nid* of the node of the process coincides with the one of the map it is acting on.

The interplay between the introduction of maps described in the present chapter and the Erlang primitives supporting distribution as formalized in [41] (to which we refer for a detailed description of the reversible semantics of the distribution primitives) is limited. In particular, updating the distribution rules in [41] requires only (i) the addition of $\mathcal{M}$ on both sides of each rule and (ii) to extend the rule for node creation (rule StartS in [41, Fig. 3]) to also create an empty map associated to the identifier of the new node. Additionally, if one wants to add support for the remote send primitive **{atom, node}!**expr (see [15] for details), which uses a pair to specify the target process, one needs to consider two aspects: (i) even if the target does not exist, no exception is raised and (ii) the **atom** is evaluated in the map of the target node. As a consequence of (ii), atoms may occur as part of the target of messages and the corresponding pid needs to be retrieved by the receiver of the message. As a result, other actions may occur in between the execution of the send and the evaluation of the **atom**.

Fig. 3.17 shows a snapshot of the new version of CauDEr, supporting the map. The interface is organized as follows. On the left, from top to bottom, we can see (i) the program under debugging, (ii) the state, call stack, history and log of the selected process, (iii) the map of the node of the current process (the main novelty in the interface due to our extension, highlighted with a red square). On the top-right we can find execution controls (they are grouped into multiple tabs, here we see the tab for rollback), and on the bottom-right information on the system structure and on the execution.

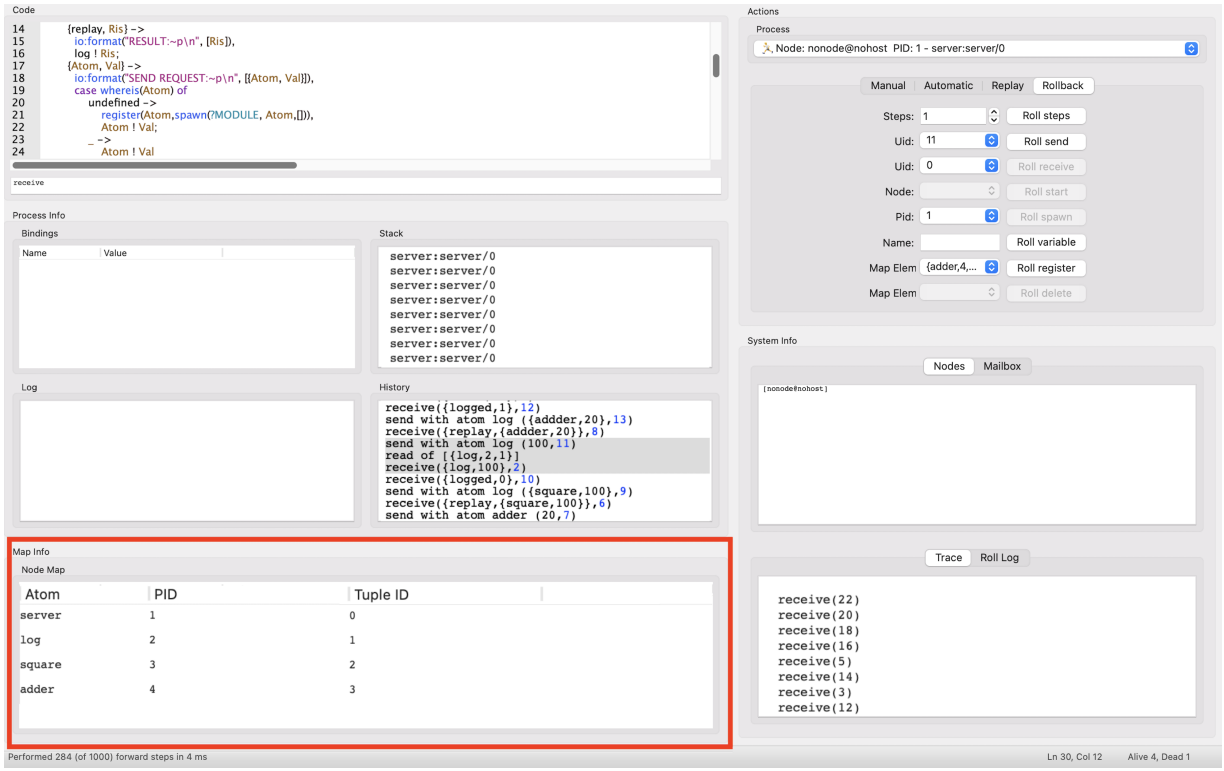


Figure 3.17: A screenshot of CauDEr

Debugging the case study CauDEr works as follows: the user selects the Erlang source file, then CauDEr loads the program and shows the source code to the user. Then, the user can choose the function that will act as an entry point, specify its arguments, and select the identifier of the node where the first process should run. The user can perform single steps of some process (both forwards and backwards), n steps in the chosen direction in an automatic modality (a scheduler decides which process will execute each step), or use the rollback or the replay operator (provided a log is available, which can be the result of previous executions inside CauDEr) described in the previous sections.

The rollback operator allows one to undo a selected action (e.g., the send of a given message) far in the past, including all and only its consequences. This corresponds to trigger a computation in our rollback semantics (cf. Fig. 3.7) with the selected target. As proved in Theorem 1 and Theorem 2 this corresponds to an uncontrolled computation undoing the selected action and only its consequences. All the consequences need to be undone thanks to causal safety, discussed at the end of Section 3.4. Hence, the rollback operator is convenient to look for a bug causing a visible misbehavior, as described below. We now put CauDEr at work on our case study.

We consider a server that dispatches requests to a set of mathematical services and logs the result of each request. We consider the code in Fig. 3.18.

The code begins with the definition of function main at lines 4-7. This function spawns and registers the processes corresponding to the server and the logger (lines 5-6), and finally sends the requests taken as input to the server through the sendRequest function.

The server (lines 9-26) waits for requests of the following forms:

- {logged,{Res,Time}} (lines 11-12): in this case, it prints out the value Time, which

corresponds to the time at which the result has been logged.

- {reply,Res} (lines 13-15): in this case, the server prints Res, the result of the mathematical service, and sends it to the logger (line 15).
- {Atom,Val} (lines 16-25): this case corresponds to a service request. The server prints the request (in line 17). If the service is already registered, then the server sends a request message to the service (lines 22-23). Otherwise, the server first creates and registers the new service and afterwards it sends the request to it (lines 19-21).

Finally, after handling a request, the server recursively invokes itself to handle the next request (line 26).

The logger (lines 28-32) keeps a log L of messages, each one equipped with a time (sequential integers for simplicity). Whenever it receives a message, it adds it to the log, and sends the information on the logged item to the server (line 31).

Our case study includes three services:

- the square service (lines 34-38) receives a value, computes its square, and sends the result to the server;
- the logarithm service (lines 40-44), analogously, receives a value, computes its logarithm, and sends the result to the server;
- the adder service (lines 46-51) receives a value, adds it to the accumulator N, and sends the result to the server. The accumulator stores the sum of all the numbers that the service has received; it starts at 0.

In our example, we invoke the program with the following list of requests: [{square, 10}, {adder, 20}, {log, 100}, {adder, 30}, {adder, 100}]
The two first requests are successfully answered, while the request to compute the logarithm of 100 is not.

By checking the history of the server (this is exactly the one shown in Fig. 3.17, where relevant items are grayed, most recent items are on top) we notice that the request to compute the logarithm of 100 has been sent by the server as message 11. By using CauDEr rollback facilities to undo the send of message 11 (including all and only its consequences), one notices that the send has been performed at line 24 (also visible in the screenshot, upon rollback the line becomes highlighted), which is used for already spawned services. This is wrong since this is the first request to compute a logarithm.

One can now require to rollback the register of atom log (used as target of the send). We can now see that the system logger has been registered under this atom in the main function (line 6 in Fig. 3.18):
register(log,spawn(?MODULE, logger,[0,[]])),
This is wrong. The bug is that the same atom has been used both for the system logger and for the logarithm service.

Finding such a bug without the support of reversibility, and rollback in particular, would not be easy. Also, rollback allows us to go directly to the points of interest (e.g., where the atom

log has been registered), even if we do not know which process performed the action. Notice that the function server, which sent message 11 using atom log as target, and the function main, which made the register of atom log, are executed by different processes, and CauDEr guided us from one process to the other. Hence, debugging via rollback scales better than standard techniques to larger programs, where finding a bug without reversibility would be even more difficult.

Notice also that every time one goes backwards, e.g., using the rollback operator, CauDEr creates a log of the part of the computation which has been undone. If one by mistake goes too far backwards, one can use the replay operator to go forwards again, till some event of interest is re-executed. If the misbehavior was visible in the original computation, it can be shown again using the replay semantics as proved in Theorem 5.

```
1  -module(server).
2  -export([main/1]).
3
4  main(A)->
5      register(server,spawn(?MODULE, server,[])),
6      register(log,spawn(?MODULE, logger,[0,[]])),
7      sendRequest(A).
8
9  server() ->
10     receive
11         {logged,Log} ->
12           io:format("LOGGED TIME:~p\n", [Log]);
13         {replay, Ris} ->
14           io:format("RESULT:~p\n", [Ris]),
15           log ! Ris;
16         {Atom, Val} ->
17           io:format("SEND REQUEST:~p\n", [{Atom, Val}]),
18           case whereis(Atom) of
19               undefined ->
20                   register(Atom,spawn(?MODULE, Atom,[])),
21                   Atom ! Val;
22               _ ->
23                   Atom ! Val
24           end
25     end,
26     server().
27
28 logger(N,L)->
29     receive
30       Val ->
31         server ! {logged,N}, logger(N+1,L++[Val])
32     end.
33
34 square() ->
35   receive
36     N ->
37       server ! {replay,{square,N*N}}, square()
38   end.
39
40 log() ->
41   receive
42     N ->
43       server ! {replay,{log,math:log10(N)}}, log()
44   end.
45
46 adder() -> adder(0).
47 adder(N)->
48   receive
49     Val ->
50       server ! {replay,{addder,Val + N}}, adder(Val + N)
51   end.
52
53 sendRequest([]) -> ok;
54 sendRequest([El | T]) ->
55   server ! El, sendRequest(T).
```

Figure 3.18: Case study

Chapter 4

A Meta-model for Memory Models

In this chapter, we present our framework for describing memory models. This framework is based on the concept of a synchronous product of LTSs and extends the framework proposed in [34] described in Section 2.2. The framework proposed in [34] defines how the components of its model interact, whereas our framework is described as a product of LTSs in which we explicitly define how the synchronization relation is composed (i.e., which describes how the various LTSs interact with each other). Furthermore, unlike [34], our system is described as a product of three LTSs (thread, memory and scheduler) instead of only two components (thread and memory). We believe that the use of three components makes our framework closer to the description of a real system and increases the modularity of the system.

The use of an LTS product instead of a fixed structure allows us to have more possible forms of synchronization than Effinger.

In this Chapter we present several instances: the sequential consistency memory model as a base; the write-buffer memory model as a well-studied model with a formalized operational semantics; and finally automatic mutual exclusion (AME) as an example of a transactional memory model, with three different semantics: the strong semantics where the semantics provides strong atomicity in the executions, the weak semantics where is added the possibility to interleave unprotected and protected step, and the weak semantics with optimistic rollback which gives also the possibility to rollback the code when a computation is blocked on a false condition.

The addition of the scheduler component gives us greater modularity in fact in the case of the weak memory model we can see how changing a single rule allows us to obtain a different model. In fact, as we shall see, changing a single rule in the scheduler of the strong semantics gives us the weak memory model.

The Chapter is structured as follows. Section 4.1 presents the state of the art. Section 4.2 introduces the notion of synchronous products of LTSs. In Section 4.3 we describe our framework. Finally, in Sections 4.4, 4.5 and 4.6 we outline and describe three instances of our framework and we prove the bi-similarity with the corresponding memory model in the literature.

4.1 State of the Art

Each programming language typically has its own memory model that reflects its specific design goals, trade-offs, and limitations [21]. This diversity derives from the different requirements and optimizations supported by each language, leading to a variety of memory models that meet the individual features of concurrent programming in those languages. While there are general principles that can guide the design of memory models, each programming language typically has its own distinct memory model tailored to its specific needs and optimizations.

There are several studies that describe memory models specific to various programming languages. For example, the memory models for Java [59, 60] and C/C++ [61] have been extensively studied. Most of these works use trace analysis and/or event structures to describe the system. Additionally, there are models described through operational semantics, such as transactions [16, 17], but these are focused on specific languages and memory models.

To the best of our knowledge, the only work that uses a modular approach to describe a variety of memory models is the framework proposed by [34]. This framework is characterized by its ability to represent different memory models in a composite way, offering higher modularity than traditional methods. In our work, we propose to extend this modular approach by introducing a framework based on the concept of synchronous product of LTSs. This allows us to represent the interactions between threads, memory and schedulers in a more accurate and detailed manner, bringing the theoretical model closer to the reality of complex systems.

4.2 Synchronous Product of Labeled Transition Systems

We begin by introducing the notion of a Labeled Transition System (LTS) with idle transitions. Idle transitions indicate that the LTS is not involved in any synchronizations. This concept will be used later in the synchronous product of LTSs to represent scenarios where a LTS does not participate in the synchronization process.

Definition 28 (LTS with idle transitions)**.** *Given a LTS $L = (\mathcal{S}, \Lambda, \rightarrow)$, we define the LTS with idle transitions as $L^\phi = (\mathcal{S}, \Lambda^\phi, \rightarrow^\phi)$, where the label ϕ denotes the idle transitions. Here, $\Lambda^\phi = \Lambda \cup \{\phi\}$, while $\rightarrow^\phi$ represents the set of possible transitions, including idle transitions, defined as $\rightarrow^\phi = \rightarrow \cup \{p \xrightarrow{\phi} p \mid p \in \mathcal{S}\}$.*

Note that an idle transition $p \xrightarrow{\phi} p$ is a transition that does not change the state of the LTS. Now, we introduce the concept of synchronous products of LTS, we follow the notations used in [62].

Definition 29 (Synchronous products of LTS)**.** *Let $L_1, \ldots, L_n$ be LTSs, we now consider the corresponding LTSs that also include idle transitions: $L_i^\phi = (\mathcal{S}_i, \Lambda_i^\phi, \rightarrow_i^\phi)$. A* synchronization relation *$(\mathcal{R})$ is $\mathcal{R} \subseteq \Lambda_1^\phi \times \cdots \times \Lambda_n^\phi \setminus \{\langle \phi_1, \ldots, \phi_n \rangle\}$. The synchronous product of $L_1, \ldots, L_n$ indicated with $L_1 \otimes_\mathcal{R} \ldots \otimes_\mathcal{R} L_n$ is defined as:*

- $\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$,
- *the labeled transition relation is defined by rule* Sync *below:*

$$\textsc{Sync}\ \frac{p_1 \xrightarrow{\ell_1} q_1 \quad \ldots \quad p_n \xrightarrow{\ell_n} q_n \quad \langle \ell_1, \ldots, \ell_n \rangle \in \mathcal{R}}{p_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n \xrightarrow{\langle \ell_1, \ldots, \ell_n \rangle} q_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} q_n}$$

where $p_i \xrightarrow{\ell_i} q_i \in \rightarrow_i^{\phi}$.

In the definition of synchronous products of LTSs, we include idle transitions as part of the LTSs. Consequently, in the synchronous product of LTSs, the LTSs executing a idle transition do not contribute to the transitions of the synchronous product. It is important to note that the transition $p_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n \xrightarrow{\langle \phi_1, \ldots, \phi_n \rangle} p_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n$ is not possible in the synchronous product, since no LTS would contribute to this transition.

4.3 A General Framework to Describe Memory Models

In this section we describe our framework which is based on the synchronous product of three LTSs (Definition 29), each representing a different component: *threads*, *memory*, and *scheduler*. In this framework, we define explicitly how to construct the synchronization relation among these components.

The framework is described using the synchronous product $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$, where:

- **Thread Semantics**: Represented by the LTS $T = (\mathcal{S}_T, \Lambda_T, \rightarrow_T)$, which defines the operational rules for the programming language, including syntax and operational rules for commands and expressions.
- **Memory Semantics**: Represented by the LTS $M = (\mathcal{S}_M, \Lambda_M, \rightarrow_M)$, which manages the representation and manipulation of memory, defining how variables, data structures, and other memory-related entities are stored, accessed, and updated.
- **Scheduler Semantics**: Represented by the LTS $S = (\mathcal{S}_S, \Lambda_S, \rightarrow_S)$, which controls the execution order of processes, defining how processes are scheduled, interact, and resolve concurrency issues.

After introducing the three components of the LTSs product, and before defining the synchronization relation, we categorize the labels of the LTSs into the following groups:

$\lambda_{\mathbf{thread}} \in (\Lambda_T \setminus \Lambda_M \setminus \Lambda_S)$ denotes a label exclusive to the thread semantics, representing an operation involving only the thread semantics;

$\lambda_{\mathbf{mem}} \in (\Lambda_M \setminus \Lambda_T \setminus \Lambda_S)$ denotes a label exclusive to the memory semantics, representing an operation involving only the memory semantics;

$\lambda_{\mathbf{sched}} \in (\Lambda_S \setminus \Lambda_T \setminus \Lambda_M)$ denotes a label exclusive to the scheduler semantics, representing an operation involving only the scheduler semantics;

$\lambda_{\mathbf{mt}} \in (\Lambda_T \cap \Lambda_M \setminus \Lambda_S)$ denotes a label common to both thread and memory semantics, representing an operation involving both memory and threads;

$\lambda_{\mathbf{st}} \in (\Lambda_S \cap \Lambda_T \setminus \Lambda_M)$ denotes a label common to both thread and scheduler semantics, representing an operation involving both scheduler and threads;

$\lambda_{\mathbf{sm}} \in (\Lambda_S \cap \Lambda_M \setminus \Lambda_T)$ denotes a label common to both memory and scheduler semantics, representing an operation involving both scheduler and memory;

$\lambda_{\mathbf{a}} \in (\Lambda_T \cap \Lambda_M \cap \Lambda_S)$ denotes a label common to all three semantics, representing an operation that involves the entire system.

Finally, to define the synchronous product of the LTSs that describe our model, we need to define the synchronization relation $\mathcal{R}$ that we consider:

$$\begin{aligned}\mathcal{R} = \{&\langle \lambda_{\text{thread}}, \phi, \phi \rangle,\\ &\langle \phi, \lambda_{\text{mem}}, \phi \rangle,\\ &\langle \phi, \phi, \lambda_{\text{sched}} \rangle,\\ &\langle \lambda_{\text{mt}}, \lambda_{\text{mt}}, \phi \rangle,\\ &\langle \lambda_{\text{st}}, \phi, \lambda_{\text{st}} \rangle,\\ &\langle \phi, \lambda_{\text{sm}}, \lambda_{\text{sm}} \rangle,\\ &\langle \lambda_{\text{a}}, \lambda_{\text{a}}, \lambda_{\text{a}} \rangle \quad \}\end{aligned}$$

The synchronization rule SYNC obtained is:

$$\text{SYNC}\ \frac{\mathcal{T} \xrightarrow{\ell_T} \mathcal{T}' \qquad \mathcal{M} \xrightarrow{\ell_M} \mathcal{M}' \qquad \mathcal{S} \xrightarrow{\ell_S} \mathcal{S}' \qquad \langle (\ell_T, \ell_M, \ell_S) \rangle \in \mathcal{R}}{\mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{(\ell_T, \ell_M, \ell_S)} \mathcal{T}' \otimes_{\mathcal{R}} \mathcal{M}' \otimes_{\mathcal{R}} \mathcal{S}'}$$

The division into three levels, with respect to [34] which considers only a division into two levels, offers some advantages. It allows us to develop dedicated models for both memory and the scheduler. The scheduler manages the interactions between processes, ensuring that complexity is distributed across multiple semantic layers rather than being confined to a single one. This modular approach simplifies the semantics of each component.

Moreover, dividing the system into three levels aligns more closely with the architecture of real-world systems. In actual implementations, language semantics, memory management, and process scheduling are often handled by distinct components.

In the following sections, we will show how to instantiate our framework to obtain memory models known from the literature. First, we define semantics for the language that the thread semantics will use to perform a single step. After that, we will describe the three semantics describing the LTSs corresponding to the thread, memory, and scheduler semantics. Then, we will show the labels used by the system and to which labels they correspond in our framework. Finally, we will show that our instances correspond to the memory model we wish to describe thanks to the bi-similarity relation between two LTS, the LTS corresponding to our framework, and the LTS of the model to which the memory model corresponds in the literature. Below we give the definitions of similarity and bi-similarity that we will use for the proof.

Definition 30 (F-Simulation)**.** *Given two LTSs $L_1 = (S_1, \Lambda_1, \rightarrow_1)$ and $L_2 = (S_2, \Lambda_2, \rightarrow_2)$, a function $f : \Lambda_1 \rightarrow \Lambda_2$ and a relation $R \subseteq S_1 \times S_2$ is a f-simulation (indicated with $L_1 \overset{f}{\prec}_R L_2$) if and only if for every pair of states $(p, q) \in R$ and all labels $\lambda \in \Lambda$:*

$$\textsc{Thread Step}\ \frac{e \overset{\alpha}{\hookrightarrow} e'}{\langle \theta, e\rangle \xrightarrow{(\theta,\alpha)} \langle \theta, e'\rangle}\ \alpha \neq \mathsf{sp}(\theta') \qquad \textsc{Thread Spawn}\ \frac{e \xhookrightarrow{\mathsf{sp}(\theta')} e'; e'' \qquad \theta' \text{ is fresh}}{\langle \theta, e\rangle \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \langle \theta, e'\rangle \mid \langle \theta', e''\rangle} \qquad \textsc{Thread Par}\ \frac{\mathcal{T} \xrightarrow{\ell_t} \mathcal{T}'}{\mathcal{T} \mid \mathcal{T}_1 \xrightarrow{\ell_t} \mathcal{T}' \mid \mathcal{T}_1}$$

$$\alpha = \mathsf{pure}, \mathsf{ref}(r, v), \mathsf{rd}(r, v), \mathsf{wr}(r, v), \mathsf{acq}(l), \mathsf{rel}(l)$$

Figure 4.1: Threads semantics

- *if* $p \xrightarrow{\lambda} p'$*, then there is* $q \overset{f(\lambda)}{\rightarrow} q'$ *such that* $(p', q') \in R$*.*

Definition 31 (F-Bisimulation)**.** *Given two LTSs* $L_1 = (S_1, \Lambda_1, \rightarrow_1)$ *and* $L_2 = (S_2, \Lambda_2, \rightarrow_2)$*, a function* $f : \Lambda_1 \rightarrow \Lambda_2$ *and a relation* $R \subseteq S_1 \times S_2$ *is a* f*-bisimulation (indicated with* $L_1 \overset{f}{\sim}_R L_2$*) if* $L_1 \overset{f}{\prec}_R L_2$ *and* $L_2 \overset{f^{-1}}{\prec}_{R^{-1}} L_1$

4.4 Instance: Sequential Consistency

Sequential consistency memory model [20] is a type of memory model used in parallel computing. In this model, the program order of memory operations within a single thread is maintained, meaning that memory operations on a single thread are executed in the same order in which they are defined in the program. This ordering is not affected by memory operations from other threads, which may be executed concurrently. However, the ordering of memory operations across multiple threads is not strictly enforced, meaning that there may be differences in the order in which memory operations are executed across different threads. Sequential consistency memory models often require the use of synchronization primitives, such as locks or barriers, to enforce a stricter ordering of memory operations across threads. These primitives can be used to ensure that memory operations are performed in a consistent order across all threads.

In this section, we describe the LTSs that will compose the synchronous product that describe the sequential consistent memory model. We start with the description of the thread semantics, after we will describe the memory semantics and the scheduler semantics. Then we show the synchronous relation that we will consider and in the end we show the bi-simulation between our model and the sequential consistency memory model described in Section 2.2.2.

Threads

We describe a thread with the tuple $\langle \theta, e\rangle$ where θ represents the unique identifier of the thread, while e denotes the expression to be evaluated, where the expression is the same as described in Section 2.2.1 We show the syntax format of the LTS that describes the threads: $\mathcal{T} ::= \langle \theta, e\rangle \mid (\mathcal{T} \mid \mathcal{T})$.

In Fig. 4.1 we show the rules that describes the thread execution. Rule Thread Step explains the evaluation of an action, different from the spawn action (sp), as indicated by the condition $\alpha \neq \mathsf{sp}$. Thread Spawn rule creates a new thread with a new identifier θ'.

$$\textsc{Alloc}\ \frac{}{(r \mapsto \circ) \xrightarrow{(\theta,\mathsf{ref}(r,v))} (r \mapsto v)} \qquad \textsc{Read}\ \frac{}{(r \mapsto v) \xrightarrow{(\theta,\mathsf{rd}(r,v))} (r \mapsto v)} \qquad \textsc{Write}\ \frac{}{(r \mapsto v_{old}) \xrightarrow{(\theta,\mathsf{wr}(r,v))} (r \mapsto v)}$$

$$\textsc{Memory Par}\ \frac{\mathcal{M} \xrightarrow{\ell_m} \mathcal{M}'}{\mathcal{M} \mid \mathcal{M}_1 \xrightarrow{\ell_m} \mathcal{M}' \mid \mathcal{M}_1}$$

Figure 4.2: Memory Semantics

$$\textsc{Acquire}\ \frac{}{(l \mapsto \circ) \xrightarrow{(\theta,\mathsf{acq}(l))} (l \mapsto \theta)} \qquad \textsc{Release}\ \frac{}{(l \mapsto \theta) \xrightarrow{(\theta,\mathsf{rel}(l))} (l \mapsto \circ)} \qquad \textsc{Scheduler Par}\ \frac{\mathcal{S} \xrightarrow{\ell_s} \mathcal{S}'}{\mathcal{S} \mid \mathcal{S}_1 \xrightarrow{\ell_s} \mathcal{S}' \mid \mathcal{S}_1}$$

Figure 4.3: Scheduler Semantics

Memory

Here, we present the LTS representing the memory. We assume a memory where all cells are initially set to a symbol $(r \mapsto \circ)$, indicating that the specific memory cell r has not yet been allocated with an action $(\mathsf{ref}(v))$. This assumption is reasonable since, in practice, we typically know beforehand which are all memory cells in the system. The syntax format for memory is $\mathcal{M} ::= (r \mapsto v) \mid (r \mapsto \circ) \mid (\mathcal{M} \mid \mathcal{M})$.

In Fig. 4.2, we provide rules for memory operations, including allocation, reading, and writing. The memory semantics is defined by the following rules: Alloc describes the allocation of memory for a new variable, Read governs the process of reading a value from memory, and Write specifies how a value is written to memory.

Scheduler:

Here, we present the model of the LTS representing the scheduler that manages the acquisition and release of locks. The syntax format for the scheduler is $\mathcal{S} ::= (l \mapsto \theta) \mid (l \mapsto \circ) \mid (\mathcal{S} \mid \mathcal{S})$, where $(l \mapsto \circ)$ indicates a lock that has not yet been acquired, a similar reasoning was discussed earlier for memory.

In Fig. 4.3 we present the scheduler semantics defined by rule Acquire, which describes the process of acquiring a lock, and by rule Release, which governs the release of a previously acquired lock.

Labels:

We consider the synchronous products $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ where $\mathcal{R}$ is composed of the following element:

$\langle(\theta, \mathsf{pure}), \phi, \phi\rangle$ correspond to the evaluation of a lambda expression applied to an argument;

$\langle(\theta, \mathsf{sp}(\theta')), \phi, \phi\rangle$ correspond to the creation of a new process;

$\langle(\theta, \mathsf{ref}(r, v)), (\theta, \mathsf{ref}(r, v)), \phi\rangle$ correspond to the allocation of a memory cell;

$\langle(\theta, \mathsf{rd}(r, v)), (\theta, \mathsf{rd}(r, v)), \phi\rangle$ correspond to the reading of a memory cell;

$\langle(\theta, \mathsf{wr}(r, v)), (\theta, \mathsf{wr}(r, v)), \phi\rangle$ correspond to the modification of a memory cell;

$\langle(\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l))\rangle$ correspond to the acquisition of a lock;

$\langle(\theta, \mathsf{rel}(l)), \phi, (\theta, \mathsf{rel}(l))\rangle$ correspond to the release of a previously acquired lock.

Bi-simulation:

We aim to compare our sequential consistent memory model with the one presented in [34].

Our goal is to establish a bi-simulation between the model described in 2.2.2 and the product of LTSs $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$.

To do this we need to define the function for the label and the relation between the state of the two models.

Definition 32. *The function $\varphi(\cdot)$ is defined over labels:*

$$\begin{gathered}
\varphi(((\theta, \alpha), \phi, \phi)) = (\theta, \alpha),\\
\varphi(((\theta, \mathsf{ref}(r, v)), (\theta, \mathsf{ref}(r, v)), \phi)) = (\theta, \mathsf{ref}(r, v)),\\
\varphi(((\theta, \mathsf{rd}(r, v)), (\theta, \mathsf{rd}(r, v)), \phi)) = (\theta, \mathsf{rd}(r, v)),\\
\varphi(((\theta, \mathsf{wr}(r, v)), (\theta, \mathsf{wr}(r, v)), \phi)) = (\theta, \mathsf{wr}(r, v)),\\
\varphi(((\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l)))) = (\theta, \mathsf{acq}(l)),\\
\varphi(((\theta, \mathsf{rel}(l)), \phi, (\theta, \mathsf{rel}(l)))) = (\theta, \mathsf{rel}(l))
\end{gathered}$$

where $\alpha = \mathsf{sp}(\theta'), \mathsf{pure}$.

Theorem 8. $P \mid (\sigma; L) \sim (T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S)$

$$\begin{array}{ccc}
P \mid (\sigma; L) & \underset{Rel^{-1}}{\overset{Rel}{\longleftrightarrow}} & T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S \\
{\scriptstyle\varphi(\alpha)}\downarrow & & \downarrow{\scriptstyle\alpha} \\
P' \mid (\sigma'; L') & \underset{Rel^{-1}}{\overset{Rel}{\longleftrightarrow}} & T' \otimes_{\mathcal{R}} M' \otimes_{\mathcal{R}} S'
\end{array}$$

Proof. Firstly, we need to define the relation (Rel) between the states of $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ and of $P \mid (\sigma; L)$, where

$$\begin{gathered}
Rel = \{(T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S; P \mid (\sigma; L)) \mid \forall\langle\theta, e\rangle \in T \text{ then } P(\theta) = e;\\
\forall(r \mapsto v) \in M \text{ then } \sigma(r) = v;\\
\forall(r \mapsto \circ) \in M \text{ then } r \notin \mathsf{dom}(\sigma);\\
\forall(l \mapsto \theta) \in S \text{ then } L(l) = \theta;\\
\forall(l \mapsto \circ) \in S \text{ then } l \notin \mathsf{dom}(L)\}
\end{gathered}$$

Now we show by case that for each transitions done by $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ then the $P \mid (\sigma; L)$ can do the same transitions. Now we proceed by cases for on the labels:

- We consider the case where $\varphi(\langle(\theta, \mathsf{pure}), \phi, \phi\rangle) = (\theta, \mathsf{pure})$, then in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{\mathsf{pure}} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{pure})} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{pure})} \langle\theta, e'\rangle \mid \mathcal{T}} \qquad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \qquad \langle((\theta, \mathsf{pure}), \phi, \phi)\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\mathsf{pure}),\phi,\phi)} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

  in $P \mid (\sigma; L)$ we have $P[\theta \mapsto e]$ (by Rel), and we consider the label (θ, pure). Then, we obtain:

$$\textsc{Program}\ \frac{\textsc{PrgmStep}\ \dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{pure}} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{pure})} P[\theta \mapsto e']}}{P \mid (\sigma; L) \xrightarrow{(\theta,\mathsf{pure})} P[\theta \mapsto e'] \mid (\sigma; L)}$$

  we can easily see that the final states are still related.

Now we consider the case starting from $P \mid (\sigma; L)$ and we consider then Rel^{-1} and φ^{-1}. We proceed by the case for each possible labels:

- We consider the case where $\varphi^{-1}((\theta, \mathsf{pure})) = \langle(\theta, \mathsf{pure}), \phi, \phi\rangle$, then in $P \mid (\sigma; L)$ we have that:

$$\textsc{Program}\ \frac{\textsc{PrgmStep}\ \dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{pure}} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{pure})} P[\theta \mapsto e']}}{P \mid (\sigma; L) \xrightarrow{(\theta,\mathsf{pure})} P[\theta \mapsto e'] \mid (\sigma; L)}$$

  in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ we have $(\theta \mapsto e) \in T$ (by Rel^{-1}), and we consider the label $\langle(\theta, \mathsf{pure}), \phi, \phi\rangle$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{\mathsf{pure}} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{pure})} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{pure})} \langle\theta, e'\rangle \mid \mathcal{T}} \qquad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \qquad \langle((\theta, \mathsf{pure}), \phi, \phi)\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\mathsf{pure}),\phi,\phi)} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

  we can easily see that the final states are still related.

The other cases are similar and they are shown in Appendix B.1. □

4.5 Instance: Write Buffering

The write-buffering model [24, 25] implements the Partial Store Order (PSO) memory model [23].

In this model, write operations are first buffered in a local buffer before being committed to memory. Each processor or thread has its own buffer, which allows write operations to be performed more quickly without immediately affecting the visible state of memory for other processors or threads. However, to maintain the PSO memory consistency model, certain ordering constraints must still be enforced when committing writes from the buffer to memory. Specifically, any buffered writes to a particular memory location must be committed to memory in the order in which they were issued by the writing processor or thread. To enforce this ordering, each buffer maintains an index that tracks the order in which write operations were issued. When a buffered write is committed to memory, the buffer checks the index to ensure that all previous writes to the same memory location have already been committed in the correct order. In addition, read operations must also respect the ordering constraints of the PSO model. When a read operation is issued, it checks the local buffer of the issuing processor or thread first to see if there are any buffered writes that have not yet been committed to memory. If a buffered write exists, the read operation will return the value from the buffered write instead of the value stored in memory. Overall, this write-buffering model provides a trade off between performance and memory consistency, allowing write operations to be performed quickly while still ensuring the ordering constraints of the PSO model are maintained. However, additional synchronization mechanisms, such as memory barriers or fences, may still be necessary to enforce consistent memory ordering across multiple processors or threads.

PSO model relaxes the ordering of writes with respect to other processors or threads, meaning that writes from different processors or threads can be reordered with respect to each other. In other words, if there are multiple threads or processors accessing the same memory location, there is no guarantee on the order in which writes from different processors or threads will be committed to memory. However, the PSO model does not relax the program order for writes with respect to other memory operations (read or write) from the same thread, meaning that writes cannot be delayed such that they actually commit after reads or writes to other locations by the same thread.

Thread:

We describe a thread with the tuple $\langle \theta, e \rangle$ where θ represents the unique identifier of the thread, while e denotes the expression to be evaluated, where the expression is the same as described in Section 2.2.1. We show the syntax format of the LTS that describes the threads: $\mathcal{T} ::= \langle \theta, e \rangle \mid (\mathcal{T} \mid \mathcal{T})$.

In Fig. 4.4 we give the rules that describes the thread execution. Rule Thread Step explains the evaluation of an action, Thread Spawn rule creates a new thread with a new identifier θ'. We consider also two rule for the reading: ReadM that correspond to a read from the memory, while ReadB is a read from the buffer.

$$\textsc{Thread Step}\ \frac{e \overset{\alpha}{\hookrightarrow} e'}{\langle \theta, e\rangle \xrightarrow{(\theta,\alpha)} \langle \theta, e'\rangle}\ \alpha \neq \mathsf{sp}(\theta'), \mathsf{rd}(r,v) \qquad \textsc{Thread ReadM}\ \frac{e \overset{\mathsf{rd}(r,v)}{\hookrightarrow} e'}{\langle \theta, e\rangle \xrightarrow{(\theta,\mathsf{rdM}(r,v))} \langle \theta, e'\rangle} \qquad \textsc{Thread ReadB}\ \frac{e \overset{\mathsf{rd}(r,v)}{\hookrightarrow} e'}{\langle \theta, e\rangle \xrightarrow{(\theta,\mathsf{rdB}(r,v))} \langle \theta, e'\rangle}$$

$$\textsc{Thread Spawn}\ \frac{e \overset{\mathsf{sp}(\theta')}{\hookrightarrow} e'; e'' \qquad \theta' \ \ \mathsf{fresh}}{\langle \theta, e\rangle \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \langle \theta, e'\rangle \mid \langle \theta', e''\rangle} \qquad \textsc{Thread Par}\ \frac{\mathcal{T} \xrightarrow{\ell_t} \mathcal{T}'}{\mathcal{T} \mid \mathcal{T}_1 \xrightarrow{\ell_t} \mathcal{T}' \mid \mathcal{T}_1}$$

Figure 4.4: Threads semantics

$$\textsc{Alloc}\ \frac{}{(r \mapsto \circ) \xrightarrow{(\theta,\mathsf{ref}(r,v))} (r \mapsto v)} \qquad \textsc{ReadM}\ \frac{}{(r \mapsto v) \xrightarrow{(\theta,\mathsf{rdM}(r,v))} (r \mapsto v)}$$

$$\textsc{ReadB}\ \frac{}{(r \mapsto v_1) \xrightarrow{(\theta,\mathsf{rdB}(r,v))} (r \mapsto v_1)} \qquad \textsc{Commit}\ \frac{}{(r \mapsto v_{old}) \xrightarrow{(\theta,\mathsf{mem}(r,v))} (r \mapsto v)} \qquad \textsc{Memory Par}\ \frac{\mathcal{M} \xrightarrow{\ell_m} \mathcal{M}'}{\mathcal{M} \mid \mathcal{M}_1 \xrightarrow{\ell_m} \mathcal{M}' \mid \mathcal{M}_1}$$

Figure 4.5: Memory Semantics

Memory:

Here, we present the LTS representing the memory. We assume a memory where all cells are initially set to a symbol $(r \mapsto \circ)$, indicating that the specific memory cell r has not yet been allocated with an action $(\mathsf{ref}(v))$. This assumption is reasonable since, in practice, we typically know beforehand which are all memory cells in the system. The syntax format for memory is $\mathcal{M} ::= (r \mapsto v) \mid (r \mapsto \circ) \mid (\mathcal{M} \mid \mathcal{M})$.

In Fig. 4.5, we provide rules for memory operations, including allocation, reading, and writing. The memory semantics is defined by the following rules: Alloc describes the allocation of memory for a new variable, ReadM governs the process of reading a value from the memory, while ReadB governs the process of reading a value from the buffer, and Commit specifies how a value is written to memory.

Scheduler:

Here, we present the model of the LTS representing the scheduler that manages the acquisition and release of locks. The syntax format for the scheduler is $\mathcal{S} ::= \mathcal{L} \mid \mathcal{B}$ where $\mathcal{L} ::= (l \mapsto \theta) \mid (l \mapsto \circ) \mid (\mathcal{L} \mid \mathcal{L})$, where $(l \mapsto \circ)$ indicates a lock that has not yet been acquired, a similar reasoning was discussed earlier for memory.

While $\mathcal{B}$ is data structure called a buffer that stores the pending writes for each thread. The buffers are maps of thread ID/position pairs into lists of values that are used as FIFO queues. The idea is that each thread's writes are stored in a buffer and are not visible to other threads until they are not committed in the global memory. The buffer can be seen as the memory view

$$\textsc{Acquire}\ \frac{}{(l \mapsto \circ) \xrightarrow{(\theta,\mathsf{acq}(l))} (l \mapsto \theta);} \qquad \textsc{Release}\ \frac{\forall r.\mathcal{B}(\theta,r) = \bullet}{(l \mapsto \theta) \mid \mathcal{B} \xrightarrow{(\theta,\mathsf{rel}(l))} (l \mapsto \circ) \mid \mathcal{B}} \qquad \textsc{Spawn}\ \frac{\forall r.\mathcal{B}(\theta,r) = \bullet}{\mathcal{B} \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \mathcal{B}}$$

$$\textsc{Write}\ \frac{\mathcal{B}(\theta,r) = q}{\mathcal{B} \xrightarrow{(\theta,\mathsf{wr}(r,v))} \mathcal{B}[(\theta,r) \mapsto q, v]} \qquad \textsc{Commit}\ \frac{\mathcal{B}(\theta,r) = v, q}{\mathcal{B} \xrightarrow{\mathsf{mem}(r,v)} \mathcal{B}[(\theta,r) \mapsto q]} \qquad \textsc{ReadM}\ \frac{\mathcal{B}(\theta,r) = \bullet}{\mathcal{B} \xrightarrow{(\theta,\mathsf{rdM}(r,v))} \mathcal{B}}$$

$$\textsc{ReadB}\ \frac{\mathcal{B}(\theta,r) = q, v}{\mathcal{B} \xrightarrow{(\theta,\mathsf{rdB}(r,v))} \mathcal{B}} \qquad \textsc{Scheduler Par}\ \frac{\mathcal{S} \xrightarrow{\ell_s} \mathcal{S}'}{\mathcal{S} \mid \mathcal{S}_1 \xrightarrow{\ell_s} \mathcal{S}' \mid \mathcal{S}_1}$$

Figure 4.6: Scheduler

of each thread. When a value is not present, the thread reads from the shared memory. In this way, a thread respects the order of its reads and writes, but reads and writes can be delayed for other threads. The buffer has the same shape as the buffer described in Section 2.2.3.

In Fig. 4.6 we define the scheduler semantics defined by rule Acquire, which describes the process of acquiring a lock, and by rule Release, which governs the release of a previously acquired lock, can be fired when all queues in $\mathcal{B}$ for θ are empty. The Spawn rule can be fired when all the queues in B for θ are empty, it governs the creation of a new thread. The Write rule add to the value written is added to the queue of (θ, r). The Commit rule states that the least recent value in the queue for (θ, r) is inserted into memory. The ReadM and ReadB rules handle reads from shared memory: if the queue for (θ, r) is empty, memory is read; otherwise, the scheduler reads the least recent value in the queue for (θ, r) and leaves B unchanged.

Labels:

We consider the synchronous products $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ where $\mathcal{R}$ is composed of the following element:

$\langle(\theta, \mathsf{pure}), \phi, \phi\rangle$ correspond to the evaluation of a lambda expression applied to an argument;

$\langle(\theta, \mathsf{sp}(\theta')), \phi, (\theta, \mathsf{sp}(\theta'))\rangle$ correspond to the creation of a new process;

$\langle(\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l))\rangle$ correspond to the acquisition of a lock;

$\langle(\theta, \mathsf{rel}(l)), \phi, (\theta, \mathsf{rel}(l))\rangle$ correspond to the release of a previously acquired lock.

$\langle(\theta, \mathsf{ref}(r, v)), (\theta, \mathsf{ref}(r, v)), \phi\rangle$ correspond to the allocation of a memory cell;

$\langle(\theta, \mathsf{rdM}(r, v)), (\theta, \mathsf{rdM}(r, v)), (\theta, \mathsf{rdM}(r, v))\rangle$ correspond to the reading an element from memory when the buffer is empty buffer.

$\langle(\theta, \mathsf{rdB}(r, v)), (\theta, \mathsf{rdB}(r, v)), (\theta, \mathsf{rdB}(r, v))\rangle$ correspond to the reading an element from the buffer.

$\langle(\theta, \mathsf{wr}(r, v)), \phi, (\theta, \mathsf{wr}(r, v))\rangle$ correspond to the write a value in the buffer;

$\langle \phi, (\theta, \mathsf{mem}(r, v)), (\theta, \mathsf{mem}(r, v)) \rangle$ correspond to the update of a memory cell using the first value of the buffer;

Bi-simulation:

We aim to compare our write buffer memory model with the one presented in [34].

Our goal is to establish a bi-simulation between the model described in 2.2.3 and the product of LTSs $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$.

To do this we need to define the function for the label and the relation between the state of the two models.

Definition 33. *The function $\varphi(\cdot)$ is defined over labels:*

$$\varphi(\langle(\theta, \mathsf{pure}), \phi, \phi\rangle) = (\theta, \mathsf{pure}),$$
$$\varphi(\langle(\theta, \mathsf{sp}(\theta')), \phi, (\theta, \mathsf{sp}(\theta'))\rangle) = (\theta, \mathsf{sp}(\theta')),$$
$$\varphi(\langle(\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l))\rangle) = (\theta, \mathsf{acq}(l)),$$
$$\varphi(\langle(\theta, \mathsf{rel}(l)), \phi, (\theta, \mathsf{rel}(l))\rangle) = (\theta, \mathsf{rel}(l)),$$
$$\varphi(\langle(\theta, \mathsf{ref}(r, v)), (\theta, \mathsf{ref}(r, v)), \phi\rangle) = (\theta, \mathsf{ref}(r, v)),$$
$$\varphi(\langle(\theta, \mathsf{rdM}(r, v)), (\theta, \mathsf{rdM}(r, v)), (\theta, \mathsf{rdM}(r, v))\rangle) = (\theta, \mathsf{rd}(r, v)),$$
$$\varphi(\langle(\theta, \mathsf{rdB}(r, v)), (\theta, \mathsf{rdB}(r, v)), (\theta, \mathsf{rdB}(r, v))\rangle) = (\theta, \mathsf{rd}(r, v)),$$
$$\varphi(\langle(\theta, \mathsf{wr}(r, v)), \phi, (\theta, \mathsf{wr}(r, v))\rangle) = (\theta, \mathsf{wr}(r, v)),$$
$$\varphi(\langle \phi, (\mathsf{mem}(r, v)), (\mathsf{mem}(r, v))\rangle) = \epsilon$$

Theorem 9. $P \mid (\sigma; L; B) \sim T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$

$$\begin{array}{ccc} P \mid (\sigma; L; B) & \xleftarrow[Rel^{-1}]{Rel} & T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}) \\ \Big\downarrow{\scriptstyle \varphi(\alpha)} & & \Big\downarrow{\scriptstyle \alpha} \\ P' \mid (\sigma'; L'; B') & \xleftarrow[Rel^{-1}]{Rel} & T' \otimes_{\mathcal{R}} M' \otimes_{\mathcal{R}} (\mathcal{L}' \mid \mathcal{B}') \end{array}$$

Proof. Firstly, we need to define the relation (Rel) between the states of $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ and of $P \mid (\sigma; L; B)$, where

$$\begin{aligned} Rel = \{&(T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}); P \mid (\sigma; L; B)) \mid \forall \langle \theta, e\rangle \in T \text{ then } P(\theta) = e; \\ &\forall (r \mapsto v) \in M \text{ then } \sigma(r) = v; \\ &\forall (r \mapsto \circ) \in M \text{ then } r \notin \mathsf{dom}(\sigma); \\ &\forall (l \mapsto \theta) \in \mathcal{L} \text{ then } L(l) = \theta; \\ &\forall (l \mapsto \circ) \in \mathcal{L} \text{ then } l \notin \mathsf{dom}(L); \\ &B = \mathcal{B}\} \end{aligned}$$

Now we show by case that for each transitions done by $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ then the $P \mid (\sigma; L; B)$ can do the same transitions. Now we proceed by cases on the labels:

- We consider the case where $\varphi(\langle(\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l))\rangle) = (\theta, \mathsf{acq}(l))$, then in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{acq}(l))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \dfrac{\dfrac{}{(l \mapsto \circ) \xrightarrow{(\theta,\mathsf{acq}(l))} (l \mapsto \theta)}}{(l \mapsto \circ) \mid \mathcal{L} \mid \mathcal{B} \xrightarrow{(\theta,\mathsf{acq}(l))} (l \mapsto \theta) \mid \mathcal{L} \mid \mathcal{B}} \quad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \quad \langle((\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l)))\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \circ) \mid \mathcal{L} \mid \mathcal{B} \xrightarrow{((\theta,\mathsf{acq}(l)),\phi,(\theta,\mathsf{acq}(l)))} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \theta) \mid \mathcal{L} \mid \mathcal{B}}$$

  in $P \mid (\sigma; L; B)$ we have $P[\theta \mapsto e]$ and $l \notin \mathsf{dom}(L)$ (by Rel), and we consider the label $(\theta, \mathsf{acq}(l))$. Then, we obtain:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{acq}(l)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{acq}(l))} P[\theta \mapsto e']} \qquad \dfrac{l \notin dom(L)}{(\sigma; L; B) \xrightarrow{(\theta,\mathsf{acq}(l))} (\sigma, L[l \mapsto \theta])}}{P \mid (\sigma; L; B) \xrightarrow{(\theta,\mathsf{acq}(l))} P[\theta \mapsto e'] \mid (\sigma; L[l \mapsto \theta]; B)}$$

  we can easily see that the final states are still related.

- We consider the case where $\varphi^{-1}((\theta, \mathsf{acq}(l))) = \langle(\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l))\rangle$, then in $P \mid (\sigma; L; B)$ we have that:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{acq}(l)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{acq}(l))} P[\theta \mapsto e']} \qquad \dfrac{l \notin dom(L)}{(\sigma; L; B) \xrightarrow{(\theta,\mathsf{acq}(l))} (\sigma, L[l \mapsto \theta])}}{P \mid (\sigma; L; B) \xrightarrow{(\theta,\mathsf{acq}(l))} P[\theta \mapsto e'] \mid (\sigma; L[l \mapsto \theta]; B)}$$

  in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have $\langle\theta, e\rangle \in \mathcal{T}$ and $(l \mapsto \circ) \in \mathcal{L}$ (by Rel^{-1}), and we consider the label $\langle(\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l))\rangle$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{acq}(l))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \dfrac{\dfrac{}{(l \mapsto \circ) \xrightarrow{(\theta,\mathsf{acq}(l))} (l \mapsto \theta)}}{(l \mapsto \circ) \mid \mathcal{L} \mid \mathcal{B} \xrightarrow{(\theta,\mathsf{acq}(l))} (l \mapsto \theta) \mid \mathcal{L} \mid \mathcal{B}} \quad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \quad \langle((\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l)))\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \circ) \mid \mathcal{L} \mid \mathcal{B} \xrightarrow{((\theta,\mathsf{acq}(l)),\phi,(\theta,\mathsf{acq}(l)))} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \theta) \mid \mathcal{L} \mid \mathcal{B}}$$

  we can easily see that the final states are still related.

The other cases are similar and they are shown in Appendix B.2. □

$$
\begin{array}{rrcl}
\textsf{Expressions} & e & ::= & V \mid e\ e \mid \textsf{ref}(e) \mid !e \mid e := e \mid \textsf{async}(e) \\
 & & & \mid \textsf{acq}(e) \mid \textsf{blockUntil}(e) \mid \textsf{unprotected}(e) \\
\textsf{Values} & V & ::= & c \mid x \mid \lambda x.e \\
\textsf{Constant} & c & ::= & \textsf{unit} \mid \textsf{true} \mid \textsf{false} \\
\textsf{Var} & x & &
\end{array}
$$

Figure 4.7: Language syntax

4.6 Instance: Automatic Mutual Exclusion

Automatic Mutual Exclusion (AME) memory model [16] is used in concurrent programming that ensures that only one thread can access a critical section of code at a time. Transactions, on the other hand, are a way to implement software transactional memory (STM) [17], a concurrency control mechanism that allows multiple threads to operate on shared memory without using locks. When AME is used in conjunction with transactions, it typically involves a lock-free or wait-free approach to managing access to critical sections of code. Each thread that wants to access a critical section first begins a transaction. Within the transaction, the thread attempts to obtain exclusive access to the critical section without using a lock or other blocking synchronization mechanism. If the thread is successful in obtaining access, it performs the necessary operations and then commits the transaction. If the thread is unsuccessful in obtaining access, it may retry the transaction or abort and try again later. In an AME system with transactions, the STM implementation typically handles the details of tracking the state of the transaction and ensuring that transactions do not conflict with each other. The use of transactions can provide a number of benefits, such as improved performance, reduced contention for locks, and better scalability in multi-threaded environments. However, care must be taken to ensure that transactions are properly implemented and that conflicts between transactions are appropriately handled to avoid errors and ensure correctness.

Automatic mutual exclusion (AME) is a technique used to ensure that only one thread or process can access a shared resource at a time. It is a concurrency control mechanism that is used to prevent race conditions and other synchronization problems.

In this section with our framework we want to create an instance that correspond to a memory model for implementing the AME technique.

4.6.1 Language

Now we consider the language presented in [16]. We show the syntax in Figure 4.7.

We also give several formal semantics in the following. The syntax introduces syntactic categories of values, constants, variables, and expressions. The values are constants, variables, and lambda abstractions. In addition to values and to expressions of the forms $\textsf{async}\ e$, $\textsf{blockUntil}\ e$, and $\textsf{unprotected}\ e$, the expressions include notations for function application ($e\ e$), allocation ($\textsf{ref}\ e$, which allocates a new reference location and returns it after initializing it to the value of e), dereferencing ($!e$, which returns the contents in the reference location that is the value of e), and assignment ($e := f$, which sets the reference location that is the value of e to the value of f). The syntax allows arbitrary nestings of the expression.

$$\textsc{Appl}\ \frac{}{(\lambda x.e)V \xhookrightarrow{\mathsf{pure}} e[V/x]} \qquad \textsc{Ref}\ \frac{}{\mathsf{ref}\ V \xhookrightarrow{\mathsf{ref}(r,v)} r} \qquad \textsc{DeRef}\ \frac{}{!r \xhookrightarrow{\mathsf{rd}(r,v)} v} \qquad \textsc{Set}\ \frac{}{r := v \xhookrightarrow{\mathsf{wr}(r,v)} \mathsf{unit}}$$

$$\textsc{Block}\ \frac{}{\mathsf{blockUntil\ true} \xhookrightarrow{\mathsf{blockUntil(true)}} \mathsf{unit}} \qquad \textsc{Async}\ \frac{}{\mathsf{async}\ e \xhookrightarrow{\mathsf{async}(e)} \mathsf{unit}}$$

Figure 4.8: Expression Semantics

Figure 4.8 shows a label semantics of the language, in the label we can see r that represent a memory location, and α that represent a generic step of the program, then $\alpha ::= \mathsf{pure} \mid \mathsf{ref}(r, v) \mid \mathsf{rd}(r, v) \mid \mathsf{wr}(r, v) \mid \mathsf{blockUntil}(\mathsf{true}) \mid \mathsf{async}(e)$.

4.6.2 Strong Semantics

In this section, we will introduce the strong semantics for AME described in [16], after which we will introduce our instance describing the components of the product of LTSs and the relation we will use.

In [16] they first introduce a strong semantics, this mean that when a thread starts a computation it has to conclude its computation before giving the control to another thread. The unprotected action can be done if the thread that have the control has as expression unit. Each evaluation step produces a new state. The next possible operation in an expression is found by decomposing the expression into an evaluation context and a sub-expression that describes this operation. As usual, a context is an expression with a hole [ ], and an evaluation context is a context of a particular kind. Given a context $\mathcal{C}$ and an expression e, we write $\mathcal{C}[e]$ for the result of placing e in the hole in $\mathcal{C}$. The context that we will use are:

Evaluation contexts:

$$\mathcal{E} ::= [\,] \mid \mathcal{E}\ e \mid V\ \mathcal{E} \mid \mathsf{ref}\ \mathcal{E} \mid\ !\mathcal{E} \mid \mathcal{E} := e \mid r := \mathcal{E} \mid \mathsf{blockUntil}\ \mathcal{E} \mid \mathsf{unprotected}\ \mathcal{E}$$

$$\mathcal{P} ::= [\,] \mid \mathcal{P}\ e \mid V\ \mathcal{P} \mid \mathsf{ref}\ \mathcal{P} \mid\ !\mathcal{P} \mid \mathcal{P} := e \mid r := \mathcal{P} \mid \mathsf{blockUntil}\ \mathcal{P}$$

$$\mathcal{U} ::= \mathsf{unprotected}\ \mathcal{E} \mid \mathcal{U}\ e \mid V\ \mathcal{U} \mid \mathsf{ref}\ \mathcal{U} \mid\ !\mathcal{U} \mid \mathcal{U} := e \mid r := \mathcal{U} \mid \mathsf{blockUntil}\ \mathcal{U}$$

About the context we said that:

- $\mathcal{P}$ evaluation contexts are for the execution of protected code: the position for evaluation is not under unprotected.
- $\mathcal{U}$ evaluation contexts are for the execution of unprotected code: the position for evaluation is under unprotected.

For more details we refer to [16].

In [16] they use a system transition of the form:

$$\langle \sigma, E, e \rangle \mapsto \langle \sigma', E', e' \rangle$$

where σ is the memory, E represents the active threads (expression) and e is the protected expression that the system evaluates.

The semantics presented in [16] is a reduction semantics without labels. Since we want to prove the bi-similarity between our system and the strong semantics for AME described in [16], we consider a semantics where the expression are updated and have the form of $\langle \theta, e \rangle$ instead of only e and where we decorate the semantics with the labels. The semantics it shown in Appendix B.3.1 in Fig. B.2 and Fig. B.3.

For this reason the update system transition that we consider have the form:

$$\langle \sigma, E, \langle \theta, e \rangle \rangle \overset{\ell}{\mapsto} \langle \sigma', E', \langle \theta, e' \rangle \rangle$$

where σ is the memory, E represents the active threads (expression) and e is the protected expression that the system evaluates and θ is a unique identifier for the expression and ℓ is the label of the transition.

For a deeper discussion we refer to [16].

Thread Semantics:

In this section we present the thread semantics, in our semantics we indicate with T the set of the threads in the system. We indicate with t a single thread. A thread is composed by an id (θ) that identifies the thread and by an expression. We denote the id and the expression of the thread t respectively with $t.\mathsf{id}$ and $t.\mathsf{exp}$. With $t[id \leftarrowtail \theta, exp \leftarrowtail e]$ we define the update/creation of the thread t where the id became θ and the expression became e.

We show the thread semantics in Fig. 4.9.

Scheduler Semantics:

The scheduler is composed by S, this element is the id of the thread that can perform a protected operation. S could be equal also to $\bullet$ in this case represent the possibility to a thread to perform a unprotected action.

We show the scheduler semantics in Fig. 4.10.

Memory Semantics:

The memory is composed by a map, σ, that associates a memory location with a value. The possible operations are: $\sigma[r \mapsto v]$ that create/update the location r and map it with the value v and $\sigma(r)$ which returns the value associated with r in σ. We indicate with M the LTS that represent the memory.

We show the memory semantics in Fig. 4.11.

Labels:

In this case we have two sets of actions:

- the operation that involve all the system (between threads, memory and scheduler) the operation $\mathsf{ref}, \mathsf{rd}$ and wr

$$\textsc{Activate1}\quad \frac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = e}{T \mid t \xrightarrow{(\theta,\text{no unit},\mathsf{act})} T \mid t}\ e \neq \mathsf{unit} \qquad \textsc{Activate2x}\quad \frac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathsf{unit}}{T \mid t \xrightarrow{(\theta,\mathsf{unit},\mathsf{act})} T}$$

$$\textsc{Step P1}\quad \frac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[e] \qquad e \overset{\alpha}{\hookrightarrow} e'}{T \mid t \xrightarrow{(\theta,\text{no unit},\alpha)} T \mid t[exp \leftarrow\!\shortmid \mathcal{P}[e']]}\ \alpha \neq \mathsf{async}(e'') \wedge \mathcal{P}[e'] \neq \mathsf{unit}$$

$$\textsc{Step P2}\quad \frac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[e] \qquad e \overset{\alpha}{\hookrightarrow} e'}{T \mid t \xrightarrow{(\theta,\mathsf{unit},\alpha)} T}\ \alpha \neq \mathsf{async}(e'') \wedge \mathcal{P}[e'] = \mathsf{unit}$$

$$\textsc{Step U}\quad \frac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{U}[e] \qquad e \overset{\alpha}{\hookrightarrow} e'}{T \mid t \xrightarrow{(\theta,\mathsf{unp}(\alpha))} T \mid t[exp \leftarrow\!\shortmid \mathcal{U}[e']]}\ \alpha \neq \mathsf{async}(e'')$$

$$\textsc{Spawn P1}\quad \frac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[\mathsf{async}\ e'] \qquad t', \theta'\ fresh}{T \mid t \xrightarrow{(\theta,\text{no unit},\mathsf{async}(e'))} T \mid t[exp \leftarrow\!\shortmid \mathcal{P}[\mathsf{unit}]] \mid t'[id \leftarrow\!\shortmid \theta', exp \leftarrow\!\shortmid e']}\ \mathcal{P}[\mathsf{unit}] \neq \mathsf{unit}$$

$$\textsc{Spawn P2}\quad \frac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[\mathsf{async}\ e'] \qquad t', \theta'\ fresh}{T \mid t \xrightarrow{(\theta,\mathsf{unit},\mathsf{async}(e'))} T \mid t'[id \leftarrow\!\shortmid \theta', exp \leftarrow\!\shortmid e']}\ \mathcal{P}[\mathsf{unit}] = \mathsf{unit}$$

$$\textsc{Spawn U}\quad \frac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{U}[\mathsf{async}\ e'] \qquad t', \theta'\ fresh}{T \mid t \xrightarrow{(\theta,\mathsf{unp}(\mathsf{async}(e')))} T \mid t[exp \leftarrow\!\shortmid \mathcal{U}[\mathsf{unit}]] \mid t'[id \leftarrow\!\shortmid \theta', exp \leftarrow\!\shortmid e']}$$

$$\textsc{Close}\quad \frac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{E}[\mathsf{unprotected}\ V]}{T \mid t \xrightarrow{(\theta,\mathsf{unp}(V))} T \mid t[exp \leftarrow\!\shortmid \mathcal{E}[V]]} \qquad \textsc{Unprotected}\quad \frac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[\mathsf{unprotected}\ e]}{T \mid t \xrightarrow{(\theta,\mathsf{unprtect})} T \mid t}$$

Remark:
$\alpha ::= \mathsf{pure} \mid \mathsf{ref}(r, v) \mid \mathsf{rd}(r, v) \mid \mathsf{wr}(r, v) \mid \mathsf{blockUntil}(\mathsf{true}) \mid \mathsf{async}(e) \qquad \beta ::= \mathsf{unp}(\alpha) \mid \mathsf{unp}(V)$

Figure 4.9: Strong Threads Semantics

- the operation between scheduler and the program: $\mathsf{pure}, \mathsf{blockUntil}(\mathsf{true}), \mathsf{async}(e')$ and act
- in this case there are not the silent operation of the scheduler, of the memory and of the thread.

We consider the synchronous products $T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma$ where $\mathcal{R}$ is composed of the following element:

$\langle(\theta, \mathsf{no\ unit}, \mathsf{act}), \phi, (\theta, \mathsf{no\ unit}, \mathsf{act})\rangle$ indicate the activation of a thread. This operation involves the scheduler and the thread model;

$$\textsc{Activate1}\ \frac{S = \bullet}{S \xrightarrow{(\theta,\text{no unit},\text{act})} \theta} \qquad \textsc{Activate2}\ \frac{S = \bullet}{S \xrightarrow{(\theta,\text{unit},\text{act})} S} \qquad \textsc{Step P1}\ \frac{S = \theta}{S \xrightarrow{(\theta,\text{no unit},\alpha)} S} \qquad \textsc{Step P2}\ \frac{S = \theta}{S \xrightarrow{(\theta,\text{unit},\alpha)} \bullet}$$

$$\textsc{Step U/Close}\ \frac{S = \bullet}{S \xrightarrow{(\theta,\beta)} S} \qquad \textsc{Unprotected}\ \frac{S = \theta}{S \xrightarrow{(\theta,\text{unprotect})} \bullet}$$

Remark:
$\alpha ::= \mathsf{pure} \mid \mathsf{ref}(r,v) \mid \mathsf{rd}(r,v) \mid \mathsf{wr}(r,v) \mid \mathsf{blockUntil}(\mathsf{true}) \mid \mathsf{async}(e) \qquad \beta ::= \mathsf{unp}(\alpha) \mid \mathsf{unp}(V)$

Figure 4.10: Strong Scheduler Semantics

$$\textsc{Ref}\ \frac{r \in (RefLoc - \mathsf{dom}(\sigma))}{\sigma \xrightarrow{(\theta,\_,\mathsf{ref}(r,v))} \sigma[r \mapsto v]} \qquad \textsc{DeRef (Read)}\ \frac{\sigma(r) = v}{\sigma \xrightarrow{(\theta,\_,\mathsf{rd}(r,v))} \sigma} \qquad \textsc{Set (Write)}\ \frac{}{\sigma \xrightarrow{(\theta,\_,\mathsf{wr}(r,v))} \sigma[r \mapsto v]}$$

$$\textsc{URef}\ \frac{r \in (RefLoc - \mathsf{dom}(\sigma))}{\sigma \xrightarrow{(\theta,\mathsf{unp}(\mathsf{ref}(r,v)))} \sigma[r \mapsto v]} \qquad \textsc{UDeRef (Read)}\ \frac{\sigma(r) = v}{\sigma \xrightarrow{(\theta,\mathsf{unp}(\mathsf{rd}(r,v)))} \sigma} \qquad \textsc{USet (Write)}\ \frac{}{\sigma \xrightarrow{(\theta,\mathsf{unp}(\mathsf{wr}(r,v)))} \sigma[r \mapsto v]}$$

Remark:
With _ we indicate either "unit" or "no unit"

Figure 4.11: Strong Memory Semantics

$\langle(\theta, \mathsf{unit}, \mathsf{act}), \phi, (\theta, \mathsf{unit}, \mathsf{act})\rangle$ indicate the activation of a thread with expression equal to unit. This operation involves the scheduler and the thread model;

$\langle(\theta, \mathsf{no\ unit}, \mathsf{pure}), \phi, (\theta, \mathsf{no\ unit}, \mathsf{pure})\rangle$ is used to evaluate a lambda expression (with a protected action). This operation involves the scheduler and the thread model;

$\langle(\theta, \mathsf{unit}, \mathsf{pure}), \phi, (\theta, \mathsf{unit}, \mathsf{pure})\rangle$ is used to evaluate a lambda expression (with a protected action and with the evaluation of the next expression equal to unit).This operation involves the scheduler and the thread model;

$\langle(\theta, \mathsf{no\ unit}, \mathsf{ref}(r, v)), (\theta, \mathsf{no\ unit}, \mathsf{ref}(r, v)), (\theta, \mathsf{no\ unit}, \mathsf{ref}(r, v))\rangle$ indicate the transition for creating a new reference in the memory (with a protected action). This operation involves all the system (scheduler, thread and memory model);

$\langle(\theta, \mathsf{unit}, \mathsf{ref}(r, v)), (\theta, \mathsf{unit}, \mathsf{ref}(r, v)), (\theta, \mathsf{unit}, \mathsf{ref}(r, v))\rangle$ indicate the transition for creating a new reference in the memory (with a protected action and with the evaluation of the next expression equal to unit). This operation involves all the system (scheduler, thread and memory model);

$\langle(\theta, \mathsf{no\ unit}, \mathsf{rd}(r, v)), (\theta, \mathsf{no\ unit}, \mathsf{rd}(r, v)), (\theta, \mathsf{no\ unit}, \mathsf{rd}(r, v))\rangle$ is used to reading an element in the memory (with a protected action). This operation involves all the system (scheduler, thread and memory model);

$\langle(\theta, \mathsf{unit}, \mathsf{rd}(r, v)), (\theta, \mathsf{unit}, \mathsf{rd}(r, v)), (\theta, \mathsf{unit}, \mathsf{rd}(r, v))\rangle$ is used to reading an element in the memory (with a protected action and with the evaluation of the next expression equal to unit). This operation involves all the system (scheduler, thread and memory model);

$\langle(\theta, \mathsf{no\ unit}, \mathsf{wr}(r, v)), (\theta, \mathsf{no\ unit}, \mathsf{wr}(r, v)), (\theta, \mathsf{no\ unit}, \mathsf{wr}(r, v))\rangle$ indicate the modification of a reference in the memory (with a protected action). This operation involves all the system (scheduler, thread and memory model);

$\langle(\theta, \mathsf{unit}, \mathsf{wr}(r, v)), (\theta, \mathsf{unit}, \mathsf{wr}(r, v)), (\theta, \mathsf{unit}, \mathsf{wr}(r, v))\rangle$ indicate the modification of a reference in the memory (with a protected action and with the evaluation of the next expression equal to unit). This operation involves all the system (scheduler, thread and memory model);

$\langle(\theta, \mathsf{no\ unit}, \mathsf{blockUntil}(\mathsf{true})), \phi, (\theta, \mathsf{no\ unit}, \mathsf{blockUntil}(\mathsf{true}))\rangle$ indicate the success of an asynchronous execution (with a protected action). This operation involves the scheduler and the thread model;

$\langle(\theta, \mathsf{unit}, \mathsf{blockUntil}(\mathsf{true})), \phi, (\theta, \mathsf{unit}, \mathsf{blockUntil}(\mathsf{true}))\rangle$ indicate the success of an asynchronous execution (with a protected action and with the evaluation of the next expression equal to unit).This operation involves the scheduler and the thread model;

$\langle(\theta, \mathsf{no\ unit}, \mathsf{async}(e')), \phi, (\theta, \mathsf{no\ unit}, \mathsf{async}(e'))\rangle$ indicate the creation of a new thread (with a protected action). This operation involves the scheduler and the thread model;

$\langle(\theta, \mathsf{unit}, \mathsf{async}(e')), \phi, (\theta, \mathsf{unit}, \mathsf{async}(e'))\rangle$ indicate the creation of a new thread (with a protected action and with the evaluation of the next expression equal to unit). This operation involves the scheduler and the thread model;

$\langle(\theta, \mathsf{unp}(\mathsf{pure})), \phi, (\theta, \mathsf{unp}(\mathsf{pure}))\rangle$ is used to evaluate a lambda expression (with an unprotected action). This operation involves the scheduler and the thread model;

$\langle(\theta, \mathsf{unp}(\mathsf{ref}(r, v))), (\theta, \mathsf{unp}(\mathsf{ref}(r, v))), (\theta, \mathsf{unp}(\mathsf{ref}(r, v)))\rangle$ indicate the transition for creating a new reference in the memory (with an unprotected action). This operation involves all the system (scheduler, thread and memory model);

$\langle(\theta, \mathsf{unp}(\mathsf{rd}(r, v))), (\theta, \mathsf{unp}(\mathsf{rd}(r, v))), (\theta, \mathsf{unp}(\mathsf{rd}(r, v)))\rangle$ is used to reading an element in the memory (with an unprotected action). This operation involves all the system (scheduler, thread and memory model);

$\langle(\theta, \mathsf{unp}(\mathsf{wr}(r, v))), (\theta, \mathsf{unp}(\mathsf{wr}(r, v))), (\theta, \mathsf{unp}(\mathsf{wr}(r, v)))\rangle$ indicate the modification of a reference in the memory (with an unprotected action). This operation involves all the system (scheduler, thread and memory model);

$\langle(\theta, \mathsf{unp}(\mathsf{blockUntil}(\mathsf{true}))), \phi, (\theta, \mathsf{unp}(\mathsf{blockUntil}(\mathsf{true})))\rangle$ indicate the success of an asynchronous execution (with an unprotected action). This operation involves the scheduler and the thread model;

$\langle(\theta, \mathsf{unp}(\mathsf{async}(e'))), \phi, (\theta, \mathsf{unp}(\mathsf{async}(e')))\rangle$ indicate the creation of a new thread (with an unprotected action). This operation involves the scheduler and the thread model;

$\langle(\theta, \mathsf{unp}(V)), \phi, (\theta, \mathsf{unp}(V))\rangle$ indicate the finish of an unprotected evaluation. This operation involves the scheduler and the thread model;

$\langle(\theta, \mathsf{unprotect}), \phi, (\theta, \mathsf{unprotect})\rangle$ indicate the start of unprotected evaluation. This operation involves the scheduler and the thread model;

Bi-similarity:

Now we have introduce the element to define our model we aim to establish a bi-simulation (see Def.31) between the strong semantics for AME described in [16] and the product of LTSs $T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma$.

To do this we need to define the function for the label and the relation between the state of the two models.

Definition 34. *The function $\varphi(\cdot)$ is defined over labels:*

$$\varphi(\alpha, \phi, \alpha) = \alpha,$$
$$\varphi(\beta, \beta, \beta) = \beta,$$

where $\alpha = (\theta, \_, \mathsf{act}), (\theta, \_, \mathsf{pure}), (\theta, \_, \mathsf{blockUntil}(\mathsf{true})), (\theta, \_, \mathsf{async}(e)), (\theta, \mathsf{unp}(\mathsf{pure})), (\theta, \mathsf{unp}(\mathsf{blockUntil}(\mathsf{true}))), (\theta, \mathsf{unp}(\mathsf{async}(e))), (\theta, \mathsf{unp}(V)), (\theta, \mathsf{unprotect})$*;*
and where $\beta = (\theta, \_, \mathsf{ref}(r, v)), (\theta, \_, \mathsf{rd}(r, v)), (\theta, \_, \mathsf{wr}(r, v)), (\theta, \mathsf{unp}(\mathsf{ref}(r, v))), (\theta, \mathsf{unp}(\mathsf{rd}(r, v))), (\theta, \mathsf{unp}(\mathsf{wr}(r, v)))$ *where with* _ *we indicate* no unit *or* unit.

Theorem 10. $\langle \sigma_A, E, e\rangle \sim (T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma)$

$$\begin{array}{ccc} \langle \sigma_A, E, e\rangle & \underset{Rel^{-1}}{\overset{Rel}{\longleftrightarrow}} & (T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma) \\ {\scriptstyle\varphi(\alpha)}\downarrow & & \downarrow{\scriptstyle\alpha} \\ \langle \sigma'_A, E', e'\rangle & \underset{Rel^{-1}}{\overset{Rel}{\longleftrightarrow}} & (T' \otimes_{\mathcal{R}} S' \otimes_{\mathcal{R}} \sigma') \end{array}$$

Proof. We start by defining *Rel* and then show that it is a strong similarity relation, after which we will show that Rel^{-1} is also a strong similarity relation. Given $\{\langle \sigma_A, E, (\theta, e)\rangle, \langle T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\}$ then this tuple is in *Rel* if:

- $\sigma = \sigma_A$
- $\forall e_i \in E$ exist θ_i such that $t_i \in T$ and $t_i.\mathsf{id} = \theta_i \wedge t_i.\mathsf{exp} = e_i$
- if $e = \mathsf{unit}$ then $S = \bullet$ otherwise (if $e \neq \mathsf{unit}$) exist $t \in T$ where $t.\mathsf{id} = \theta \wedge t.\mathsf{exp} = e$ and $S = \theta$

After the definition of *Rel* we start to give for each label the proof that there is a strong similarity between Abadi system and our system:

- if in Abadi.system we have this transition:

  $$\textsc{Trans Activate}$$
  $$\langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle \xmapsto{(\theta,\text{no unit,act})} \langle \sigma_A, E.E', (\theta, e)\rangle$$

  in our system we have:

  $$\frac{\dfrac{\textsc{Activate1}\quad t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = e}{T \mid t \xrightarrow{(\theta,\text{no unit,act})} T \mid t}\ e \neq \mathsf{unit} \qquad \dfrac{\textsc{Activate1}\quad S = \bullet}{S \xrightarrow{(\theta,\text{no unit,act})} \theta}}{T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\text{no unit,act}),\phi,(\theta,\text{no unit,act})\rangle} T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma}$$

  We know that $\{\langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle, \langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$. Then after the transitions we have that $\{\langle \sigma_A, E.E', (\theta, e)\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$. This couple belongs to the relation *Rel* because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in $E.E'$ have a corresponding thread in T (none of the rules applied to the two systems changed either $E.E'$ or T) and the thread t corresponds to e (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$).

To proof the bi-simulation we can prove that given *Rel* as a strong simulation relation then if also Rel^{-1} is a strong simulation relation then *Rel* is a bi-simulation relation. Now we show the relationship Rel^{-1} derived directly from *Rel*. Given $\{\langle T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, e\rangle\}$ then this tuple is in Rel^{-1} if:

- $\sigma_A = \sigma$
- $\forall t_i \in T$ such that $t_i.\mathsf{id} = \theta_i$ and $\theta_i \neq S$ then exist $e_i = t_i.\mathsf{exp}$ and $e_i \in E$

- if $S = \bullet$ then $e = \mathsf{unit}$, otherwise (if $S \neq \bullet$) then exist $t \in T$ such that $t.\mathsf{id} = S$ and we have that $e = t.\mathsf{exp}$

Now we start to analyses all the possible case:

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{no\ unit},\mathsf{act}),\phi,(\theta,\mathsf{no\ unit},\mathsf{act})\rangle} T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma$$

  in Abadi.system we have:

$$\textsc{Trans Activate}$$
$$\langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{no\ unit},\mathsf{act})} \langle \sigma_A, E.E', (\theta, e)\rangle$$

  We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$. Then after the transitions we have that $\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \{\langle \sigma_A, E.E', (\theta, e)\rangle\} \in Rel^{-1}$. This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a corresponding thread in T (none of the rules applied to the two systems changed either E and E' or T) and e correspond to the thread t (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$).

The other cases are similar and they are shown in Appendix B.3.1. □

4.6.3 Weak Semantics

In this section, we will introduce the weak semantics for AME described in [16], after which we will introduce our instance describing the components of the product of LTSs and the relation we will use.

In [16] they introduce a weak semantics, this mean that this semantics allows unprotected computations to proceed even when the active expression is not unit.

We consider the same context and the same updated transition used for the strong semantics. For a deeper discussion we refer to [16], and we show the updated rule in Appendix B.3.2 in Fig. B.5 and Fig. B.6.

Thread Semantics:

The thread semantics is in Fig. 4.9.

Memory Semantics:

The memory semantics is in Fig. 4.11.

Scheduler Semantics:

In the new scheduler semantics we can see that change only the red part. This semantics is presented in Fig. 4.12.

$$\textsc{Activate1}\ \frac{S = \bullet}{S \xrightarrow{(\theta,\text{no unit},\text{act})} \theta} \qquad \textsc{Activate2}\ \frac{S = \bullet}{S \xrightarrow{(\theta,\text{unit},\text{act})} S} \qquad \textsc{Step P1}\ \frac{S = \theta}{S \xrightarrow{(\theta,\text{no unit},\alpha)} S} \qquad \textsc{Step P2}\ \frac{S = \theta}{S \xrightarrow{(\theta,\text{unit},\alpha)} \bullet}$$

$$\textsc{Step U/Close}\ \frac{S \neq \theta}{S \xrightarrow{(\theta,\beta)} S} \qquad \textsc{Unprotected}\ \frac{S = \theta}{S \xrightarrow{(\theta,\text{unprotect})} \bullet}$$

$$\alpha ::= \mathsf{pure} \mid \mathsf{ref}(r,v) \mid \mathsf{rd}(r,v) \mid \mathsf{wr}(r,v) \mid \mathsf{blockUntil}(\mathsf{true}) \mid \mathsf{async}(e) \qquad \beta ::= \mathsf{unp}(\alpha) \mid \mathsf{unp}(V)$$

Figure 4.12: Weak Scheduler Semantics

The difference between the strong semantics presented above and this semantics is that in this semantics protected actions can be interspersed with unprotected actions. Thus by analysing the rules we can see that unprotected rules can always be executed thanks to the new rule.

Labels:

We consider the synchronous products $T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma$ where $\mathcal{R}$ is composed by the same element of the previews instance described in 4.6.2.

Bi-similarity:

Now we have introduce the element to define our model we aim to establish a bi-simulation (see Def.31) between the weak semantics for AME described in [16] and the product of LTSs $T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma$.

To do this we need to define the function for the label and the relation between the states of the two models that is the same of the presented before (Def. 34).

Definition 34. *The function $\varphi(\cdot)$ is defined over labels:*

$$\varphi(\alpha, \phi, \alpha) = \alpha,$$
$$\varphi(\beta, \beta, \beta) = \beta,$$

where $\alpha = (\theta, \_, \mathsf{act}), (\theta, \_, \mathsf{pure}), (\theta, \_, \mathsf{blockUntil}(\mathsf{true})), (\theta, \_, \mathsf{async}(e)), (\theta, \mathsf{unp}(\mathsf{pure})), (\theta, \mathsf{unp}(\mathsf{blockUntil}(\mathsf{true}))), (\theta, \mathsf{unp}(\mathsf{async}(e))), (\theta, \mathsf{unp}(V)), (\theta, \mathsf{unprotect})$*;*
and where $\beta = (\theta, \_, \mathsf{ref}(r,v)), (\theta, \_, \mathsf{rd}(r,v)), (\theta, \_, \mathsf{wr}(r,v)), (\theta, \mathsf{unp}(\mathsf{ref}(r,v))), (\theta, \mathsf{unp}(\mathsf{rd}(r,v))), (\theta, \mathsf{unp}(\mathsf{wr}(r,v)))$ *where with* $\_$ *we indicate* no unit *or* unit.

Theorem 11. $\langle \sigma_A, E, e \rangle \sim (T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma)$

$$\begin{array}{ccc} \langle \sigma_A, E, e \rangle & \xleftarrow[Rel^{-1}]{Rel} & (T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma) \\ \Big\downarrow{\scriptstyle \varphi(\alpha)} & & \Big\downarrow{\scriptstyle \alpha} \\ \langle \sigma'_A, E', e' \rangle & \xleftarrow[Rel^{-1}]{Rel} & (T' \otimes_{\mathcal{R}} S' \otimes_{\mathcal{R}} \sigma') \end{array}$$

Proof. We start by defining Rel and then show that it is a strong similarity relation, after which we will show that Rel^{-1} is also a strong similarity relation. Given $\{\langle \sigma_A, E, e\rangle, \langle T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\}$ then this tuple is in Rel if:

- $\sigma = \sigma_A$
- $\forall e_i \in E$ exist θ_i such that $t_i \in T$ and $t_i.\mathsf{id} = \theta_i \wedge t_i.\mathsf{exp} = e_i$
- if $e = \mathsf{unit}$ then $S = \bullet$ otherwise (if $e \neq \mathsf{unit}$) exist $t \in T$ where $t.\mathsf{id} = \theta \wedge t.\mathsf{exp} = e$ and $S = \theta$

After the definition of Rel we start to give for each label the proof that there is a strong similarity between Abadi system and our system:

- if in Abadi.system we have this transition:

  $$\textsc{Trans Activate}\quad \langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle \xmapsto{(\theta,\text{no unit,act})} \langle \sigma_A, E.E', (\theta, e)\rangle$$

  in our system we have:

  $$\frac{\dfrac{\textsc{Activate1}\quad t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = e}{T \mid t \xrightarrow{(\theta,\text{no unit,act})} T \mid t}\ e \neq \mathsf{unit} \qquad \dfrac{\textsc{Activate1}\quad S = \bullet}{S \xrightarrow{(\theta,\text{no unit,act})} \theta}}{T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\text{no unit,act}),\phi,(\theta,\text{no unit,act})\rangle} T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma}$$

  We know that $\{\langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle, \langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$. Then after the transitions we have that $\{\langle \sigma_A, E.E', (\theta, e)\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$. This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in $E.E'$ have a corresponding thread in T (none of the rules applied to the two systems changed either $E.E'$ or T) and the thread t corresponds to e (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$).

To proof the bi-simulation we can prove that given Rel as a strong simulation relation then if also Rel^{-1} is a strong simulation relation then Rel is a bi-simulation relation. Now we show the relationship Rel^{-1} derived directly from Rel. Given $\{\langle T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, e\rangle\}$ then this tuple is in Rel^{-1} if:

- $\sigma_A = \sigma$
- $\forall t_i \in T$ such that $t_i.\mathsf{id} = \theta_i$ and $\theta_i \neq S$ then exist $e_i = t_i.\mathsf{exp}$ and $e_i \in E$
- if $S = \bullet$ then $e = \mathsf{unit}$, otherwise (if $S \neq \bullet$) then exist $t \in T$ such that $t.\mathsf{id} = S$ and we have that $e = t.\mathsf{exp}$

After the definition of Rel^{-1} we start to give for each label the proof that there is a strong similarity between Abadi system and our system:

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\text{no unit,act}),\phi,(\theta,\text{no unit,act})\rangle} T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \qquad \text{if } e \neq \mathsf{unit}$$

  in Abadi.system we have:

$$\textsc{Trans Activate}$$
$$\langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle \xmapsto{(\theta,\text{no unit,act})} \langle \sigma_A, E.E', (\theta, e)\rangle$$

  We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$. Then after the transitions we have that $\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \{\langle \sigma_A, E.E', (\theta, e)\rangle\} \in Rel^{-1}$. This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a corresponding thread in T (none of the rules applied to the two systems changed either E and E' or T) and e correspond to the thread t (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$).

The other cases are similar and they are shown in Appendix B.3.2. □

4.6.4 Weak Semantics with Optimistic Rollback

In this section, we will introduce the weak semantics with optimistic rollback for AME described in [16], after which we will introduce our instance describing the components of the product of LTSs and the relation we will use.

In [16] they use a system transition of the form:

$$\langle \sigma, E, O, l\rangle \longmapsto \langle \sigma', E', O', l'\rangle$$

where σ is the memory, E represents the active threads (expression) and O is a set of tuples representing expressions that can be evaluated, these tuples have the form $\langle e, f, a, P\rangle$ where e is the expression f is the origin, a is the location of the accesses that e has performed, and P is a list of pending threads to be forked upon commit. l is a list of memory locations and their values, to be used as a log in undos. For a deeper discussion we refer to [16].

We use an updated form where we decorate the expression with a unique identifier and the transitions with a label. Then the expressions in E, in O and in P will have the form $\langle \theta, e\rangle$. We show the updated semantics in the Appendix B.3.3 in Fig. B.8.

Thread Semantics:

We consider two types of thread: the normal threads and the pending threads which are related to an id.

We indicate with AT the set off all the thread and it is composed by T and P, we have $AT = T \mid P$. Now we describe the two component of AT. We denote the set of normal threads as T, an element of this set is indicate as t a thread is composed by three elements: the id, the expression and the expression that originated the thread (indicate whit init). With $t[\mathsf{id} \leftarrow \theta, \mathsf{exp} \leftarrow e, \mathsf{init} \leftarrow e']$ we update the identifier of the thread t with θ, the expression

$$\textsc{Thread Activate}\quad \frac{t.\mathsf{id} = \theta}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{act})} T \mid t[\mathsf{init} \hookleftarrow t.\mathsf{exp}] \mid P}$$

$$\textsc{Thread Step}\quad \frac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[e] \qquad e \overset{\alpha}{\hookrightarrow} e'}{T \mid t \mid P \xrightarrow{(\theta,\alpha)} T \mid t[exp \hookleftarrow \mathcal{P}[e']] \mid P}\ \alpha \neq \mathsf{async}(e'')$$

$$\textsc{Thread Spawn}\quad \frac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[\mathsf{async}\ e'] \qquad t', \theta'\ fresh}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{async}(e'))} T \mid t[exp \hookleftarrow \mathcal{P}[\mathsf{unit}]] \mid P \mid \lfloor t'[id \hookleftarrow \theta', exp \hookleftarrow e'] \rfloor_\theta}$$

$$\textsc{Thread Step Unprotect}\quad \frac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{U}[e] \qquad e \overset{\alpha}{\hookrightarrow} e'}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{unp}(\alpha))} T \mid t[exp \hookleftarrow \mathcal{U}[e']] \mid P}\ \alpha \neq \mathsf{async}(e'')$$

$$\textsc{Thread Spawn Unprotect}\quad \frac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{U}[\mathsf{async}\ e'] \qquad t', \theta'\ fresh}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{unp}(\mathsf{async}(e')))} T \mid t[exp \hookleftarrow \mathcal{U}[\mathsf{unit}]] \mid t'[id \hookleftarrow \theta', exp \hookleftarrow e'] \mid P}$$

$$\textsc{Close}\quad \frac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{E}[\mathsf{unprotected}\ V]}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{unp}(V))} T \mid t[exp \hookleftarrow \mathcal{E}[V]] \mid P}$$

$$\textsc{Unprotect}\quad \frac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[\mathsf{unprotected}\ e]}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{unprtect})} T \mid t \mid P(\theta)}$$

$$\textsc{Undo}\quad \frac{}{T \mid P \xrightarrow{(\theta_1,\ldots,\theta_n,\mathsf{undo}(L))} init(\theta_1, \ldots, \theta_n, T) \mid \emptyset}$$

$$\textsc{Done}\quad \frac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathsf{unit}}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{done})} T \mid P(\theta)}$$

Figure 4.13: Optimistic Weak Threads Semantics with rollback

to evaluate with e and the init expression with e'. If the thread is not in the pool of process we create a new thread with id equals to θ, the expression to evaluate equal to e and the init expression equal to e'.

We denote the set of pending thread as P, an element of this set is indicate as $\lfloor t \rfloor_\theta$ where t is a thread and θ is the thread which create this pending thread. With $P(\theta)$ we activate all the pending thread created by θ, if $P = |_{i \in I}\ \lfloor t_i \rfloor_\theta \mid P'$, then $P(\theta) = |_{i \in I}\ t_i \mid P'$. With $init(\theta_1, \ldots, \theta_n, T)$ we perform the undo of all the threads whit id in $\theta_1, \ldots, \theta_n$, if $T = |_{i=1\ldots n}\ t_i \mid T'$ then $init(\theta_1, \ldots, \theta_n, T) = \ |_{i=1\ldots n}\ t_i[\mathsf{exp} \hookleftarrow t_i.\mathsf{init}] \mid T'$.

We show the thread semantics in Fig. 4.13.

Scheduler Semantics:

The scheduler is composed by three elements: $SC := \{S; A; L\}$. S is the set of the id, these ids correspond to the threads that can perform a protected operation.

L is a list of memory locations and their values, to be used as a log in undos. With $L.[r \mapsto v]$ we add an element memory location, value to L.

A is the description of the accesses that all threads have performed, which are used for conflict detection and which here is simply a set of tuples composed by the id and the list of reference locations. $A(\theta)$ is the set that correspond to the accesses done by the thread with id

$$\textsc{Activate}\ \frac{\theta \notin S}{\{S;A;L\} \xrightarrow{(\theta,\mathsf{act})} \{S \cup \{\theta\};A;L\}} \qquad \textsc{Step}\ \frac{\theta \in S}{\{S;A;L\} \xrightarrow{(\theta,\alpha)} \{S;A;L\}} \qquad \textsc{U Step/Close}\ \frac{\theta \notin S}{\{S;A;L\} \xrightarrow{(\theta,\beta)} \{S;A;L\}}$$

$$\textsc{Unprotect/Done}\ \frac{\theta \in S \wedge \mathsf{no\ conflicts}}{\{S;A;L\} \xrightarrow{(\theta,\mathsf{unprotect})} \{S \setminus \theta; A \setminus A(\theta); L - A(\theta)\}} \qquad \textsc{Undo}\ \frac{}{\{S;A;L\} \xrightarrow{(S,\mathsf{undo}(L))} \{\emptyset;\emptyset;\emptyset\}}$$

$$\textsc{Set (Write)}\ \frac{\theta \in S}{\{S;A;L\} \xrightarrow{(\theta,\mathsf{rd}(r,v))} \{S;A \cup \{\theta,r\};L.[r \mapsto \sigma(r)]\}} \qquad \textsc{DeRef (Read)}\ \frac{\theta \in S}{\{S;A;L\} \xrightarrow{(\theta,\mathsf{wr}(r,v))} \{S;A \cup \{\theta,r\};L\}}$$

Remark:

$\alpha ::= \mathsf{pure} \mid \mathsf{ref}(r,v) \mid \mathsf{rd}(r,v) \mid \mathsf{wr}(r,v) \mid \mathsf{blockUntil}(\mathsf{true}) \mid \mathsf{async}(e)$

$\beta ::= \mathsf{unp}(\alpha) \mid \mathsf{unp}(V)$

Figure 4.14: Optimistic Weak Scheduler Semantics

equal θ, $A(\theta) = \{(\theta,r) \mid (\theta,r) \in A\}$.

We show the scheduler semantics in Fig. 4.14.

Memory Semantics:

The memory is composed by a map, σ, that associates a memory location with a value. The possible operations are: $\sigma[r \mapsto v]$ that creates/updates the location r and map it with the value v, $\sigma(r)$ which returns the value associated with r in σ, and σL that restores the elements in σ with the elements of L, we have that $\sigma L = \forall\ [r \mapsto v] \in L$, then $\sigma[r \mapsto v]$. $L - A(\theta)$ deletes from L all the element that have memory location in $A(\theta)$.

$L - A(\theta) = \{[r \mapsto v] \mid (\theta,r) \notin A(\theta)\}$

We show the memory semantics in Fig. 4.15.

Labels:

We consider the synchronous products $AT \otimes_{\mathcal{R}} \{S;A;L\} \otimes_{\mathcal{R}} \sigma$ where $\mathcal{R}$ is composed of the following element:

$\langle(\theta,\mathsf{act}),\phi,(\theta,\mathsf{act})\rangle$ indicates the activation of a thread. This operation involves the scheduler and the thread model;

$\langle(\theta,\mathsf{pure}),\phi,(\theta,\mathsf{pure})\rangle$ is used to evaluate a lambda expression (with a protected action). This operation involves the scheduler and the thread model;

$\langle(\theta,\mathsf{unp}(\mathsf{pure})),\phi,(\theta,\mathsf{unp}(\mathsf{pure}))\rangle$ is used to evaluate a lambda expression (with an unprotected action). This operation involves the scheduler and the thread model;

$$\textsc{Ref}\ \frac{r \in (RefLoc - \mathsf{dom}(\sigma))}{\sigma \xrightarrow{(\theta,\mathsf{ref}(r,v))} \sigma[r \mapsto v]} \qquad \textsc{DeRef (Read)}\ \frac{\sigma(r) = v}{\sigma \xrightarrow{(\theta,\mathsf{rd}(r,v))} \sigma} \qquad \textsc{Set (Write)}\ \frac{}{\sigma \xrightarrow{(\theta,\mathsf{wr}(r,v))} \sigma[r \mapsto v]}$$

$$\textsc{URef}\ \frac{r \in (RefLoc - \mathsf{dom}(\sigma))}{\sigma \xrightarrow{(\theta,\mathsf{unp}(\mathsf{ref}(r,v)))} \sigma[r \mapsto v]} \qquad \textsc{UDeRef (Read)}\ \frac{\sigma(r) = v}{\sigma \xrightarrow{(\theta,\mathsf{unp}(\mathsf{wr}(r,v)))} \sigma} \qquad \textsc{USet (Write)}\ \frac{}{\sigma \xrightarrow{(\theta,\mathsf{unp}(\mathsf{rd}(r,v)))} \sigma[r \mapsto v]}$$

$$\textsc{Undo}\ \frac{}{\sigma \xrightarrow{(\theta_1,\ldots,\theta_n,\mathsf{undo}(L))} \sigma L}$$

Figure 4.15: Optimistic Weak Memory Semantics

$\langle(\theta, \mathsf{ref}(r, v)), (\theta, \mathsf{ref}(r, v)), (\theta, \mathsf{ref}(r, v))\rangle$ indicates the transition for creating a new reference in the memory (with a protected action). This operation involves all the system (scheduler, thread and memory model);

$\langle(\theta, \mathsf{unp}(\mathsf{ref}(r, v))), (\theta, \mathsf{unp}(\mathsf{ref}(r, v))), (\theta, \mathsf{unp}(\mathsf{ref}(r, v)))\rangle$ indicates the transition for creating a new reference in the memory (with an unprotected action). This operation involves all the system (scheduler, thread and memory model);

$\langle(\theta, \mathsf{rd}(r, v)), (\theta, \mathsf{rd}(r, v)), (\theta, \mathsf{rd}(r, v))\rangle$ is used to reading an element in the memory (with a protected action). This operation involves all the system (scheduler, thread and memory model);

$\langle(\theta, \mathsf{unp}(\mathsf{rd}(r, v))), (\theta, \mathsf{unp}(\mathsf{rd}(r, v))), (\theta, \mathsf{unp}(\mathsf{rd}(r, v)))\rangle$ is used to reading an element in the memory (with an unprotected action). This operation involves all the system (scheduler, thread and memory model);

$\langle(\theta, \mathsf{wr}(r, v)), (\theta, \mathsf{wr}(r, v)), (\theta, \mathsf{wr}(r, v))\rangle$ indicates the modification of a reference in the memory (with a protected action). This operation involves all the system (scheduler, thread and memory model);

$\langle(\theta, \mathsf{unp}(\mathsf{wr}(r, v))), (\theta, \mathsf{unp}(\mathsf{wr}(r, v))), (\theta, \mathsf{unp}(\mathsf{wr}(r, v)))\rangle$ indicates the modification of a reference in the memory (with an unprotected action). This operation involves all the system (scheduler, thread and memory model);

$\langle(\theta, \mathsf{async}(e)), \phi, (\theta, \mathsf{async}(e))\rangle$ indicates the creation of a new thread (with a protected action). This operation involves the scheduler and the thread model;

$\langle(\theta, \mathsf{unp}(\mathsf{async}(e))), \phi, (\theta, \mathsf{unp}(\mathsf{async}(e)))\rangle$ indicates the creation of a new thread (with an unprotected action). This operation involves the scheduler and the thread model;

$\langle(\theta, \mathsf{blockUntil}(\mathsf{true})), \phi, (\theta, \mathsf{blockUntil}(\mathsf{true}))\rangle$ indicates the success of an asynchronous execution (with a protected action). This operation involves the scheduler and the thread model;

$\langle(\theta, \mathsf{unp}(\mathsf{blockUntil}(\mathsf{true}))), (\theta, \mathsf{unp}(\mathsf{blockUntil}(\mathsf{true}))), (\theta, \mathsf{unp}(\mathsf{blockUntil}(\mathsf{true})))\rangle$ indicates the success of an asynchronous execution (with an unprotected action). This operation involves all the system (scheduler, thread and memory model);

$\langle(\theta_1, \ldots, \theta_n, \mathsf{undo}(l)), (\theta_1, \ldots, \theta_n, \mathsf{undo}(l)), (\theta_1, \ldots, \theta_n, \mathsf{undo}(l))\rangle$ indicates the undo of the thread with id $\theta_1, \ldots, \theta_n$. This operation involves all the system (scheduler, thread and memory model);

$\langle(\theta, \mathsf{unprotect}), \phi, (\theta, \mathsf{unprotect})\rangle$ indicates the start of unprotected evaluation. This operation involves the scheduler and the thread model;

$\langle \mathsf{done}, \phi, \mathsf{done}\rangle$ indicates the finish of a protected thread. This operation involves the scheduler and the thread model;

$\langle(\theta, \mathsf{unp}(V)), \phi, (\theta, \mathsf{unp}(V))\rangle$ indicates the finish of an unprotected evaluation. This operation involves the scheduler and the thread model;

Bi-similarity:

Now we have introduce the element to define our model we aim to establish a bi-simulation (see Def.31) between the weak semantics with optimistic rollback for AME described in [16] and the product of LTSs $AT \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma$.

To do this we need to define the function for the label and the relation between the states of the two models.

Definition 35. *The function $\varphi(\cdot)$ is defined over labels:*

$$\varphi(\alpha, \phi, \alpha) = \alpha,$$
$$\varphi(\beta, \beta, \beta) = \beta,$$

where $\alpha = (\theta, \mathsf{act}), (\theta, \mathsf{pure}), (\theta, \mathsf{blockUntil}(\mathsf{true})), (\theta, \mathsf{async}(e)), (\theta, \mathsf{unp}(\mathsf{pure})), (\theta, \mathsf{unp}(\mathsf{async}(e))),$
$(\theta, \mathsf{unp}(\mathsf{blockUntil}(\mathsf{true}))), (\theta, \mathsf{unprotect}), \mathsf{done}, (\theta, \mathsf{unp}(V))$*;*
and where $\beta = (\theta, \mathsf{ref}(r, v)), (\theta, \mathsf{rd}(r, v)), (\theta, \mathsf{wr}(r, v)), (\theta, \mathsf{unp}(\mathsf{ref}(r, v))), (\theta, \mathsf{unp}(\mathsf{rd}(r, v))),$
$(\theta, \mathsf{unp}(\mathsf{wr}(r, v)))$.

Theorem 12. $\langle \sigma_A, E, O, l\rangle \sim AT \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma$

$$\begin{array}{ccc} \langle \sigma_A, E, O, l\rangle & \underset{Rel^{-1}}{\overset{Rel}{\longleftrightarrow}} & (AT \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma) \\ \Big\downarrow{\scriptstyle \varphi(\alpha)} & & \Big\downarrow{\scriptstyle \alpha} \\ \langle \sigma'_A, E', O', l'\rangle & \underset{Rel^{-1}}{\overset{Rel}{\longleftrightarrow}} & (AT' \otimes_{\mathcal{R}} \{S'; A'; L'\} \otimes_{\mathcal{R}} \sigma') \end{array}$$

Proof. We start by defining *Rel* and then show that it is a strong similarity relation, after which we will show that Rel^{-1} is also a strong similarity relation.

Given $\{\langle \sigma_A, E, O, l\rangle, \langle T \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\}$ then this tuple is in *Rel* if:

- $\sigma = \sigma_A$

- $\forall e_i \in E$ exist θ_i such that $t_i \in T$ and $t_i.\mathsf{id} = \theta_i \wedge t_i.\mathsf{exp} = e_i$
- $\forall o = (e, f, a, P_o) \in O$ we have that exist θ_o such that:
  - exist $t_o \in T$ and $t_o.\mathsf{id} = \theta_o$, $t_i.\mathsf{exp} = e$ and $t_o.\mathsf{init} = f$
  - $\theta_o \in S$
  - $\forall r \in a$ then $(\theta_o, r) \in A$
  - $\forall p \in P_o$ then exist θ_p such that exist $\lfloor t[\mathsf{id} \leftarrow\!\shortmid \theta_p, \mathsf{exp} \leftarrow\!\shortmid p] \rfloor_{\theta_o} \in P$
- $L = l$

After the definition of *Rel* we start to give for each label the proof that there is a strong similarity between Abadi system and our system:

- if in Abadi.system we have this transition:

  $$\textsc{Trans Activate}$$
  $$\langle \sigma_A, E.(\theta, e).E', O, l\rangle \xmapsto{(\theta,\mathsf{act})} \langle \sigma_A, E.E', ((\theta, e), e, \emptyset, \emptyset).O, l\rangle$$

  in our system we have:

  $$\frac{\textsc{Thread Activate}\quad t.\mathsf{id} = \theta}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{act})} T \mid t[\mathsf{init} \leftarrow\!\shortmid t.\mathsf{exp}] \mid P} \qquad \frac{\textsc{Activate}\quad \theta \notin S}{\{S; A; L\} \xrightarrow{(\theta,\mathsf{act})} \{S \cup \{\theta\}; A; L\}}$$
  $$\frac{}{T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta,\mathsf{act})} T \mid t[\mathsf{init} \leftarrow\!\shortmid t.\mathsf{exp}] \mid P \otimes_{\mathcal{R}} \{S \cup \{\theta\}; A; L\} \otimes_{\mathcal{R}} \sigma}$$

  We know that $\{\langle \sigma_A, E.(\theta, e).E', O, l\rangle, \langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$.

  Then after the transitions we have that $\{\langle \sigma_A, E.E', ((\theta, e), e, \emptyset, \emptyset).O, l\rangle$,
  $\langle T \mid t[\mathsf{init} \leftarrow\!\shortmid t.\mathsf{exp}] \mid P \otimes_{\mathcal{R}} \{S \cup \{\theta\}; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

  This couple belongs to the relation *Rel* because we have that $\sigma = \sigma_A$ and $L = l$ because none of the rules applied to the two systems changed the memory and the logs.

  We also have that all the expressions in E and E' have a corresponding thread in T (none of the rules applied to the two systems changed either E and E' or T) and the thread t corresponds to $(e, e, \emptyset, \emptyset)$ (because $\theta \in S$, $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = e$ and $t.\mathsf{init} = e$). A and P do not change in fact in $(e, f, a, P_o).O$ we have a and P_o equal to $\emptyset$.

To proof the bi-simulation we can prove that given *Rel* as a strong simulation relation then if also Rel^{-1} is a strong simulation relation then *Rel* is a bi-simulation relation. Now we show the relationship Rel^{-1} derived directly from *Rel*.

Given $\{\langle T \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, O, l\rangle\}$ then this tuple is in Rel^{-1} if:

- $\sigma_A = \sigma$
- $\forall \theta_i \in S$:
  - we have $t_i \in T$ such that $t_i.\mathsf{id} = \theta_i$, $t_i.\mathsf{exp} = e_i$ and $t_i.\mathsf{init} = f_i$,

  - we have a set of $\lfloor t'_i \rfloor_{\theta_i} \in P$ where $t'_i.\mathsf{exp} = e'_i$, the set of this expression is indicate with E_i
  - then exist $o \in O$ such that $o = (e_i, f_i, A(\theta_i), E_i)$

- $\forall \theta_j \notin S$ we have $t_j \in T$ such that $t_j.\mathsf{id} = \theta_j$ and $t_j.\mathsf{exp} = e_j$, then exist $e_j \in E$
- $l = L$

Now we start to analyses all the possible case:

- if in our system we have:

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta,\mathsf{act})} T \mid t \mid P \otimes_{\mathcal{R}} \{S \cup \{\theta\}; A; L\} \otimes_{\mathcal{R}} \sigma$$

  in Abadi.system we have this transition:

$$\textsc{Trans Activate}$$
$$\langle \sigma_A, E.(\theta, e).E', O, l\rangle \xmapsto{(\theta,\mathsf{act})} \langle \sigma_A, E.E', ((\theta, e), e, \emptyset, \emptyset).O, l\rangle$$

  We know that $\{\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta, e).E', O, l\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$.

  Then after the transitions we have that $\{\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S \cup \{\theta\}; A; L\} \otimes_{\mathcal{R}} \sigma\rangle$, $\langle \sigma_A, E.E', ((\theta, e), e, \emptyset, \emptyset).O, l\rangle\} \in Rel^{-1}$.

  This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ and $l = L$ because none of the rules applied to the two systems changed the memory and the logs.

  We also have that all the expressions in E and E' have a corresponding thread in T (none of the rules applied to the two systems changed either E and E' or T) and $(e, e, \emptyset, \emptyset)$ correspond to the thread t (in fact $\theta \in S$, $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = e$ and $t.\mathsf{init} = e$). A and P do not change in fact we add to O the element $(e, E, \emptyset, \emptyset).O$.

The other cases are similar and they are shown in Appendix B.3.3. □

Chapter 5

A compositional theory of causal consistent reversibility

The aim of this chapter is to explore Causal Consistency in concurrent systems described as synchronous product of labeled transition systems. We develop a reversible semantics for these systems, investigate Causal Consistency, and apply our theory to the sequential consistency memory model.

First, we discuss how to define a reversible semantics for synchronous products. We then propose two approaches for establishing Causal Consistency within synchronous products. The main approach involves proving Causal Consistency by establishing four properties: Loop Lemma, Square Property, Backward Transitions are Independent, and Well-Foundedness. The alternative approach assumes the causally consistent nature of the transition systems within synchronous products and aims to prove their causal consistency.

We then provide an example of applying the discussed concepts to a model of sequential consistency memory. This example describes the semantics of thread execution, memory operations, and scheduler actions, both forward and backward, within the context of the sequential consistency model.

In summary, this chapter is structured as follow. In Section 5.1 we describe how to obtain the reversible semantics in a synchronous product of labeled transition system. In Section 5.2 we discuss how to use this theory using as example the sequential consistency memory model describes in the previews chapter in Section 4.4, and we show the benefit of applying our theory.

It is possible note that we do not provide a state of the art review because, to the best of our knowledge, there are not any study on reversing synchronous products of LTSs or on proving causal-consistency for them.

5.1 Reversing Synchronous Products

We begin by introducing the notion of a Labeled Transition System with Independence (LTSI) with idle transitions. Idle transitions indicate that the LTS is not involved in any synchronization. This concept will be used later in the reverse synchronous product of LTSs to represent scenarios where an LTS does not participate in the synchronization process.

Definition 36 (LTSI with idle transitions)**.** *Given an LTSI $L = (\mathcal{S}, \Lambda, \rightarrow, \iota)$, we define the LTSI with idle transitions as $L^\phi = (\mathcal{S}, \Lambda^\phi, \rightarrow^\phi, \iota^\phi)$. Here, ι^ϕ represents the extended independence relation that considers idle transitions. It is defined as the union of three sets $\iota^\phi = \iota \cup \iota_1 \cup \iota_2$ where ι is the original independence relation, ι_1 is the set of independence between idle transitions, defined as $\iota_1 = \{\{p \xrightarrow{\phi} p, q \xrightarrow{\phi} q\} \mid p, q \in \mathcal{S}\}$, and ι_2 is the set of independence between idle transitions and other transitions, defined as $\iota_2 = \{\{p \xrightarrow{\ell} p', q \xrightarrow{\phi} q\} \mid p \xrightarrow{\ell} p' \in \rightarrow \text{ and } q \in \mathcal{S}\}$.*

This definition ensures that the independence relation ι^ϕ accounts for both idle transitions and their interactions with other transitions in the system.

Definition 37 (Synchronous products with Independence)**.** *Let $L_1, \ldots, L_n$ be LTSI, we consider the corresponding LTSIs that also include idle transitions: $L_i^\phi = (\mathcal{S}_i, \Lambda_i^\phi, \rightarrow_i^\phi, \iota^\phi)$. The synchronous product with Independence $L_1 \otimes_\mathcal{R} \ldots \otimes_\mathcal{R} L_n$ is defined by the product of the correspondent LTSs where the notion of independence ι is defined on the transition as follows:*

$$p_1 \otimes_\mathcal{R} \ldots \otimes_\mathcal{R} p_n \xrightarrow{\langle a_1, \ldots, a_n \rangle} p_1' \otimes_\mathcal{R} \ldots \otimes_\mathcal{R} p_n' \;\iota\; q_1 \otimes_\mathcal{R} \ldots \otimes_\mathcal{R} q_n \xrightarrow{\langle b_1, \ldots, b_n \rangle} q_1' \otimes_\mathcal{R} \ldots \otimes_\mathcal{R} q_n'$$

if $\forall i \in \{1, \ldots, n\}$ $p_i \xrightarrow{a_i} p_i'$ ι_i^ϕ $q_i \xrightarrow{b_i} q_i'$.

When defining a reversible causal-consistent semantics, the initial step involves defining the backward semantics. This entails specifying how the system can revert from a given state to a previous state, undoing the effects of previous operations. In this section, we introduce the concept of a reversible synchronous product LTSs.

Definition 38 (Reverse synchronous products)**.** *Let $L_1, \ldots, L_n$ be LTSs, we consider the corresponding LTSs that include idle transitions $L_i^\phi = (\mathcal{S}_i, \Lambda_i^\phi, \rightarrow_i^\phi)$ and their reverse (that consider the idle transitions) $(\mathcal{S}_i, \Lambda_i^\phi, \rightsquigarrow_i^\phi)$. A synchronization relation ($\mathcal{R}$) is $\mathcal{R} \subseteq \Lambda_1^\phi \times \cdots \times \Lambda_n^\phi \setminus \{\langle \phi_1, \ldots, \phi_n \rangle\}$. The reverse synchronous product of $L_1, \ldots, L_n$ indicated with $L_1 \otimes_\mathcal{R} \ldots \otimes_\mathcal{R} L_n$ is defined as:*

- $\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$,
- *the labeled transition relation is defined by rules* Sync *and* $\overline{\text{Sync}}$.

$$\textsc{Sync}\ \frac{p_1 \xrightarrow{\ell_1} q_1 \quad \ldots \quad p_n \xrightarrow{\ell_n} q_n \quad \langle \ell_1, \ldots, \ell_n \rangle \in \mathcal{R}}{p_1 \otimes_\mathcal{R} \ldots \otimes_\mathcal{R} p_n \xrightarrow{\langle \ell_1, \ldots, \ell_n \rangle} q_1 \otimes_\mathcal{R} \ldots \otimes_\mathcal{R} q_n}$$

where $p_i \xrightarrow{\ell_i} q_i \in \rightarrow_i^\phi$,

$$\overline{\textsc{Sync}}\ \frac{p_1 \overset{\ell_1}{\rightsquigarrow} q_1 \quad \ldots \quad p_n \overset{\ell_n}{\rightsquigarrow} q_n \quad \langle \ell_1, \ldots, \ell_n \rangle \in \mathcal{R}}{p_1 \otimes_\mathcal{R} \ldots \otimes_\mathcal{R} p_n \overset{\langle \ell_1, \ldots, \ell_n \rangle}{\rightsquigarrow} q_1 \otimes_\mathcal{R} \ldots \otimes_\mathcal{R} q_n}$$

where $q_i \overset{\ell_i}{\rightsquigarrow} p_i \in \rightsquigarrow_i^\phi$.

Another approach to consider reversible LTSs (or the reversible product of LTSs) involves defining a set of labels derived from the forward transitions to establish the corresponding backward transitions $\underline{\mathcal{R}} = \{\underline{\ell} : \ell \in \mathcal{R}\}$. Consequently, a single transition type can be defined

as the combination of the possible forward and backward transitions $\rightarrow \cup \rightsquigarrow$ with the possible labels $\mathcal{R} \cup \underline{\mathcal{R}}$. Whether we use our definition or one that uses (negated) labels for backward transitions, the concept is the same. The difference is that in our approach we look at the relationship between the states to see if it is a forward or backward operation (if we have $\rightarrow$ or $\rightsquigarrow$), whereas in the other case we look at the label of the transition to see if it is a forward or backward operation (if we have ℓ or $\underline{\ell}$).

Theorem 13. *Given n LTSs $L_1, \ldots, L_n$, where $L_i = (S_i, \Lambda_i, \rightarrow_i)$ and their combined version $L_i^r = (S_i, \Lambda_i, \twoheadrightarrow_i = \rightarrow_i \cup \rightsquigarrow_i)$, then the reverse synchronous product $L_i \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$ is equal to the synchronous product $L_i^r \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n^r$.*

Proof. If we consider a transition in the synchronous product of $L_i \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$, then the synchronous product $L_i^r \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n^r$ can perform the same transition, since we have the same $\mathcal{R}$, and since $L_i^r \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n^r$ can perform the same move with the synchronization relation $\twoheadrightarrow$. This because the synchronization relation $\twoheadrightarrow$ is equal to $\rightarrow$ when we have a forward move and is equal to $\rightsquigarrow$ when we consider a backward move. □

Before presenting our theory, we define the projection operation and a related result. The projection operation takes a trace of a product of LTSs and a position i and returns the trace of the LTS at position i.

Definition 39 (Projection (Proj))**.** *The projection of a trace is defined as follows:*

- $\mathsf{Proj}_i((p_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n \xrightarrow{\langle a_1,\ldots,a_n\rangle} p_1' \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n')s) = p_i \xrightarrow{a_i} p_i'\ \mathsf{Proj}_i(s)$
- $\mathsf{Proj}_i((p_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n \xrightarrow{\langle a_1,\ldots,\phi_i,\ldots,a_n\rangle} p_1' \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n')\ s) = \mathsf{Proj}_i(s)$
- $\mathsf{Proj}_i(\epsilon) = \epsilon$

Lemma 11. *Given two derivations of a product of LTSs s and r co-initial and co-final then $s_i = \mathsf{Proj}_i(s)$ and $r_i = \mathsf{Proj}_i(r)$ are co-initial and co-final.*

Proof. We know that s and r are co-initial and co-final then by the definition of co-initial (starting from the same state) and co-final (ending in the same state) we know that they have the same initial and final states. Therefore, if s_i and r_i were not co-initial or co-final, we would have a different starting or ending state in one of the projections. However, this cannot happen since s and r are co-initial and co-final. □

After defining the concepts of synchronous product and reversible synchronous product of LTSs, let us explore how the synchronous product of LTSs behaves depending on the properties of its components. We aim to demonstrate that a product of n LTSs, $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$, is causally consistent, considering the properties of its components L_i. We develop two approaches: the first one considers the Causal Consistency of all the LTSs in the product based on the properties Loop Lemma, SP, BTI, and WF. The second approach proves it directly, considering that all the components are only causally consistent. In both the approaches we need a restriction on the possible synchronization between transitions we can have. We show an example to explain why we need this restriction.

Example 10. Consider the product of two LTSs causal consistent. Now, we consider two trace $s = p \otimes_{\mathcal{R}} q \xrightarrow{\langle a,b \rangle} p' \otimes_{\mathcal{R}} q'$ and $r = p \otimes_{\mathcal{R}} q \xrightarrow{\langle a,\phi \rangle} p' \otimes_{\mathcal{R}} q \xrightarrow{\langle \phi,b \rangle} p' \otimes_{\mathcal{R}} q'$. Here, we can see that s and r are co-initial and co-final but we cannot state that $p \otimes_{\mathcal{R}} q \xrightarrow{\langle a,b \rangle} p' \otimes_{\mathcal{R}} q' \approx p \otimes_{\mathcal{R}} q \xrightarrow{\langle a,\phi \rangle} p' \otimes_{\mathcal{R}} q \xrightarrow{\langle \phi,b \rangle} p' \otimes_{\mathcal{R}} q'$ because it does not match any cases of the definition of causal equivalence (Definition 13), since in r we have two transitions not annullable and s is composed of one transition. ◊

This example illustrates the need to add a restriction. To address this issue, we have decided to resolve it by adding a restriction that ensures a transition of an LTS in the product cannot synchronize with multiple transitions. We refer to this property as the Unicity of Transitions, and it is formally defined below.

Definition 40 (Unicity of Transitions)**.** *A product of LTSs $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$ satisfies Unicity of Transitions if for each different co-initial transitions $t : p_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n \xrightarrow{\langle a_1,\ldots,a_n \rangle} p'_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p'_n$ and $u : p_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n \xrightarrow{\langle b_1,\ldots,b_n \rangle} p''_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p''_n$ if there exists $i \in \{1,\ldots,n\}$ such that $a_i = b_i$ (the transitions have a common label) then $p'_i \neq p''_i$. This means that the two transitions are different ($p_i \xrightarrow{a_i} p'_i \neq p_i \xrightarrow{b_i} p''_i$).*

This condition resolves the issue presented in the previous example (Example 10), as it prevents a single action from synchronizing with multiple actions (including the idle action).

Now we briefly introduce the two approaches. The main approach involves exploring whether individual causally consistent LTSs exhibit specific properties and verifying if these are preserved in the product. This approach aims to establish Causal Consistency within the framework described in [35]. Starting from a transition system with forward/backward transitions and a notion of independence, proving Causal Consistency requires proving four properties: Loop Lemma, Square Property (SP), Backward Transitions are Independent (BTI), and Well-Foundedness (WF).

The alternative approach assumes that the LTSs in our synchronous product are causally consistent and demonstrates the product's Causal Consistency without considering additional properties.

5.1.1 Main approach

In this section, we examine the properties of Loop Lemma, Square property, Backward Transitions are Independent, and Well-Foundedness in the product of LTSs denoted as $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$, with the assumption that each LTS in the product exhibits the same properties. We show that when one of these properties holds in all individual LTSs, it also holds in their product, demonstrating their compositional nature. Subsequently, we demonstrate that when all these properties are satisfied in the product of LTSs, Causal Consistency also holds, as demonstrated in [35].

Theorem 14 (Compositionality of Loop Lemma)**.** *Given n LTSs $L_1, \ldots, L_n$ satisfying the loop lemma, the loop lemma holds in the product of LTSs $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$.*

Proof. To prove Loop Lemma in $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$, we need to show that $p_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n \xrightarrow{\langle \ell_1,\ldots,\ell_n \rangle} p'_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p'_n$ iff $p'_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p'_n \xrightarrow{\langle \ell_1,\ldots,\ell_n \rangle} p_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n$. Since we have a transition we know

that $\langle \ell_1, \ldots, \ell_n \rangle \in \mathcal{R}$ because we have applied rule SYNC (resp. $\overline{\text{SYNC}}$). For each L_i we know that the Loop Lemma stands, so if $\ell_i \neq \phi$ then we have that $p_i \xrightarrow{\ell_i} p_i'$ iff $p_i' \overset{\ell_i}{\rightsquigarrow} p_i$ otherwise it is possible to observe that $p_i \xrightarrow{\phi} p_i$ iff $p_i \overset{\phi}{\rightsquigarrow} p_i$. Knowing that we can conclude the thesis because we can apply the rule $\overline{\text{SYNC}}$ (resp. SYNC). □

Theorem 15 (Compositionality of Square Property)**.** *Given n LTSIs $L_1, \ldots, L_n$ satisfying the Square Property, the Square Property holds in the product of LTSIs $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$.*

Proof. To prove Square Property in $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$, we need to show that given two transitions $t : p_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n \xrightarrow{\langle a_1,\ldots,a_n \rangle} p_1' \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n'$ and $u : p_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n \xrightarrow{\langle b_1,\ldots,b_n \rangle} p_1'' \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n''$ if $t \iota u$ then there exist:

$$u' : p_1' \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n' \xrightarrow{\langle b_1,\ldots,b_n \rangle} p_1''' \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n''' \text{ and}$$
$$t' : p_1'' \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n'' \xrightarrow{\langle a_1,\ldots,a_n \rangle} p_1''' \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n'''$$

To prove that we need to show that it is possible to apply rule SYNC for the transitions u' and t'. Since we have the two transitions t and u we know that $\langle a_1, \ldots, a_n \rangle \in \mathcal{R}$ and $\langle b_1, \ldots, b_n \rangle \in \mathcal{R}$ because we have applied the rule SYNC. We have a case analysis on the labels where for each L_i we know that the Square Property stands. Since we have that $p_i \xrightarrow{a_i} p_i'$ and $p_i \xrightarrow{b_i} p_i''$ then if:

- we have that $a_i \neq \phi$ and $b_i \neq \phi$ we can conclude that $p_i'' \xrightarrow{a_i} p_i'''$ and $p_i' \xrightarrow{b_i} p_i'''$
- we have in both cases $p_i \xrightarrow{\phi} p_i$ (we have that $a_i = \phi = b_i$) then in both the transitions (u' and t') we apply the idle transition $p_i \xrightarrow{\phi} p_i$.
- we have that $a_i \neq \phi$ and $b_i = \phi$ (in this case we can note that p_i is equal to p_i'' and p_i' is equal to p_i''') the transition of u' and t' are respectively $p_i \xrightarrow{a_i} p_i'$ and $p_i' \xrightarrow{\phi} p_i'$.
- we have that $a_i = \phi$ and $b_i \neq \phi$ is similar to the previous one.

Here for all the cases the transition u' and t' exist. Then the thesis follows. □

Theorem 16 (Compositionality of Backward Transitions are Independent (BTI))**.** *Given n LTSIs $L_1, \ldots, L_n$ satisfying Backward Transitions are Independent, then Backward Transitions are Independent holds in the product of LTSIs $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$.*

Proof. To prove Backward Transitions are Independent in $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$, we need to show that if whenever $t : p_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n \overset{\langle a_1,\ldots,a_n \rangle}{\rightsquigarrow} q_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} q_n$ and $t' : p_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n \overset{\langle b_1,\ldots,b_n \rangle}{\rightsquigarrow} q_1' \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} q_n'$ and $t \neq t'$ then $t \iota t'$. By Unicity of Transaction (Def. 40) we have for all $i \in \{1, \ldots, n\}$ that $p_i \overset{a_i}{\rightsquigarrow} p_i' \neq p_i \overset{b_i}{\rightsquigarrow} p_i''$. Assume for contradiction that $t \not\iota t'$. Since $t \not\iota t'$ we have that there exists $i \in \{1, \ldots, n\}$ such that $p_i \overset{a_i}{\rightsquigarrow} p_i' \not\iota_i^{\phi} p_i \overset{b_i}{\rightsquigarrow} p_i''$. But this it is impossible because we know that BTI holds in the individual LTSs and then for each i we have that $p_i \overset{a_i}{\rightsquigarrow} p_i' \iota_i^{\phi} p_i \overset{b_i}{\rightsquigarrow} p_i''$. We also have the independence between idle actions and all the other actions $p_i \overset{\phi}{\rightsquigarrow} p_i' \iota_i^{\phi} p_i \overset{b_i}{\rightsquigarrow} p_i''$, with b_i possibly equal to ϕ. □

To prove the next theorem we need to recall the Definition **??** of synchronous product of LTSs where it is not possible to have a idle transition (this mean with the label $\langle \phi_1, \ldots, \phi_n \rangle$) over the synchronous product.

Theorem 17 (Compositionality of Well-Foundedness (WF))**.** *Given n LTSIs $L_1, \ldots, L_n$ satisfying the Well-Foundedness, the Well-Foundedness holds in the product of LTSIs $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$.*

Proof. To prove Well-Foundedness in $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$, we need to show that there is no infinite reverse computation. We know that for each L_i the Well-Foundedness holds. We also know, thanks to the definition of synchronous product, that we cannot have an idle transition (it is not possible to have $p_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n \xrightarrow{\langle \phi_1, \ldots, \phi_n \rangle} p_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} p_n$). Then if the Well-Foundedness did not hold in the product of LTSs then we have at least one of the LTS where WF did not hold, but this is not possible since WF holds in the individual LTSs. □

Theorem 18 (Causal Consistency)**.** *Given n LTSIs $L_1, \ldots, L_n$ satisfying Loop Lemma, Square property, BTI and WF, Causal Consistency holds in the product of LTSIs $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$.*

Proof. Since $L_1, \ldots, L_n$ satisfy loop lemma, square property, Backward Transitions are Independent and Well-Foundedness then thanks to the Theorems 14, 15 16 and 17 we know that these properties hold also in the product of LTSIs $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$. Then since the four properties (Loop Lemma, Square Property, Backward Transitions are Independent, and Well-Foundedness) hold due to [35, Proposition 3.6], Causal Consistency holds in $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$. □

5.1.2 Alternative approach

In this section, we explore the property of Causal Consistency in the product of LTSs denoted as $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$, where we assume it in each LTS of the product. Despite this assumption, we present one example that highlights the need for additional properties to ensure Causal Consistency in the product of LTSs. This emphasizes that relying solely on Causal Consistency within each individual LTS is insufficient to guarantee it in their product. We illustrate this aspect through the following example that emphasizes the necessity for supplementary properties.

Example 11. Consider the following derivations:

$$s = p \otimes_{\mathcal{R}} q \xrightarrow{\langle a,b \rangle} p_1 \otimes_{\mathcal{R}} q_1 \xrightarrow{\langle a',b' \rangle} p_2 \otimes_{\mathcal{R}} q_2 \overset{\langle a,b \rangle}{\rightsquigarrow} p_3 \otimes_{\mathcal{R}} q_3 \xrightarrow{\langle a'',b'' \rangle} p_4 \otimes_{\mathcal{R}} q_4 \xrightarrow{\langle a,b \rangle} p_5 \otimes_{\mathcal{R}} q_5$$

and

$$r = p \otimes_{\mathcal{R}} q \xrightarrow{\langle a',b' \rangle} p_6 \otimes_{\mathcal{R}} q_6 \xrightarrow{\langle a,b \rangle} p_7 \otimes_{\mathcal{R}} q_7 \xrightarrow{\langle a'',b'' \rangle} p_5 \otimes_{\mathcal{R}} q_5$$

We consider the following relation of independence:

$$p_1 \xrightarrow{a'} p_2 \iota p_2 \overset{a}{\nrightarrow} p_3, \qquad q_2 \overset{b}{\nrightarrow} q_3 \iota q_3 \xrightarrow{b''} q_4,$$
$$p_3 \xrightarrow{a''} p_4 \iota p_4 \overset{a}{\nrightarrow} p_5, \qquad p \overset{b}{\nrightarrow} p_1 \iota p_1 \xrightarrow{b'} p_2$$

Despite s and r being co-initial and co-final, these two computations are not causally equivalent. Indeed, we cannot perform the swaps $p_1 \otimes_{\mathcal{R}} q_1 \xrightarrow{\langle a',b' \rangle} p_2 \otimes_{\mathcal{R}} q_2 \overset{\langle a,b \rangle}{\rightsquigarrow} p_3 \otimes_{\mathcal{R}} q_3$ or $p_2 \otimes_{\mathcal{R}} q_2 \overset{\langle a,b \rangle}{\rightsquigarrow} p_3 \otimes_{\mathcal{R}} q_3 \xrightarrow{\langle a'',b'' \rangle} p_4 \otimes_{\mathcal{R}} q_4$ because, as we can see, $p_2 \overset{a}{\nrightarrow} p_3 \not\iota p_3 \xrightarrow{a''} p_4$ and $q_1 \xrightarrow{b'} q_2 \not\iota q_2 \overset{b}{\nrightarrow} q_3$.

Note that instead the projections are causal equivalent:

$$\mathsf{Proj}_1(s) = p \xrightarrow{a} p_1 \xrightarrow{a'} p_2 \overset{a}{\dashrightarrow} p_3 \xrightarrow{a''} p_4 \xrightarrow{a} p_5 \approx p \xrightarrow{a} p_1 \overset{a}{\dashrightarrow} p \xrightarrow{a'} p_6 \xrightarrow{a} p_7 \xrightarrow{a''} p_5 \approx p \xrightarrow{a'} p_6 \xrightarrow{a} p_7 \xrightarrow{a''} p_5 = \mathsf{Proj}_1(r)$$

$$\mathsf{Proj}_2(s) = q \xrightarrow{b} q_1 \xrightarrow{b'} q_2 \overset{b}{\dashrightarrow} q_3 \xrightarrow{b''} q_4 \xrightarrow{b} q_5 \approx q \xrightarrow{b'} q_6 \xrightarrow{b} q_7 \xrightarrow{b''} q_5 \overset{b}{\dashrightarrow} q_8 \xrightarrow{b} q_5 \approx q \xrightarrow{b'} q_6 \xrightarrow{b} q_7 \xrightarrow{b''} q_5 = \mathsf{Proj}_2(r)$$

This counterexample can be ruled out by requiring that $t \iota t'$ implies $t \iota \underline{t}'$. ◊

This example motivates the introduction of an additional constraint: Reversing Preserves Independence (RPI) from [35]. This property states that if transitions t and t' are independent ($t \iota t'$) then t is independent with the reversed version of t' ($t \iota \underline{t}'$).

Definition 41 (Reversing Preserves Independence (RPI))**.** *Given an LTSI L we say that L satisfies Reversing Preserves Independence if whenever $t \iota t'$ then $t \iota \underline{t}'$.*

Now, we show that under RPI we have that if $tt' \approx t't$ then $t \iota t'$, this result is crucial to prove that Causal Consistency holds in the product of LTSs.

Lemma 12. *Consider an LTSI satisfying RPI. Given two derivations, if $r_i t t' r_e \approx r_i t' t r_e$ then $t \iota t'$.*

Proof. Consider the function $c(\cdot)$ on traces defined as follows: it counts how many times t appears before t' plus how many times $\underline{t}$ appears before $\underline{t}'$, minus how many times t appears before $\underline{t}'$, and minus how many times $\underline{t}$ appears before t'.

We observe that $c(d) = c(d' d d'')$ if d' is a derivation that does not contain either t or $\underline{t}$ and d'' is a derivation that does not contain either t' or $\underline{t}'$. This is because adding t' or $\underline{t}'$ in the beginning cannot change the number of occurrences of t before t', and similarly, adding t or $\underline{t}$ at the end does not change this count. Additionally, $c(d) = c(t \underline{t} d)$ because we add a number x, representing the occurrences of t before t', offset by the occurrences of $\underline{t}$ before t'. Similarly, we add a number y representing the occurrences of $\underline{t}$ before $\underline{t}'$, offset by the occurrences of t before $\underline{t}'$. The same logic applies for $c(d) = c(d t' \underline{t}')$.

Note that $r_i t t' r_e \approx r_i t' t r_e$ iff $\underline{r}_i r_i t t' r_e \underline{r}_e \approx \underline{r}_i r_i t' t r_e \underline{r}_e$ (closure under contexts of $\approx$) iff $tt' \approx t't$ (delete operation and transitivity of $\approx$). Note that $c(tt') = 1$ while $c(t't) = 0$. Since $tt' \approx t't$ there is a sequence of delete and swap operations transforming tt' into $t't$. Note that, as discussed above, delete operations do not change c. Also, swap operations involving at most one among t, t', $\underline{t}$, $\underline{t}'$ do not change c either. Hence, the derivation should include at least a swap operation involving two of the transitions above. Hence, we need independence among two of them. We have at least one of the following cases:

- $t \iota t'$, which proves the thesis;
- $t \iota \underline{t}'$, which proves the thesis by RPI;
- $\underline{t} \iota t'$, which prove the thesis since ι is symmetric, and we have $t' \iota \underline{t}$, then by RPI we have $t \iota t'$;
- $\underline{t} \iota \underline{t}'$, which proves the thesis since by RPI, we have $\underline{t} \iota t'$, so we are in the previous case where $t \iota t'$.

In all the cases the thesis follows. □

In the context of a product of LTSs $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$ and given a transition t of one of the LTSs L_i, we introduce the notation $\tilde{t}$. This notation represents a transition of the product of LTSs such that its projection $\mathsf{Proj}_i(\tilde{t})$ yields the original transition t. The uniqueness and existence of $\tilde{t}$ are guaranteed by the Unicity of Transitions property (Definition 40).

Definition 42. *Given a product of LTSs $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$ and a transition t of one of the LTSs L_i we define $\tilde{t}$ as a possible transition of $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$ such that $\mathsf{Proj}_i(\ \tilde{t}\ ) = t$.*

We know that this transition exists and it is unique since the Unicity of Transitions stands.

Lemma 13. *Given a product of LTSs $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$ satisfying the Unicity of Transitions and a transition t of one of the LTSs L_i, then $\tilde{t}$ is unique.*

Proof. Assume for contradiction that $\tilde{t}$ is not unique then there exist $\tilde{t}_1 \neq \tilde{t}$ such that $\mathsf{Proj}_i(\tilde{t}) = \mathsf{Proj}_i(\tilde{t}_1) = t$ but since $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$ satisfy the Unicity of Transitions (Definition 40) it is impossible because we have, at least, $\tilde{t}$ and $\tilde{t}_1$ with the same transition of L_i since $\mathsf{Proj}_i(\tilde{t}) = \mathsf{Proj}_i(\tilde{t}_1) = t$. □

Now we show that given n LTSIs $L_1, \ldots, L_n$ satisfying causal consistency, and Reversing Preserves Independence (RPI). We establish that Causal Consistency is preserved in the product of these LTSIs, denoted as $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$ if it satisfy the Unicity of Transitions. This theorem demonstrates the compositional nature of Causal Consistency across the product of LTSIs.

Theorem 19 (Compositionality of Causal Consistency)**.** *Given n LTSIs $L_1, \ldots, L_n$ satisfying causal consistency, and RPI. Then Causal Consistency holds in the product of LTSIs $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$ if Unicity of Transitions hold.*

Proof. To prove Causal Consistency in $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$, we need to show that for any two traces r and s, if they are co-initial and co-final, then $r \approx s$ (by Definition 14). We know by Lemma 11 and Causal Consistency of the individual LTSIs that $\forall\ i \in \{1, \ldots, n\}\ \mathsf{Proj}_i(s) \approx \mathsf{Proj}_i(r)$. We fix $j \in \{1, \ldots, n\}$ and we proceed by induction on the number of transformations in $\mathsf{Proj}_j(s) \approx \mathsf{Proj}_j(r)$. We consider a stronger inductive hypothesis saying that if $\mathsf{Proj}_j(s) \approx d$ in L_j then there is r in $L_1 \otimes_{\mathcal{R}} \ldots \otimes_{\mathcal{R}} L_n$ such that $\mathsf{Proj}_j(r) = d$.

Base case: we consider the case when there are 0 transformations. We have that $\mathsf{Proj}_j(s) = d$. We can take $r = s$ and the thesis follows.

Inductive case: we have n swap/delete actions in $\mathsf{Proj}_j(s) \approx d$. Let d' be the trace after one delete or swap operation. Then we have $\mathsf{Proj}_j(s) \approx d' \approx d$. We know that $d' \approx d$ using $n-1$ swap/delete actions. Here we need to analyze the two cases according to the type of action used to prove that $\mathsf{Proj}_j(s) \approx d'$:

Delete: we have $\mathsf{Proj}_j(s) = r_i t \underline{t} r_e$ and $d' = r_i r_e$. By Unicity of Transitions (Definition 40) and Lemma 13 we also have $s = \tilde{r}_i \tilde{t} \underline{\tilde{t}} \tilde{r}_e$ and we can choose $s' = \tilde{r}_i \tilde{r}_e$. We have $\mathsf{Proj}_j(s') = r_i r_e$ and $s \approx s'$ using delete. In the case $d' = r_i t \underline{t} r_e$ and $\mathsf{Proj}_j(s) = r_i r_e$ the reasoning is analogous, but we use Unicity of Transitions (Definition 40) and Lemma 13 to ensure that there is some t such that $\mathsf{Proj}_j(t) = \tilde{t}$.

Swap: we have $\mathsf{Proj}_j(s) = \tilde{r}_i\tilde{t}\tilde{t}'\tilde{r}_e$, $d' = \tilde{r}_i\tilde{t}'\tilde{t}\tilde{r}_e$ and $\tilde{t}\iota_j\tilde{t}'$. We can take $s' = r_i t' t r_e$. By definition of projection $\mathsf{Proj}_j(s') = d'$. We have that tt' and $t't$ are coinitial and cofinal (since they are both obtained from s and s' by removing r_i and r_e). Hence $\forall\, i\ \mathsf{Proj}_i(tt') \approx \mathsf{Proj}_i(t't)$ by Lemma 11 and using Causal Consistency of the individual LTSIs. By Lemma 12 then for all i we have that $\mathsf{Proj}_i(t)\iota_i\mathsf{Proj}_i(t')$. Then we also have that $t\iota t'$ by definition of ι. Hence we have $r \approx s'$ using swap.

By inductive hypothesis we know that there exists r such that $\mathsf{Proj}_j(r) = d$ and $s' \approx r$. By transitivity $s \approx r$.

□

5.2 Case Study: Sequential Consistency Memory Model

In this section, we delve into a sequential consistency memory model, described in [34] (Section 2.2.2), and explore the process of obtaining its reversible version. This memory model maintains the order of memory operations within individual threads but does not strictly enforce the sequence across multiple threads. Synchronization primitives, such as locks or barriers, are commonly used in this memory model to ensure consistent memory operation ordering across all threads.

Initially, we consider two different representations of the memory model: the synchronous product of three LTSs (as described in Section 4.4) and a monolithic LTS that describe the same memory model. The monolithic LTS is described using a monolithic semantics, meaning there is a single rule for each type of action. On the other hand, the synchronous product of LTSs, which we will refer to as the compositional LTS, is a composition of multiple LTSs, where actions are described by synchronization between different LTSs. Following this, we demonstrate that the compositional LTS and the monolithic LTS are bi-similar. Subsequently, we examine two approaches to derive the reversible version of this memory model. The first approach considers the reversible synchronous product of the three LTSs (reverse compositional LTS), while the second approach derives the reversible version of the monolithic LTS through the method described in [33, 44]. Finally, we describe the differences between the two approaches.

Using our theory, starting from n LTSs (Labeled Transition Systems) and for each of them a notion of independence, it is possible to obtain a causal consistent product of LTSs. This is possibile if we can show the Causal Consistency of the reversible LTSs that make up the product (Theorem 19). Causal Consistency of the product is guaranteed if the individual LTSs that form the product satisfy the properties of SP (Square Property), BTI (Backward Transition Independence), and WF (Well-Foundedness) in this case we can apply Theorem 18. Furthermore, our theory shows that if the individual LTSs are causally consistent, then the reverse of the product is equivalent to the product of the reversible LTSs (Theorem 13).

If we consider the product of LTSs described in Section 4.4, with a specific notion of independence for each LTS, in order to apply our theory, we need to show that the properties SP, BTI, and WF hold. The notions of independence for each component are defined as follows:

threads two transitions are independent if the same θ does not appear in both labels.

memory two transitions are independent if the same memory cell r does not appear in both labels.

scheduler two transitions are independent if the same lock l does not appear in both labels.

However, it is not possible to prove the BTI property for these LTSs. For example, in the case of memory, we must show that two backward transitions applied to the same state are not independent. Consider the state (r, v): we can apply both the read and write transitions backward, but these operations are dependent because they refer to the same memory cell r, with one being a write and the other a read. This dependency arises in all three LTSs, meaning that our theory cannot be directly applied.

Thus we need to obtain LTSs with Independence for our threads, scheduler and memory components that either satisfy the three properties SP, BTI and WF, or directly obtain causally consistent reversible variants of these LTSs in order to apply our result on their product.

We have decided to apply the approach in [33] for the reversal of reduction systems to each individual LTS (threads, memory, scheduler), interpreting each labeled transition of these LTSs as a reduction relation.

Using this method, we obtain a notion of independence defined by specific keys for each LTS, where the keys are distinct and not shared among the LTSs.

Definition 43. *Two co-initial transitions* $t : R \xrightarrow{\mu} R'$ *and* $t' : R \xrightarrow{\mu'} R''$ *are independent, written* $t \iota t'$, *if* $\mathsf{key}(\mu) \cap \mathsf{key}(\mu') = \emptyset$.

As part of this case study on the sequential consistent memory model, we will compare the the product of reversible LTSs we obtain by the above method with the monolithic LTS we can obtain using the reversal of reduction system method to the whole LTS obtained joining the individual LTSs (threads, memory, scheduler) specified in Section 4.4. We will then show that:

- The monolithic is indeed bi-similar to the product of components specified in Section 4.4;
- That the product of our reversed components differ from the reversed monolithic LTS, in that the latter introduces causal dependencies which are unwarranted according to the independence relation obtained via the product of our reversed components.

The rest of this section is organized as follows: in Section 5.2.1, we delineate the monolithic LTS and we show the bi-similarity with the compositional LTS described in Section 4.4. In Section 5.2.2 we show the reversible version of the monolithic and of the compositional LTS. Finally, in Section 5.2.3, we show that the independence relation obtained with the reverse compositional LTS is different from the one obtained by the reverse monolithic LTS. Notably, the independence relation derived from the compositional approach better matches the desired one, while the monolithic approach introduces undesired spurious dependencies.

$$\textsc{Alloc}\quad \frac{e \xhookrightarrow{(\theta,\mathsf{ref}(r,v))} e'}{\langle \theta, e\rangle \mid (r \mapsto \circ) \xrightarrow{(\theta,\mathsf{ref}(r,v))} \langle \theta, e'\rangle \mid (r \mapsto v)} \qquad \textsc{Read}\quad \frac{e \xhookrightarrow{(\theta,\mathsf{rd}(r,v))} e'}{\langle \theta, e\rangle \mid (r \mapsto v) \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle \theta, e'\rangle \mid (r \mapsto v)}$$

$$\textsc{Write}\quad \frac{e \xhookrightarrow{(\theta,\mathsf{wr}(r,v))} e'}{\langle \theta, e\rangle \mid (r \mapsto v_{old}) \xrightarrow{(\theta,\mathsf{wr}(r,v))} \langle \theta, e'\rangle \mid (r \mapsto v)} \qquad \textsc{Acquire}\quad \frac{e \xhookrightarrow{(\theta,\mathsf{acq}(l))} e'}{\langle \theta, e\rangle \mid (l \mapsto \circ) \xrightarrow{(\theta,\mathsf{acq}(l))} \langle \theta, e'\rangle \mid (l \mapsto \theta)}$$

$$\textsc{Release}\quad \frac{e \xhookrightarrow{(\theta,\mathsf{rel}(l))} e'}{\langle \theta, e\rangle \mid (l \mapsto \theta) \xrightarrow{(\theta,\mathsf{rel}(l))} \langle \theta, e'\rangle \mid (l \mapsto \circ)} \qquad \textsc{Thread Step}\quad \frac{e \xhookrightarrow{\alpha} e'}{\langle \theta, e\rangle \xrightarrow{(\theta,\alpha)} \langle \theta, e'\rangle}\ \alpha = \mathsf{pure}$$

$$\textsc{Thread Spawn}\quad \frac{e \xhookrightarrow{\mathsf{sp}\ e''} e'; e'' \qquad \theta'\ \ \mathsf{fresh}}{\langle \theta, e\rangle \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \langle \theta, e'\rangle \mid \langle \theta', e''\rangle} \qquad \textsc{Sys Par}\quad \frac{\mathcal{SYS} \xrightarrow{\ell} \mathcal{SYS}'}{\mathcal{SYS} \mid \mathcal{SYS}_1 \xrightarrow{\ell} \mathcal{SYS}' \mid \mathcal{SYS}_1}$$

Figure 5.1: Monolithic LTS Semantics

5.2.1 Monolithic LTS

Now we consider a monolithic LTS where the system is

$$\begin{gathered}\mathcal{SYS} ::= \mathcal{T} \mid \mathcal{M} \mid \mathcal{S} \mid (\mathcal{SYS} \mid \mathcal{SYS}) \\ \mathcal{T} ::= \langle \theta, e\rangle \mid (\mathcal{T} \mid \mathcal{T}) \\ \mathcal{M} ::= (r \mapsto \circ) \mid (r \mapsto v) \mid (\mathcal{M} \mid \mathcal{M}) \\ \mathcal{S} ::= (l \mapsto \circ) \mid (l \mapsto \theta) \mid (\mathcal{S} \mid \mathcal{S})\end{gathered}$$

The rules are detailed in Fig. 5.1, where rules Alloc, Read and Write describe respectively the allocation, reading, and writing of memory cells, rules Acquire and Release describe respectively the acquisition and release of locks, Thread Spawn and Thread Step describe respectively the creation of a new thread and a beta reduction.

Now, we consider the compositional LTS $\mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}$ described in Section 4.4 where we recall the synchronization relation $\mathcal{R}$ (the set of the possible labels):

$$\begin{gathered}\langle (\theta, \alpha), \phi, \phi\rangle, \\ \langle (\theta, \mathsf{ref}(r,v)), (\theta, \mathsf{ref}(r,v)), \phi\rangle, \\ \langle (\theta, \mathsf{rd}(r,v)), (\theta, \mathsf{rd}(r,v)), \phi\rangle, \\ \langle (\theta, \mathsf{wr}(r,v)), (\theta, \mathsf{wr}(r,v)), \phi\rangle, \\ \langle (\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l))\rangle, \\ \langle (\theta, \mathsf{rel}(l)), \phi, (\theta, \mathsf{rel}(l))\rangle\end{gathered}$$

where $\alpha = \mathsf{sp}(\theta'), \mathsf{pure}$.

The shape of Sync for the compositional LTS in the case of a read action is the following:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{rd}(r,v))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \dfrac{\dfrac{}{(r \mapsto v) \xrightarrow{(\theta,\mathsf{rd}(r,v))} (r \mapsto v)}}{(r \mapsto v) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{rd}(r,v))} (r \mapsto v) \mid \mathcal{M}} \quad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \quad \langle((\theta,\mathsf{rd}(r,v)),(\theta,\mathsf{rd}(r,v)),\phi)\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\mathsf{rd}(r,v)),(\theta,\mathsf{rd}(r,v)),\phi)} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

In order to show the bi-simulation (see Definition 31) between the monolithic LTS $(\mathcal{T} \mid \mathcal{M} \mid \mathcal{S})$ and the compositional LTS $(\mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S})$ we define a function $\varphi(\cdot)$ to obtain a relation between the labels of the compositional LTS and the labels of the monolithic LTS.

Definition 44. *The function $\varphi(\cdot)$ is defined over labels:*

$$\begin{gathered}
\varphi(((\theta,\alpha),\phi,\phi)) = (\theta,\alpha),\\
\varphi(((\theta,\mathsf{ref}(r,v)),(\theta,\mathsf{ref}(r,v)),\phi)) = (\theta,\mathsf{ref}(r,v)),\\
\varphi(((\theta,\mathsf{rd}(r,v)),(\theta,\mathsf{rd}(r,v)),\phi)) = (\theta,\mathsf{rd}(r,v)),\\
\varphi(((\theta,\mathsf{wr}(r,v)),(\theta,\mathsf{wr}(r,v)),\phi)) = (\theta,\mathsf{wr}(r,v)),\\
\varphi(((\theta,\mathsf{acq}(l)),\phi,(\theta,\mathsf{acq}(l)))) = (\theta,\mathsf{acq}(l)),\\
\varphi(((\theta,\mathsf{rel}(l)),\phi,(\theta,\mathsf{rel}(l)))) = (\theta,\mathsf{rel}(l))
\end{gathered}$$

where $\alpha = \mathsf{sp}(\theta'),\mathsf{pure}$.

Theorem 20 (Bi-Simulation)**.** $(\mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}) \overset{\mathbf{lab}}{\sim} (\mathcal{T}_1 \mid \mathcal{M}_1 \mid \mathcal{S}_1)$.

Proof. Firstly, we need to define the relation between the states of the compositional LTS and of the monolithic LTS, we define the relation as a set of couple $\mathcal{RL} = \{\langle(\mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}), (\mathcal{T}_1 \mid \mathcal{M}_1 \mid \mathcal{S}_1)\rangle$ such that $\mathcal{T} = \mathcal{T}_1, \mathcal{M} = \mathcal{M}_1, \mathcal{S} = \mathcal{S}_1\}$. Now we show by case that for each transition done by the compositional LTS (with the transformed label) then the monolithic LTS can do the same transition. Formally we show that for each label ℓ such that $\mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{\ell} \mathcal{T}' \otimes_{\mathcal{R}} \mathcal{M}' \otimes_{\mathcal{R}} \mathcal{S}'$ then $\mathcal{T} \mid \mathcal{M} \mid \mathcal{S} \xrightarrow{\varphi(\ell)} \mathcal{T}' \mid \mathcal{M}' \mid \mathcal{S}'$. Now we proceed by case analysis on the label:

- We consider the case where $\varphi(((\theta,\mathsf{rd}(r,v)),(\theta,\mathsf{rd}(r,v)),\phi)) = (\theta,\mathsf{rd}(r,v))$, then in the compositional LTS we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{rd}(r,v))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \dfrac{\dfrac{}{(r \mapsto v) \xrightarrow{(\theta,\mathsf{rd}(r,v))} (r \mapsto v)}}{(r \mapsto v) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{rd}(r,v))} (r \mapsto v) \mid \mathcal{M}} \quad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \quad \langle((\theta,\mathsf{rd}(r,v)),(\theta,\mathsf{rd}(r,v)),\phi)\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\mathsf{rd}(r,v)),(\theta,\mathsf{rd}(r,v)),\phi)} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

in the monolithic LTS we have the following configuration $\mathcal{T} \mid \langle \theta, e \rangle \mid \mathcal{M} \mid (r \mapsto v) \mid \mathcal{S}$ obtained by $\mathcal{RL}$, and we consider the label $(\theta, \mathsf{rd}(r, v))$. Then, we obtain:

$$\textsc{Par}\ \frac{\textsc{Read}\ \dfrac{e \xhookrightarrow{(\theta,\mathsf{rd}(r,v))} e'}{\langle \theta, e \rangle \mid (r \mapsto v) \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle \theta, e' \rangle \mid (r \mapsto v)}}{\mathcal{T} \mid \langle \theta, e \rangle \mid \mathcal{M} \mid (r \mapsto v) \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle \theta, e' \rangle \mid \mathcal{T} \mid (r \mapsto v) \mid \mathcal{M} \mid \mathcal{S}}$$

we can easily see that the final states are still related.

Now we consider the case starting from the monolithic LTS and we consider then $\mathcal{RL}^{-1}$ and φ^{-1}. Formally we show that for each label ℓ such that $\mathcal{T} \mid \mathcal{M} \mid \mathcal{S} \xrightarrow{\ell} \mathcal{T}' \mid \mathcal{M}' \mid \mathcal{S}'$ then $\mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{\varphi^{-1}(\ell)} \mathcal{T}' \otimes_{\mathcal{R}} \mathcal{M}' \otimes_{\mathcal{R}} \mathcal{S}'$. We proceed by case analysis on the label:

- We consider the case where $\varphi^{-1}((\theta, \mathsf{rd}(r, v))) = ((\theta, \mathsf{rd}(r, v)), (\theta, \mathsf{rd}(r, v)), \phi)$, then in monolithic LTS we have that:

$$\textsc{Par}\ \frac{\textsc{Read}\ \dfrac{e \xhookrightarrow{(\theta,\mathsf{rd}(r,v))} e'}{\langle \theta, e \rangle \mid (r \mapsto v) \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle \theta, e' \rangle \mid (r \mapsto v)}}{\mathcal{T} \mid \langle \theta, e \rangle \mid \mathcal{M} \mid (r \mapsto v) \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle \theta, e' \rangle \mid \mathcal{T} \mid (r \mapsto v) \mid \mathcal{M} \mid \mathcal{S}}$$

in the compositional LTS we have the following configuration $\mathcal{T} \mid \langle \theta, e \rangle \otimes_{\mathcal{R}} \mathcal{M} \mid (r \mapsto v) \otimes_{\mathcal{R}} \mathcal{S}$ obtained by $\mathcal{RL}^{-1}$, and we consider the label $((\theta, \mathsf{rd}(r, v)), (\theta, \mathsf{rd}(r, v)), \phi)$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{rd}(r,v))} e'}{\langle \theta, e \rangle \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle \theta, e' \rangle}}{\langle \theta, e \rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle \theta, e' \rangle \mid \mathcal{T}} \qquad \dfrac{\dfrac{}{(r \mapsto v) \xrightarrow{(\theta,\mathsf{rd}(r,v))} (r \mapsto v)}}{(r \mapsto v) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{rd}(r,v))} (r \mapsto v) \mid \mathcal{M}} \quad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \qquad \langle ((\theta, \mathsf{rd}(r, v)), (\theta, \mathsf{rd}(r, v)), \phi) \rangle \in \mathcal{R}}{\langle \theta, e \rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\mathsf{rd}(r,v)),(\theta,\mathsf{rd}(r,v)),\phi)} \langle \theta, e' \rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

we can easily see that the final states are still related.

All other cases are similar and are shown in Appendix C.1. □

By applying the method described in [33], we obtain labels in the form of memories ($[k : old, k' : \bullet]$) as shown in Fig. C.3 and Fig. C.4. However, we have opted to use more convenient labels for our purposes. This does not change the result, as each transition with the new labels can be associated with a transition using memory labels, and vice versa. Now we will show an example, with all other cases being analogous. We consider now a transition from the LTS with memory labels and how it corresponds to a transition in the monolithic LTS:

$$k_1 : \langle \theta, e \rangle \mid k_2 : (r \mapsto \circ) \xrightarrow{[k_1:\langle \theta,e\rangle \mid k_2:(r\mapsto\circ); k_1':\bullet_1 \mid k_2':\bullet_2]} k_1' : \langle \theta, e' \rangle \mid k_2' : (r \mapsto v) \mid [k_1 : \langle \theta, e \rangle \mid k_2 : (r \mapsto \circ); k_1' : \bullet_1 \mid k_2' : \bullet_2]$$

From this transition we can always understand that the label in the monolithic LTS is $(\theta, \mathsf{ref}(r, v))$ because in the initial state we have $(r \mapsto \circ)$ and in the final one $(r \mapsto v)$ and from here

READ

$$\frac{e \xhookrightarrow{(\theta,\mathsf{rd}(r,v))} e' \qquad k_1', k_2' \;\; fresh}{k_1 : \langle\theta, e\rangle \mid k_2 : (r \mapsto v) \xrightarrow{(\theta,\mathsf{rd}(r,v))} k_1' : \langle\theta, e'\rangle \mid k_2' : (r \mapsto v) \mid [k_1 : \langle\theta, e\rangle \mid k_2 : (r \mapsto v); k_1' : \bullet_1 \mid k_2' : \bullet_2]}$$

$\overline{\text{READ}}$

$$k_1' : \langle\theta, e'\rangle \mid k_2' : (r \mapsto v) \mid [k_1 : \langle\theta, e\rangle \mid k_2 : (r \mapsto v); k_1' : \bullet_1 \mid k_2' : \bullet_2] \overset{(\theta,\mathsf{rd}(r,v))}{\rightsquigarrow} k_1 : \langle\theta, e\rangle \mid k_2 : (r \mapsto v)$$

Figure 5.2: Forward and backward READ reversible rules of monolithic LTS

we understand both the operation and the memory cell r and the value v, instead θ can be understood because the process has θ as id. We can apply this reasoning to all rules. We can see how from a transition of the monolithic LTS we obtain a transition of the LTS with memories. This is always possible because we have the memory inside each transition.

5.2.2 Reversing Monolithic and Compositional LTSs

Now that we have shown the relationship between the monolithic LTS and the compositional LTS, let us consider the reversible compositional LTS and the reversible monolithic LTS. They are not bi-similar because we also consider the memories used by the approach we have adopted. In the case of the compositional LTS, we have one memory for each LTS involved in the action instead in the monolithic LTS we have one memory for each action. Indeed, as we will see, with a synchronization move, we obtain one memory for each involved LTS. For example, for the read operation, we obtain two memories.

In Fig. 5.2 we show the forward and backward version of rule READ obtained through [33, 44] for the monolithic LTS. It is possible to see the other rules in Fig. C.1 and Fig. C.2 in Appendix C.2.

Now, let us consider the compositional LTSs $\mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}$. To obtain their reversible form, we can use Theorem 13 to consider the product of reversible LTSs while maintaining the same synchronization relation, resulting in $\mathcal{T}^{\mathcal{RV}} \otimes_{\mathcal{R}} \mathcal{M}^{\mathcal{RV}} \otimes_{\mathcal{R}} \mathcal{S}^{\mathcal{RV}}$. The reversible compositional LTS will be built from the LTSs where we apply the method described in [33, 44] to reverse them. In this way, the LTSs that make up our reversible compositional LTS will satisfy the following properties: Loop Lemma, SP, BTI, and WF.

Now we show the new form of the reversible compositional LTS $\mathcal{T}^{\mathcal{RV}} \otimes_{\mathcal{R}} \mathcal{M}^{\mathcal{RV}} \otimes_{\mathcal{R}} \mathcal{S}^{\mathcal{RV}}$. The syntax format of the thread semantics is $\mathcal{T}^{\mathcal{RV}} ::= \kappa : \langle\theta, e\rangle \mid (\mathcal{T}^{\mathcal{R}}\mathcal{V} \mid \mathcal{T}^{\mathcal{R}}\mathcal{V})$. In Figures 5.3 and 5.4 we depict respectively the forward and backward semantics obtained with the framework described in the article [33, 44]. Rule THREAD STEP explains the evaluation of an action, different from the spawn action (sp), as indicated by the condition $\alpha \neq \mathsf{sp}$. THREAD SPAWN rule creates a new thread with a new identifier θ'.

The syntax format of the memory semantics is $\mathcal{M}^{\mathcal{RV}} ::= \kappa : (r \mapsto \circ) \mid \kappa : (r \mapsto v) \mid (\mathcal{M}^{\mathcal{RV}} \mid \mathcal{M}^{\mathcal{RV}})$. The forward and backward semantics are depicted in Figures 5.5 and 5.6, both generated with the approach described in [33, 44]. The memory semantics is defined by the following rules: ALLOC describes the allocation of memory for a new variable, READ governs the process of

$$\textsc{Thread Step}\quad \frac{e \overset{\alpha}{\hookrightarrow} e' \qquad k_1 \;\; fresh}{k : \langle \theta, e \rangle \xrightarrow{(\theta,\alpha)} k_1 : \langle \theta, e' \rangle \mid [k : \langle \theta, e \rangle; k_1 : \bullet_1]}\ \alpha \neq \mathsf{sp}$$

$$\textsc{Thread Spawn}\quad \frac{e \xrightarrow{\mathsf{sp}\ e''} e'; e'' \qquad \theta' \;\; \mathsf{fresh} \qquad k_1, k_2 \;\; fresh}{k : \langle \theta, e \rangle \xrightarrow{(\theta,\mathsf{sp}(\theta'))} k_1 : \langle \theta, e' \rangle \mid k_2 : \langle \theta', e'' \rangle \mid [k : \langle \theta, e \rangle; k_1 : \bullet_1 \mid k_2 : \bullet_2]}$$

Figure 5.3: Forward reversible semantics of Threads

$$\frac{}{\textsc{Thread Step}}\quad k_1 : \langle \theta, e' \rangle \mid [k : \langle \theta, e \rangle; k_1 : \bullet_1] \overset{(\theta,\alpha)}{\rightsquigarrow} k : \langle \theta, e \rangle \;\; \text{WITH } \alpha \neq \mathsf{sp}$$

$$\frac{}{\textsc{Thread Spawn}}\quad k_1 : \langle \theta, e' \rangle \mid k_2 : \langle \theta', e'' \rangle \mid [k : \langle \theta, e \rangle; k_1 : \bullet_1 \mid k_2 : \bullet_2] \overset{(\theta,\mathsf{sp}(\theta'))}{\rightsquigarrow} k : \langle \theta, e \rangle$$

Figure 5.4: Backward reversible semantics of Threads

reading a value from memory, and Write specifies how a value is written to memory.

The syntax format of the scheduler semantics is $\mathcal{S}^{\mathcal{RV}} ::= \kappa : (l \mapsto \circ) \mid \kappa : (l \mapsto \theta) \mid (\mathcal{S}^{\mathcal{RV}} \mid \mathcal{S}^{\mathcal{RV}})$. Figure 5.7 depicts the forward reversible semantics and Figure 5.8 represents the backward semantics, both generated with the approach described in [33, 44]. The scheduler semantics defined by rule Acquire, which describes the process of acquiring a lock, and by rule Release, which governs the release of a previously acquired lock.

Sync rules to go forward are analogous to those shown previously with the addition of

$$\textsc{Alloc}\quad \frac{k_1 \;\; fresh}{k : (r \mapsto \circ) \xrightarrow{(\theta,\mathsf{ref}(r,v))} k_1 : (r \mapsto v) \mid [k : (r \mapsto \circ); k_1 : \bullet_1]}$$

$$\textsc{Read}\quad \frac{k_1 \;\; fresh}{k : (r \mapsto v) \xrightarrow{(\theta,\mathsf{rd}(r,v))} k_1 : (r \mapsto v) \mid [k : (r \mapsto v); k_1 : \bullet_1]}$$

$$\textsc{Write}\quad \frac{k_1 \;\; fresh}{k : (r \mapsto v_{old}) \xrightarrow{(\theta,\mathsf{wr}(r,v))} k_1 : (r \mapsto v) \mid [k : (r \mapsto v_{old}); k_1 : \bullet_1]}$$

Figure 5.5: Forward reversible semantics of Memory

$$\frac{}{k_1 : (r \mapsto v) \mid [k : (r \mapsto \circ); k_1 : \bullet_1] \overset{(\theta,\mathsf{ref}(r,v))}{\rightsquigarrow} k : (r \mapsto \circ)}\ \textsc{Alloc}$$

$$\frac{}{k_1 : (r \mapsto v) \mid [k : (r \mapsto v); k_1 : \bullet_1] \overset{(\theta,\mathsf{rd}(r,v))}{\rightsquigarrow} k : (r \mapsto v)}\ \textsc{Read}$$

$$\frac{}{k_1 : (r \mapsto v) \mid [k : (r \mapsto v_{old}); k_1 : \bullet_1] \overset{(\theta,\mathsf{wr}(r,v))}{\rightsquigarrow} k : (r \mapsto v_{old})}\ \textsc{Write}$$

Figure 5.6: Backward reversible semantics of Memory

$$\frac{k_1 \quad fresh}{k : (l \mapsto \circ) \xrightarrow{(\theta,\mathsf{acq}(l))} k_1 : (l \mapsto \theta) \mid [k : (l \mapsto \circ); k_1 : \bullet_1]}\ \textsc{Acquire}$$

$$\frac{k_1 \quad fresh}{k : (l \mapsto \theta) \xrightarrow{(\theta,\mathsf{rel}(l))} k_1 : (l \mapsto \circ) \mid [k : (l \mapsto \theta); k_1 : \bullet_1]}\ \textsc{Release}$$

Figure 5.7: Forward reversible semantics of Scheduler

$$\frac{}{k_1 : (l \mapsto \theta) \mid [k : (l \mapsto \circ); k_1 : \bullet_1] \overset{(\theta,\mathsf{acq}(l))}{\rightsquigarrow} k : (l \mapsto \circ)}\ \textsc{Acquire}$$

$$\frac{}{k_1 : (l \mapsto \circ) \mid [k : (l \mapsto \theta); k_1 : \bullet_1] \overset{(\theta,\mathsf{rel}(l))}{\rightsquigarrow} k : (l \mapsto \theta)}\ \textsc{Release}$$

Figure 5.8: Backward reversible semantics of Scheduler

memories (shown in red). We exhibit the example for the application of the read operation.

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xleftarrow{(\theta,\mathsf{rd}(r,v))} e' \qquad k_1 \quad fresh}{k : \langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{rd}(r,v))} k_1 : \langle\theta, e'\rangle \mid [k : \langle\theta, e\rangle; k_1 : \bullet_1]}}{k : \langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{rd}(r,v))} k_1 : \langle\theta, e'\rangle \mid [k : \langle\theta, e\rangle; k_1 : \bullet_1] \mid \mathcal{T}} \qquad \frac{\dfrac{k'_1 \quad fresh}{k' : (r \mapsto v) \xrightarrow{(\theta,\mathsf{rd}(r,v))} k'_1 : (r \mapsto v) \mid [k' : (r \mapsto v); k'_1 : \bullet_1]}}{k' : (r \mapsto v) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{rd}(r,v))} k'_1 : (r \mapsto v) \mid [k' : (r \mapsto v); k'_1 : \bullet_1] \mid \mathcal{M}}}{k : \langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} k' : (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\mathsf{rd}(r,v)),(\theta,\mathsf{rd}(r,v)),\phi)} k_1 : \langle\theta, e'\rangle \mid [k : \langle\theta, e\rangle; k_1 : \bullet_1] \mid \mathcal{T} \otimes_{\mathcal{R}} k'_1 : (r \mapsto v) \mid [k' : (r \mapsto v); k'_1 : \bullet_1] \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

$$\mathcal{S} \xrightarrow{\phi} \mathcal{S} \qquad \langle((\theta, \mathsf{rd}(r,v)), (\theta, \mathsf{rd}(r,v)), \phi)\rangle \in \mathcal{R}$$

While to go backwards, we present the rule corresponding to the undo of read action, the other

cases are similar.

SYNC

$$\frac{k_1 : \langle \theta, e' \rangle \mid [k : \langle \theta, e \rangle; k_1 : \bullet_1] \xrightarrow{(\theta, \mathsf{rd}(r,v))} k : \langle \theta, e \rangle \qquad k'_1 : (r \mapsto v) \mid [k' : (r \mapsto v); k'_1 : \bullet_1] \xrightarrow{(\theta, \mathsf{rd}(r,v))} k' : (r \mapsto v)}{k_1 : \langle \theta, e' \rangle \mid [k : \langle \theta, e \rangle; k_1 : \bullet_1] \mid \mathcal{T} \xrightarrow{(\theta, \mathsf{rd}(r,v))} k : \langle \theta, e \rangle \mid \mathcal{T} \quad k'_1 : (r \mapsto v) \mid [k' : (r \mapsto v); k'_1 : \bullet_1] \mid \mathcal{M} \xrightarrow{(\theta, \mathsf{rd}(r,v))} k' : (r \mapsto v) \mid \mathcal{M} \quad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \quad \langle ((\theta, \mathsf{rd}(r,v)), (\theta, \mathsf{rd}(r,v)), \phi) \rangle \in \mathcal{R}}{k_1 : \langle \theta, e' \rangle \mid [k : \langle \theta, e \rangle; k_1 : \bullet_1] \mid \mathcal{T} \otimes_{\mathcal{R}} k'_1 : (r \mapsto v) \mid [k' : (r \mapsto v); k'_1 : \bullet_1] \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta, \mathsf{rd}(r,v)), (\theta, \mathsf{rd}(r,v)), \phi)} k : \langle \theta, e \rangle \mid \mathcal{T} \otimes_{\mathcal{R}} k' : (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

If we compare these rules with those in Fig. 5.2 (which illustrates the read action for the reverse monolithic LTS), a significant distinction emerges. In the rule for the reverse monolithic LTS, the keys of the memory ($\mathcal{M}$) are directly associated with those of the thread that performed the memory cell read. However, this association does not exist in the compositional LTS. We have instead two distinct sets of keys, one for the threads and one for the memory, which are not linked between them. This distinction is crucial because it enables us to synchronize this operation with other read operations of the same cell resulting in the same value. This capability becomes evident in the example discussed later, where it allows us to leverage the automatic approach described in [33, 44]. Previously, this was not feasible because the approach erroneously identified these read operations as dependencies, despite our intention for them not to be.

After showing how the reversible compositional LTS is formed, in order to show that it is causally consistent, we must show that it satisfies the Unicity of Transitions (Definition 40).

Lemma 14. *In the product of LTSs $\mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}$ the Unicity of Transition holds.*

Proof. Each LTS ($\mathcal{T}$, $\mathcal{M}$, $\mathcal{S}$) has only one possible transition for each label. Additionally, in the synchronization relation $\mathcal{R}$, there is only one possible synchronization for each label of the various LTSs. Therefore, in the product of LTSs $\mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}$, the Unicity of Transitions is ensured. □

Example 12. Consider the label $((\theta, \mathsf{rd}(r,v)), (\theta, \mathsf{rd}(r,v)), \phi)$ in $\mathcal{R}$. It can be observed that both in $\mathcal{T}$ and in $\mathcal{M}$ only one transition is possible with the label $(\theta, \mathsf{rd}(r,v))$ in the form $\langle \theta, e \rangle \xrightarrow{(\theta, \mathsf{rd}(r,v))} \langle \theta, e' \rangle$ (in $\mathcal{T}$) and $(r \mapsto v) \xrightarrow{(\theta, \mathsf{rd}(r,v))} (r \mapsto v)$ (in $\mathcal{M}$). ◊

Thanks to Lemma 14 and since in the individual LTSs the proprieties Loop Lemma, SP, BTI and WF hold since we have used the approach [33] described in Section 2.4, for these reason it is possible to apply the approach described in Section 5.1.1. Then we can stand that the reverse compositional LTS is causal consistent thanks to Theorem 18.

5.2.3 Comparison between Reversible Monolithic and Compositional LTSs

For sake of simplicity, we consider a system with two threads (θ_1, θ_2) that needs to read from the memory cell x, which has already been initialized. We also have a single lock l that has not been acquired yet. So, we have $T = k_{t1} : \langle \theta_1, !x \rangle \mid k_{t2} : \langle \theta_2, !x \rangle$, $M = k_{m1} : (x \mapsto 0)$, and $S = k_{s1} : (l \mapsto \circ)$. As a result, considering the reverse compositional LTS we show the projection of the three LTSs of the trace corresponding to the read of x by θ_1, followed by the read of x by θ_2, and finally the undo of the read operation by θ_1. In red, we show which element is modified in the transition.

$$
\begin{array}{lll}
\ldots & \rightarrow & k_{t1} : \langle \theta_1, !x \rangle \mid k_{t2} : \langle \theta_2, !x \rangle \\
& \xrightarrow{(\theta_1, \mathsf{rd}(x,0))} & k_{t12} : \langle \theta_1, 0 \rangle \mid [k_{t1} : \langle \theta_1, !x \rangle; k_{t12} : \bullet_{t12}] \mid k_{t2} : \langle \theta_2, !x \rangle \\
& \xrightarrow{(\theta_2, \mathsf{rd}(x,0))} & k_{t12} : \langle \theta_1, 0 \rangle \mid [k_{t1} : \langle \theta_1, !x \rangle; k_{t12} : \bullet_{t12}] \mid k_{t21} : \langle \theta_2, 0 \rangle \mid [k_{t2} : \langle \theta_2, !x \rangle; k_{t21} : \bullet_{t21}] \\
& \rightsquigarrow^{(\theta_1, \mathsf{rd}(x,0))} & k_{t1} : \langle \theta_1, !x \rangle \mid k_{t21} : \langle \theta_2, 0 \rangle \mid [k_{t2} : \langle \theta_2, !x \rangle; k_{t21} : \bullet_{t21}] \\
& \rightarrow & \ldots
\end{array}
$$

$$
\begin{array}{lll}
\ldots & \rightarrow & k_{m1} : (x \mapsto 0) \\
& \xrightarrow{(\theta_1, \mathsf{rd}(x,0))} & k_{m12} : (x \mapsto 0) \mid [k_{m1} : (x \mapsto 0); k_{m12} : \bullet_{m12}] \\
& \xrightarrow{(\theta_2, \mathsf{rd}(x,0))} & k_{m13} : (x \mapsto 0) \mid [k_{m12} : (x \mapsto 0); k_{m13} : \bullet_{m13}] \mid [k_{m1} : (x \mapsto 0); k_{m12} : \bullet_{m12}] \\
& \rightsquigarrow^{(\theta_1, \mathsf{rd}(x,0))} & k_{m12} : (x \mapsto 0) \mid [k_{m1} : (x \mapsto 0); k_{m12} : \bullet_{m12}] \\
& \rightarrow & \ldots
\end{array}
$$

$$
\begin{array}{lll}
\ldots & \rightarrow & k_{s1} : (l \mapsto \circ) \\
& \xrightarrow{\phi} & k_{s1} : (l \mapsto \circ) \\
& \xrightarrow{\phi} & k_{s1} : (l \mapsto \circ) \\
& \rightsquigarrow^{\phi} & k_{s1} : (l \mapsto \circ) \\
& \rightarrow & \ldots
\end{array}
$$

Additionally, we examine the trace if we considered the same actions performed by the monolithic LTS.

$$
\begin{array}{lll}
\ldots & \rightarrow & k_{t1} : \langle \theta_1, !x \rangle \mid k_{t2} : \langle \theta_2, !x \rangle \mid k_{m1} : (x \mapsto 0) \mid k_{s1} : (l \mapsto \circ) \\
& \xrightarrow{(\theta_1, \mathsf{rd}(x,0))} & k_{t12} : \langle \theta_1, 0 \rangle \mid k_{m12} : (x \mapsto 0) \mid [k_{t1} : \langle \theta_1, !x \rangle \mid k_{m1} : (x \mapsto 0); k_{t12} : \bullet_{t12} \mid k_{m12} : \bullet_{m12}] \\
& & \mid k_{t2} : \langle \theta_2, !x \rangle \mid k_{s1} : (l \mapsto \circ) \\
& \xrightarrow{(\theta_2, \mathsf{rd}(x,0))} & k_{t21} : \langle \theta_2, 0 \rangle \mid k_{m13} : (x \mapsto 0) \mid [k_{t2} : \langle \theta_2, !x \rangle \mid k_{m12} : (x \mapsto 0); k_{t21} : \bullet_{t21} \mid k_{m13} : \bullet_{m13}] \\
& & \mid k_{t12} : \langle \theta_1, 0 \rangle \mid [k_{t1} : \langle \theta_1, !x \rangle \mid k_{m1} : (x \mapsto 0); k_{t12} : \bullet_{t12} \mid k_{m12} : \bullet_{m12}] \mid k_{s1} : (l \mapsto \circ) \\
& \not{\rightsquigarrow}^{(\theta_1, \mathsf{rd}(x,0))} &
\end{array}
$$

As we can notice in the trace of the monolithic LTS it is not possible to execute $\rightsquigarrow^{(\theta_1, \mathsf{rd}(x,0))}$, in fact in the trace we have $\not{\rightsquigarrow}^{(\theta_1, \mathsf{rd}(x,0))}$. This happens because with only one LTS we link the process and the memory and then we must use the same key to undo this operation. Instead with compositional LTS we do not link the key of the process with the key of the memory (we have different memories for each LTS) and then it is possible to undo this action.

Chapter 6

Conclusions and Future Works

6.1 Conclusions

The aim of this thesis has been to investigate causal consistent reversibility in memory models.

We have extended CauDEr and the underlying reversible semantics of Erlang to support imperative primitives used to associate names to pids. This required to distinguish write accesses from read accesses to the map, since the latter commute while the former do not. Also, the interplay between delete and read operations required us to keep track of removed tuples. Notably, a similar approach needs to be used to define the reversible semantics of imperative languages, such as C or Java. We have also extended the definitions of rollback and replay operators, as well as the underlying theory.

This extension highlights the difficulties when you want reach the Causal Consistency in a programming language, for this reason we extend the Effinger's framework, in this way we have a model that can describe more memory models. We described our framework using the synchronous product of three LTSs, each representing different components: thread semantics, memory semantics, and scheduler semantics. We divided the labels of the LTSs into groups and defined a synchronization relation to manage the interactions among these components. Our framework successfully described some memory models, including sequential consistency, write buffer (PSO), and transactions memory models, demonstrating their bi-similarity with the corresponding models in the literature.

We then presented a new compositional theory to show how to reverse a product of LTSs. We demonstrated that the synchronous product of these LTSs achieves causal consistency, given certain properties of the individual LTSs. We applied this theory to a monolithic LTS with a sequential consistency memory model, using an automatic method to make it reversible. By comparing this to the sequential consistency memory model obtained through our framework, we illustrated that our compositional theory better captured the desired dependencies, avoiding some of the unwanted dependencies captured by the automatic method.

In conclusion, our framework offers a tool for representing memory models, and our compositional theory shows how to obtain Causal Consistency in synchronous product of LTSs, then also in systems described with our framework.

6.2 Future Works

Concerning future work, the approach used to extend CauDEr, as well as the theory in [35] on which we rely to prove properties, defines independence as a binary relation on transitions. We plan to extend this approach in future work by defining independence as a binary relation on sequences of transitions, since we have found cases where single transitions do not commute, while sequences can. For instance, in the exertion of CauDEr a registered() does not commute with either register(a, _) or unregister(a), but it can commute with their composition since the set of registered tuples is the same before and after. Notably, covering this case would require to extend the theory in [35] as well.

This extension would also necessitate an update to our compositional theory. Specifically, the condition of Unicity of Transitions could be deleted. By removing the requirement for a binary relation, an action could commute with a sequence of actions, including the idle action. Consequently, Example 10 would no longer be valid.

Other possible extensions include supporting further Erlang constructs, the first target would be the global map shared by all the nodes of an Erlang application. Given that our replay technique makes no assumption about distribution, the approach should work in this context as well. More difficult target includes Erlang code hot-swap.

A possible future work involves describing the semantics of Erlang within our framework. To achieve this, it will be necessary to understand how to implement the Erlang map and messages in our framework. One possible division would be to manage the map through the memory component and the messages through the scheduler component. Subsequently, it would be interesting to make the obtained semantics reversible using our theory and compare the results with the reversible semantics we have previously created.

Another potential study, still in the same direction, would be to apply our theory to the strong and weak transactions memory models. This would allow us to understand the differences in the reversible models since the only rule that changes is in the scheduler semantics.

Regarding transactions, another possible work could be to reverse the weak memory model with optimistic rollback. In this case, it would be interesting to see if it is possible to achieve the same results without implementing the rollback operator as currently done, but by using reversibility to go back in case of conflicts. This approach would explore reversibility in the context of transactions and could simplify the existing model by implementing the rollback operator through reversibility.

Concerning efficiency, we have disregarded this aspect till now, hence large improvements are possible. E.g., at each step we store the whole state of the process, but states in consecutive steps are closely related, hence one could store far less information, at the cost of doing some more computation to reconstruct the process state.

Bibliography

[1] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM J. Res. Dev.*, vol. 5, no. 3, pp. 183–191, 1961.

[2] C. D. Carothers, K. S. Perumalla, and R. Fujimoto, "Efficient optimistic parallel simulations using reverse computation," *ACM Trans. Model. Comput. Simul.*, vol. 9, no. 3, pp. 224–253, 1999.

[3] J. S. Laursen, U. P. Schultz, and L. Ellekilde, "Automatic error recovery in robot assembly operations using reverse execution," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*, pp. 1785–1792, IEEE, 2015.

[4] I. Phillips, I. Ulidowski, and S. Yuen, "A reversible process calculus and the modelling of the ERK signalling pathway," in *Reversible Computation, 4th International Workshop, RC 2012, Copenhagen, Denmark, July 2-3, 2012. Revised Papers* (R. Glück and T. Yokoyama, eds.), vol. 7581 of *Lecture Notes in Computer Science*, pp. 218–232, Springer, 2012.

[5] CauDEr, "CauDEr repository." Available at `https://github.com/mistupv/cauder`, 2022.

[6] C. Lutz and H. Derby, "Janus: a time-reversible language," *Letter to R. Landauer*, vol. 2, 1986.

[7] T. Haulund, "Design and implementation of a reversible object-oriented programming language," *CoRR*, vol. abs/1707.07845, 2017.

[8] U. P. Schultz and H. B. Axelsen, "Elements of a reversible object-oriented language - work-in-progress report," in *Reversible Computation - 8th International Conference, RC 2016, Bologna, Italy, July 7-8, 2016, Proceedings* (S. J. Devitt and I. Lanese, eds.), vol. 9720 of *Lecture Notes in Computer Science*, pp. 153–159, Springer, 2016.

[9] U. P. Schultz, "Reversible object-oriented programming with region-based memory management - work-in-progress report," in *Reversible Computation - 10th International Conference, RC 2018, Leicester, UK, September 12-14, 2018, Proceedings* (J. Kari and I. Ulidowski, eds.), vol. 11106 of *Lecture Notes in Computer Science*, pp. 322–328, Springer, 2018.

[10] T. Yokoyama, H. B. Axelsen, and R. Glück, "Towards a reversible functional language," in *Reversible Computation - Third International Workshop, RC 2011, Gent, Belgium, July*

4-5, 2011. Revised Papers (A. D. Vos and R. Wille, eds.), vol. 7165 of *Lecture Notes in Computer Science*, pp. 14–29, Springer, 2011.

[11] P. A. H. Jacobsen, R. Kaarsgaard, and M. K. Thomsen, "\mathsf corefun : A typed functional reversible core language," in *Reversible Computation - 10th International Conference, RC 2018, Leicester, UK, September 12-14, 2018, Proceedings* (J. Kari and I. Ulidowski, eds.), vol. 11106 of *Lecture Notes in Computer Science*, pp. 304–321, Springer, 2018.

[12] C. H. Bennett, "Logical Reversibility of Computation," *IBM Journal of Research and Development*, vol. 17, no. 6, pp. 525–532, 1973.

[13] V. Danos and J. Krivine, "Reversible communicating systems," in *CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK, August 31 - September 3, 2004, Proceedings* (P. Gardner and N. Yoshida, eds.), vol. 3170 of *Lecture Notes in Computer Science*, pp. 292–307, Springer, 2004.

[14] C. Hewitt, P. B. Bishop, and R. Steiger, "A universal modular ACTOR formalism for artificial intelligence," in *Proceedings of the 3rd International Joint Conference on Artificial Intelligence. Standford, CA, USA, August 20-23, 1973* (N. J. Nilsson, ed.), pp. 235–245, William Kaufmann, 1973.

[15] Erlang, "Erlang/OTP 24.1.5." Available at `https://www.erlang.org/doc/index.html`, 1986.

[16] M. Abadi, A. Birrell, T. Harris, and M. Isard, "Semantics of transactional memory and automatic mutual exclusion," *ACM Trans. Program. Lang. Syst.*, vol. 33, no. 1, pp. 2:1–2:50, 2011.

[17] N. Shavit and D. Touitou, "Software transactional memory," *Distributed Comput.*, vol. 10, no. 2, pp. 99–116, 1997.

[18] D. Gelernter, "Generative communication in linda," *ACM Trans. Program. Lang. Syst.*, vol. 7, no. 1, pp. 80–112, 1985.

[19] R. D. Nicola, G. Ferrari, and R. Pugliese, "KLAIM: A kernel language for agents interaction and mobility," *IEEE Trans. Software Eng.*, vol. 24, no. 5, pp. 315–330, 1998.

[20] L. Lamport, "How to make a multiprocessor computer that correctly executes multiprocess programs," *IEEE Trans. Computers*, vol. 28, no. 9, pp. 690–691, 1979.

[21] E. Moiseenko, A. Podkopaev, and D. V. Koznov, "A survey of programming language memory models," *Program. Comput. Softw.*, vol. 47, no. 6, pp. 439–456, 2021.

[22] E. W. Dijkstra, "Cooperating sequential processes," in *The origin of concurrent programming: from semaphores to remote procedure calls*, pp. 65–138, Springer, 2002.

[23] S. I. Inc and D. L. Weaver, *SPARC architecture manual - version 8*. Prentice Hall, 1992.

[24] G. Boudol and G. Petri, "Relaxed memory models: an operational approach," in *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009* (Z. Shao and B. C. Pierce, eds.), pp. 392–403, ACM, 2009.

[25] M. Dubois, C. Scheurich, and F. A. Briggs, "Memory access buffering in multiprocessors," *ACM SIGARCH computer architecture news*, pp. 434–442, 1986.

[26] L. Howes, D. Hower, and B. Gaster, "Chapter 5 - hsa memory model," in *Heterogeneous System Architecture* (W. mei W. Hwu, ed.), pp. 53–75, Boston: Morgan Kaufmann, 2016.

[27] V. Danos and J. Krivine, "Reversible communicating systems," in *CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK, August 31 - September 3, 2004, Proceedings* (P. Gardner and N. Yoshida, eds.), vol. 3170 of *Lecture Notes in Computer Science*, pp. 292–307, Springer, 2004.

[28] I. C. C. Phillips and I. Ulidowski, "Reversing algebraic process calculi," *J. Log. Algebraic Methods Program.*, vol. 73, no. 1-2, pp. 70–96, 2007.

[29] I. Cristescu, J. Krivine, and D. Varacca, "A compositional semantics for the reversible p-calculus," in *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pp. 388–397, IEEE Computer Society, 2013.

[30] I. Lanese, C. A. Mezzina, and J. Stefani, "Reversibility in the higher-order π-calculus," *Theor. Comput. Sci.*, vol. 625, pp. 25–84, 2016.

[31] I. Lanese, N. Nishida, A. Palacios, and G. Vidal, "A theory of reversibility for erlang," *J. Log. Algebraic Methods Program.*, vol. 100, pp. 71–97, 2018.

[32] E. Giachino, I. Lanese, C. A. Mezzina, and F. Tiezzi, "Causal-consistent rollback in a tuple-based language," *J. Log. Algebraic Methods Program.*, vol. 88, pp. 99–120, 2017.

[33] I. Lanese and D. Medic, "A general approach to derive uncontrolled reversible semantics," in *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)* (I. Konnov and L. Kovács, eds.), vol. 171 of *LIPIcs*, pp. 33:1–33:24, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[34] L. Effinger-Dean and D. Grossman, "Modular metatheory for memory consistency models," tech. rep., University of Washington Department of Computer Science & Engineering, 2011.

[35] I. Lanese, I. C. C. Phillips, and I. Ulidowski, "An axiomatic approach to reversible computation," in *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings* (J. Goubault-Larrecq and B. König, eds.), vol. 12077 of *Lecture Notes in Computer Science*, pp. 442–461, Springer, 2020.

[36] E. Giachino, I. Lanese, and C. A. Mezzina, "Causal-consistent reversible debugging," in *Fundamental Approaches to Software Engineering - 17th International Conference, FASE 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings* (S. Gnesi and A. Rensink, eds.), vol. 8411 of *Lecture Notes in Computer Science*, pp. 370–384, Springer, 2014.

[37] I. Lanese, C. A. Mezzina, A. Schmitt, and J. Stefani, "Controlling reversibility in higher-order pi," in *CONCUR 2011 - Concurrency Theory - 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings* (J. Katoen and B. König, eds.), vol. 6901 of *Lecture Notes in Computer Science*, pp. 297–311, Springer, 2011.

[38] I. Lanese, A. Palacios, and G. Vidal, "Causal-consistent replay reversible semantics for message passing concurrent programs," *Fundam. Informaticae*, vol. 178, no. 3, pp. 229–266, 2021.

[39] I. Lanese, N. Nishida, A. Palacios, and G. Vidal, "Cauder: A causal-consistent reversible debugger for erlang," in *Functional and Logic Programming - 14th International Symposium, FLOPS 2018, Nagoya, Japan, May 9-11, 2018, Proceedings* (J. P. Gallagher and M. Sulzmann, eds.), vol. 10818 of *Lecture Notes in Computer Science*, pp. 247–263, Springer, 2018.

[40] J. J. González-Abril and G. Vidal, "Causal-consistent reversible debugging: Improving cauder," in *Practical Aspects of Declarative Languages - 23rd International Symposium, PADL 2021, Copenhagen, Denmark, January 18-19, 2021, Proceedings* (J. F. Morales and D. A. Orchard, eds.), vol. 12548 of *Lecture Notes in Computer Science*, pp. 145–160, Springer, 2021.

[41] G. Fabbretti, I. Lanese, and J. Stefani, "Causal-consistent debugging of distributed erlang programs," in *Reversible Computation - 13th International Conference, RC 2021, Virtual Event, July 7-8, 2021, Proceedings* (S. Yamashita and T. Yokoyama, eds.), vol. 12805 of *Lecture Notes in Computer Science*, pp. 79–95, Springer, 2021.

[42] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM J. Res. Dev.*, vol. 5, no. 3, pp. 183–191, 1961.

[43] J. Gray, "Why do computers stop and what can be done about it?," in *Fifth Symposium on Reliability in Distributed Software and Database Systems, SRDS 1986, Los Angeles, California, USA, January 13-15, 1986, Proceedings*, pp. 3–12, IEEE Computer Society, 1986.

[44] G. Fabbretti, I. Lanese, and J. Stefani, "Generation of a reversible semantics for erlang in maude," in *Formal Methods and Software Engineering - 23rd International Conference on Formal Engineering Methods, ICFEM 2022, Madrid, Spain, October 24-27, 2022, Proceedings* (A. Riesco and M. Zhang, eds.), vol. 13478 of *Lecture Notes in Computer Science*, pp. 106–122, Springer, 2022.

[45] J. Hoey and I. Ulidowski, "Reversible imperative parallel programs and debugging," in *Reversible Computation - 11th International Conference, RC 2019, Lausanne, Switzerland, June 24-25, 2019, Proceedings* (M. K. Thomsen and M. Soeken, eds.), vol. 11497 of *Lecture Notes in Computer Science*, pp. 108–127, Springer, 2019.

[46] J. Hoey and I. Ulidowski, "Reversing an imperative concurrent programming language," *Sci. Comput. Program.*, vol. 223, p. 102873, 2022.

[47] M. Schordan, T. Oppelstrup, D. R. Jefferson, and P. D. B. Jr., "Generation of reversible C++ code for optimistic parallel discrete event simulation," *New Gener. Comput.*, vol. 36, no. 3, pp. 257–280, 2018.

[48] K. Shibanai and T. Watanabe, "Actoverse: a reversible debugger for actors," in *Proceedings of the 7th ACM SIGPLAN International Workshop on Programming Based on Actors, Agents, and Decentralized Control, AGERE 2017, Vancouver, BC, Canada, October 23 - 27, 2017* (J. D. Koster and F. Bergenti, eds.), pp. 50–57, ACM, 2017.

[49] D. Aumayr, S. Marr, C. Béra, E. G. Boix, and H. Mössenböck, "Efficient and deterministic record & replay for actor languages," *CoRR*, vol. abs/1805.06267, 2018.

[50] Y. Chen, S. Zhang, Q. Guo, L. Li, R. Wu, and T. Chen, "Deterministic replay: A survey," *ACM Comput. Surv.*, vol. 48, no. 2, pp. 17:1–17:47, 2015.

[51] H. Patil, C. Pereira, M. Stallcup, G. Lueck, and J. Cownie, "Pinplay: a framework for deterministic replay and reproducible analysis of parallel programs," in *Proceedings of the CGO 2010, The 8th International Symposium on Code Generation and Optimization, Toronto, Ontario, Canada, April 24-28, 2010* (A. Moshovos, J. G. Steffan, K. M. Hazelwood, and D. R. Kaeli, eds.), pp. 2–11, ACM, 2010.

[52] T. Arts, J. Hughes, J. Johansson, and U. T. Wiger, "Testing telecoms software with quviq quickcheck," in *Proceedings of the 2006 ACM SIGPLAN Workshop on Erlang, Portland, Oregon, USA, September 16, 2006* (M. Feeley and P. W. Trinder, eds.), pp. 2–10, ACM, 2006.

[53] L. Aceto, A. Achilleos, D. P. Attard, L. Exibard, A. Francalanza, and A. Ingólfsdóttir, "A monitoring tool for linear-time μhml," in *Coordination Models and Languages - 24th IFIP WG 6.1 International Conference, COORDINATION 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings* (M. H. ter Beek and M. Sirjani, eds.), vol. 13271 of *Lecture Notes in Computer Science*, pp. 200–219, Springer, 2022.

[54] K. Claessen, M. H. Palka, N. Smallbone, J. Hughes, H. Svensson, T. Arts, and U. T. Wiger, "Finding race conditions in erlang with quickcheck and PULSE," in *Proceeding of the 14th ACM SIGPLAN international conference on Functional programming, ICFP 2009, Edinburgh, Scotland, UK, August 31 - September 2, 2009* (G. Hutton and A. P. Tolmach, eds.), pp. 149–160, ACM, 2009.

[55] J. J. González-Abril and G. Vidal, "A lightweight approach to computing message races with an application to causal-consistent reversible debugging," *CoRR*, vol. abs/2112.12869, 2021.

[56] A. J. Bernstein, "Analysis of programs for parallel processing," *IEEE Trans. Electron. Comput.*, vol. 15, no. 5, pp. 757–763, 1966.

[57] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[58] Available at `https://github.com/PietroLami/cauder`, 2022.

[59] J. Manson, W. W. Pugh, and S. V. Adve, "The java memory model," in *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2005, Long Beach, California, USA, January 12-14, 2005* (J. Palsberg and M. Abadi, eds.), pp. 378–391, ACM, 2005.

[60] J. Bender and J. Palsberg, "A formalization of java's concurrent access modes," *Proc. ACM Program. Lang.*, vol. 3, no. OOPSLA, pp. 142:1–142:28, 2019.

[61] M. Batty, S. Owens, S. Sarkar, P. Sewell, and T. Weber, "Mathematizing C++ concurrency," in *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011* (T. Ball and M. Sagiv, eds.), pp. 55–66, ACM, 2011.

[62] A. Arnold, "Transition systems and concurrent processes," *Banach Center Publications*, vol. 21, no. 1, pp. 9–20, 1988.

Appendix A

Imperative Constructs: Square Property

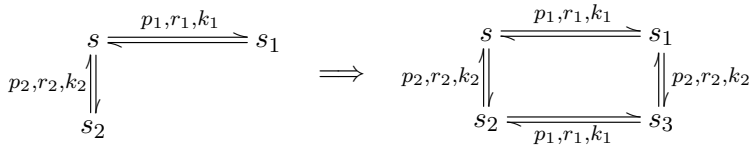
Lemma 2 (Square Property)**.** *Given two co-initial concurrent transitions*

$$t_1 = (s \rightleftharpoons_{p_1,r_1,k_1} s_1) \qquad \textit{and} \qquad t_2 = (s \rightleftharpoons_{p_2,r_2,k_2} s_2)$$

there exist two transitions

$$t_2/t_1 = (s_1 \rightleftharpoons_{p_2,r_2,k_2} s_3) \qquad \textit{and} \qquad t_1/t_2 = (s_2 \rightleftharpoons_{p_1,r_1,k_1} s_3)$$

Graphically:



Proof. We distinguish the following cases depending on the applied rules:

1. Two forward transitions. We have the following subcases:

   - the two transitions are not on the map: we can prove that by applying rule r_2 to p_1 in s_1 and rule r_1 to p_2 in s_2 we have two transitions t_1/t_2 and t_2/t_1 which are co-final, as desired;
   - one of the two transitions is on the map and the other one is not: we can apply the same reasoning as above;
   - both transitions are on the map: we have a case analysis on the applied rules. We show a few examples, the others are similar.
     - if $r_1 \in \{\textsc{UnregisterS}, \textsc{EndUn}\}$, $k_1 = \mathbf{del}(\theta_1, e_1, S_1, \mathrm{M}_1, \mathrm{M}')$ and $r_2 \in \{\textsc{UnregisterS}, \textsc{EndUn}\}$, $k_2 = \mathbf{del}(\theta_2, e_2, S_2, \mathrm{M}_2, \mathrm{M}'')$ we have that both $\mathrm{M}_1 = \{\langle a_1, p_1, t_1, \top\rangle\}$ and $\mathrm{M}_2 = \{\langle a_2, p_2, t_2, \top\rangle\}$ are in the map M of s and by definition of concurrent transitions (Definition 18) we also have that $a_1 \neq a_2$ and $p_1 \neq p_2$. Then applying k_1 followed by k_2 or vice-versa leads us to the same system because

$$(\mathrm{M} \setminus \mathrm{M}_1 \cup \mathrm{kill}(\mathrm{M}_1)) \setminus \mathrm{M}_2 \cup \mathrm{kill}(\mathrm{M}_2) = (\mathrm{M} \setminus \mathrm{M}_2 \cup \mathrm{kill}(\mathrm{M}_2)) \setminus \mathrm{M}_1 \cup \mathrm{kill}(\mathrm{M}_1)$$

The only side effect of each rule is to kill a tuple from the system's map.

- if $r_1 = \text{RegisterS}$ and $r_2 \in \{\text{UnregisterS}, \text{EndUn}\}$, $k_2 = \mathbf{del}(\_, \_, \_, \mathrm{M}_1, \mathrm{M}_2)$ we know thanks to rule r_1 that $\langle a_1, p_1, t_1, \top\rangle$ is not in the map M of s. Thanks to rule r_2 we know that the tuple $\mathrm{M}_1 = \langle a_2, p_2, t_2, \top\rangle$ is in M. By Definition 18 we have that $a_1 \neq a_2$, $p_1 \neq p_2$ and, for this reason, M_2 is not affected by r_1. We can see that adding the tuple $\langle a_1, p_1, t_1, \top\rangle$ and then killing the tuple $\langle a_2, p_2, t_2, \top\rangle$, or vice-versa, leads us to the same system because

$$(\mathrm{M} \cup \{\langle a_1, p_1, t_1, \top\rangle\}) \setminus \mathrm{M}_1 \cup \mathrm{kill}(\mathrm{M}_1) = (\mathrm{M} \setminus \mathrm{M}_1 \cup \mathrm{kill}(\mathrm{M}_1)) \cup \{\langle a_1, p_1, t_1, \top\rangle\}$$

  The only side effects of the two rules are, respectively, an insertion and a kill of a tuple from the map M.
- if $r_1 = \text{RegisterS}$ and $r_2 \in \{\text{UnregisterF}, \text{SendF}, \text{Whereis2}, \text{End}\}$, $k_2 = \mathbf{readF}(\_, \_, \_, \iota, \mathrm{M}_2)$, we know thanks to rule r_1 that $\mathrm{M}_1 = \{\langle a_1, p_1, t_1, \top\rangle\}$ is not in the map M of s. Thanks to rule r_2 we know that M_2 is a subset of M. By Definition 18 we have that $a_1 \neq \iota \wedge \iota \neq p_1$ so M_2 is not affected by r_1, hence adding the tuple $\langle a_1, p_1, t_1, \top\rangle$ and then reading a ghost tuple, or vice-versa, leads us to the same system.
- if $r_1 \in \{\text{UnregisterS}, \text{EndUn}\}$, $k_1 = \mathbf{del}(\theta_1, e_1, S_1, \mathrm{M}_1, \mathrm{M})$ and $r_2 \in \{\text{UnregisterF}, \text{SendF}, \text{Whereis2}, \text{End}\}$, $k_2 = \mathbf{readF}(\_, \_, \_, \iota, \mathrm{M}_2)$, we know thanks to rule r_1 that $\mathrm{M}_1 = \{\langle a_1, p_1, t_1, \top\rangle\}$ is in the map M of s. By Definition 18 we have that $a_1 \neq \iota \wedge \iota \neq p_1$ so M_2 is not affected by r_1, hence killing the tuple $\langle a_1, p_1, t_1, \top\rangle$ and then reading a ghost tuple, or vice-versa, leads us to the same system.
- if $k_1 = \mathbf{del}(\theta_1, e_1, S_1, \mathrm{M}_1, \mathrm{M})$ and $k_2 = \mathbf{readS}(\_, \_, \_, \mathrm{M}_2)$, we know thanks to rule r_1 that $\mathrm{M}_1 = \{\langle a_1, p_1, t_1, \top\rangle\}$ is in the map M of s. Thanks to rule r_2 we know that M_2 is a subset of M. By Definition 18 we have that $\mathrm{M}_1 \cap \mathrm{M}_2 = \emptyset$ and so M_2 is not affected by r_1. We can see that killing the tuple $\langle a_1, p_1, t_1, \top\rangle$ and then reading one or two tuples, or vice-versa, leads us to the same system.
- if $r_1 = \text{RegisterS}$ and $r_2 \in \{\text{RegisterF}, \text{Whereis1}\}$, $k_2 = \mathbf{readS}(\_, \_, \_, \mathrm{M}_2)$ we know thanks to rule r_1 that $\mathrm{M}_1 = \{\langle a_1, p_1, t_1, \top\rangle\}$ is not in the map M of s. Thanks to rule r_2 we know that M_2 is a subset of M. By Definition 18 we have that $\langle \_, p_1, \_, \_\rangle \notin \mathrm{M}_2$ and $\langle a_1, \_, \_, \_\rangle \notin \mathrm{M}_2$ and, for this reason, M_2 is not affected by r_1. We can see that adding the tuple $\langle a_1, p_1, t_1, \top\rangle$ and then reading one or two tuples, or vice-versa, leads us to the same system.
- if $r_1 = r_2 = \text{RegisterS}$ we know thanks to rule r_1 that $\mathrm{M}_1 = \{\langle a_1, p_1, t_1, \top\rangle\}$ is not in the map M of s and thanks to rule r_2 that $\mathrm{M}_2 = \{\langle a_2, p_2, t_2, \top\rangle\}$ is not in M. By Definition 18 we have that $a_1 \neq a_2$ and $p_1 \neq p_2$. We can see that adding the tuple M_1 and then adding the tuple M_2, or vice-versa, leads us to the same system because

$$(\mathrm{M} \setminus \mathrm{M}_1) \cup \mathrm{M}_2 = (\mathrm{M} \setminus \mathrm{M}_2) \cup \mathrm{M}_1$$

2. One forward transition and one backward transition. We have the following subcases:

- the two transitions are both on the map, then we have the following subcases:
  - $r_1 = \textsc{RegisterS}$, where $k_1 = \mathbf{regS}(\_, \_, \_, \mathrm{M}_1)$ with $\mathrm{M}_1 = \{\langle a_1, p_1, t_1, \top\rangle\}$, and $r_2 = \overline{\textsc{Del}}$, where $k_2 = \mathbf{del}(\_, \_, \_, \mathrm{M}_2, \_)$ with $\mathrm{M}_2 = \{\langle a_2, p_2, t_2, \top\rangle\}$. By Definition 18, we know that $a_1 \neq a_2$ and $p_1 \neq p_2$ (because otherwise they would be in conflict), then the map M of s contains the tuple $\langle a_2, p_2, t_2, \bot\rangle$ (otherwise it would not be possible to apply rule r_2) and we can see that the applications of r_1 and r_2 commute because

$$(\mathrm{M} \cup \mathrm{M}_1) \setminus \mathrm{kill}(\mathrm{M}_2) \cup \mathrm{M}_2 = (\mathrm{M} \setminus \mathrm{kill}(\mathrm{M}_2) \cup \mathrm{M}_2) \cup \mathrm{M}_1$$

  - $r_1 \in \{\textsc{UnregisterS}, \textsc{EndUn}\}$, $k_1 = \mathbf{del}(\_, \_, \_, \mathrm{M}_1, \mathrm{M}_1')$, where $\mathrm{M}_1 = \{\langle a_1, p_1, t_1, \top\rangle\}$, and $r_2 = \overline{\textsc{RegisterS}}$, where $k_2 = \mathbf{regS}(\_, \_, \_, \mathrm{M}_2)$ and $\mathrm{M}_2 = \{\langle a_2, p_2, t_2, \top\rangle\}$. By Definition 18 we have that $a_1 \neq a_2$ and $p_1 \neq p_2$, then the map M of s contains both the tuple $\langle a_1, p_1, t_1, \top\rangle$ and the tuple $\langle a_2, p_2, t_2, \top\rangle$. Hence the applications of the two rules commute because

$$(\mathrm{M} \setminus \mathrm{M}_1 \cup \mathrm{kill}(\mathrm{M}_1)) \setminus \mathrm{M}_2 = (\mathrm{M} \setminus \mathrm{M}_2) \setminus \mathrm{M}_1 \cup \mathrm{kill}(\mathrm{M}_1)$$

  - $r_1 \in \{\textsc{UnregisterF}, \textsc{SendF}, \textsc{Whereis2}, \textsc{End}\}$, $k_1 = \mathbf{readF}(\_, \_, \_, \iota, \mathrm{M})$ and $r_2 = \overline{\textsc{RegisterS}}$, where $k_2 = \mathbf{regS}(\_, \_, \_, \mathrm{M}_2)$ and $\mathrm{M}_2 = \{\langle a_2, p_2, t_2, \top\rangle\}$. By Definition 18 we have that $a_2 \neq \iota \wedge \iota \neq p_2$ and so we can see that the applications of the two rules commute.
  - $r_1 \in \{\textsc{UnregisterS}, \textsc{EndUn}\}$, $k_1 = \mathbf{del}(\_, \_, \_, \mathrm{M}_1, \mathrm{M}_1')$, where $\mathrm{M}_1 = \{\langle a_1, p_1, t_1, \top\rangle\}$, and $r_1 = \overline{\textsc{ReadS}}$ where $k_2 = \mathbf{readS}(\_, \_, \_, \mathrm{M}_2)$. By Definition 18 we have that $\langle \_, \_, t_1, \_\rangle \notin \mathrm{M}_2$, also we know that r_2 does not affect M_1' so we can conclude that the applications of the two rules commute.
  - $r_1 \in \{\textsc{UnregisterS}, \textsc{EndUn}\}$, $k_1 = \mathbf{del}(\_, \_, \_, \mathrm{M}_1, \mathrm{M}_1')$, where $\mathrm{M}_1 = \{\langle a_1, p_1, t_1, \top\rangle\}$, and $r_1 = \overline{\textsc{ReadF}}$ where $k_2 = \mathbf{readF}(\_, \_, \_, \iota, \mathrm{M}_2)$. By Definition 18 we have that $a_1 \neq \iota \wedge \iota \neq p_1$, so r_1 does not affect M_2. Also, r_2 does not affect M_1' since it is a read operation, so we can conclude that the applications of the two rules commute.
  - $r_1 \in \{\textsc{UnregisterS}, \textsc{EndUn}\}$, $k_1 = \mathbf{del}(\_, \_, \_, \mathrm{M}_1, \mathrm{M}_1')$, where $\mathrm{M}_1 = \{\langle a_1, p_1, t_1, \top\rangle\}$, $r_2 = \overline{\textsc{Del}}$ and $k_2 = \mathbf{del}(\_, \_, \_, \mathrm{M}_2, \mathrm{M}_2')$, where $\mathrm{M}_2 = \{\langle a_2, p_2, t_2, \top\rangle\}$. By Definition 18 we have that $a_1 \neq a_2 \wedge p_1 \neq p_2$, so r_1 does not affect M_2', and similarly r_2 does not affect M_1', so the applications of the two rules commute because

$$(\mathrm{M} \setminus \mathrm{M}_1 \cup \mathrm{kill}(\mathrm{M}_1)) \setminus \mathrm{kill}(\mathrm{M}_2) \cup \mathrm{M}_2 = (\mathrm{M} \setminus \mathrm{kill}(\mathrm{M}_2) \cup \mathrm{M}_2) \setminus \mathrm{M}_1 \cup \mathrm{kill}(\mathrm{M}_1)$$

  - in the other cases we can apply a similar reasoning.
- the two transitions are not both on the map: the claim follows easily (see [39, Lemma 13] and [41, Lemma 3.1]).

3. Two backward transitions. We have a case analysis on the two rules. We show below a

few examples related to the map, we refer to [39, Lemma 13] and [41, Lemma 3.1] for additional examples related to the other constructs:

- $r_1 = \overline{\text{REGISTERS}}$, where $k_1 = \mathbf{regS}(\_, \_, \_, \mathrm{M}_1)$ with $\mathrm{M}_1 = \{\langle a_1, p_1, t_1, \top\rangle\}$, and $r_2 = \overline{\text{DEL}}$, where $k_2 = \mathbf{del}(\_, \_, \_, \mathrm{M}_2, \_)$ with $\mathrm{M}_2 = \{\langle a_2, p_2, t_2, \top\rangle\}$. The map M of s contains the tuple $\langle a_2, p_2, t_2, \bot\rangle$ (otherwise it would not possible to apply rule r_2) and the tuple $\langle a_1, p_1, t_1, \top\rangle$ (otherwise it would not possible to apply rule r_1). Also, we know that $t_1 \neq t_2$ since identifiers are unique. Now we can see that the applications of rules r_1 and r_2 commute because

$$(\mathrm{M} \setminus \mathrm{M}_1) \setminus \mathrm{kill}(\mathrm{M}_2) \cup \mathrm{M}_2 = (\mathrm{M} \setminus \mathrm{kill}(\mathrm{M}_2) \cup \mathrm{M}_2) \setminus \mathrm{M}_1$$

- $r_1 = \overline{\text{DEL}}$, $k_1 = \mathbf{del}(\_, \_, \_, \mathrm{M}_1, \mathrm{M}_1')$, where $\mathrm{M}_1 = \{\langle a_1, p_1, t_1, \top\rangle\}$, and $r_1 = \overline{\text{READF}}$ where $k_2 = \mathbf{readF}(\_, \_, \_, \iota, \mathrm{M}_2)$. From the side condition $\mathrm{readghost}(t_1, \Pi) = \emptyset$ of rule $\overline{\text{DEL}}$ we know that the tuple with identifier t_1 is not in M_2. By combining this with the side condition $\mathrm{M}_2 = \mathrm{M}^{\iota}$ of rule $\overline{\text{READF}}$ we have that $a_1 \neq \iota \wedge \iota \neq p_1$, so r_1 does not affect M_2. Also, r_2 does not affect M_1' since it is a read operation. We can conclude that the applications of the two rules commute. □

Appendix B

Memory Models Proofs

B.1 Sequential Consistency

Theorem 8. $P \mid (\sigma; L) \sim (T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S)$

$$\begin{array}{ccc} P \mid (\sigma; L) & \underset{Rel^{-1}}{\overset{Rel}{\longleftrightarrow}} & T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S \\ {\scriptstyle \varphi(\alpha)}\downarrow & & \downarrow{\scriptstyle \alpha} \\ P' \mid (\sigma'; L') & \underset{Rel^{-1}}{\overset{Rel}{\longleftrightarrow}} & T' \otimes_{\mathcal{R}} M' \otimes_{\mathcal{R}} S' \end{array}$$

Proof. Firstly, we need to define the relation (Rel) between the states of $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ and of $P \mid (\sigma; L)$, where

$$\begin{gathered} Rel = \{(T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S; P \mid (\sigma; L)) \mid \forall \langle \theta, e \rangle \in T \text{ then } P(\theta) = e; \\ \forall (r \mapsto v) \in M \text{ then } \sigma(r) = v; \\ \forall (r \mapsto \circ) \in M \text{ then } r \notin \mathsf{dom}(\sigma); \\ \forall (l \mapsto \theta) \in S \text{ then } L(l) = \theta; \\ \forall (l \mapsto \circ) \in S \text{ then } l \notin \mathsf{dom}(L)\} \end{gathered}$$

Now we show by case that for each transitions done by $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ then the $P \mid (\sigma; L)$ can do the same transitions. Now we proceed by cases on the labels:

- We consider the case where $\varphi(\langle(\theta, \mathsf{pure}), \phi, \phi\rangle) = (\theta, \mathsf{pure})$, then in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{\mathsf{pure}} e'}{\langle \theta, e \rangle \xrightarrow{(\theta, \mathsf{pure})} \langle \theta, e' \rangle}}{\langle \theta, e \rangle \mid \mathcal{T} \xrightarrow{(\theta, \mathsf{pure})} \langle \theta, e' \rangle \mid \mathcal{T}} \qquad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \qquad \langle((\theta, \mathsf{pure}), \phi, \phi)\rangle \in \mathcal{R}}{\langle \theta, e \rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta, \mathsf{pure}), \phi, \phi)} \langle \theta, e' \rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

  in $P \mid (\sigma; L)$ we have $P[\theta \mapsto e]$ (by Rel), and we consider the label (θ, pure). Then, we

obtain:

$$\textsc{Program}\ \frac{\textsc{PrgmStep}\ \dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{pure}} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{pure})} P[\theta \mapsto e']}}{P \mid (\sigma; L) \xrightarrow{(\theta,\mathsf{pure})} P[\theta \mapsto e'] \mid (\sigma; L)}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi(\langle(\theta, \mathsf{sp}(\theta')), \phi, \phi\rangle) = (\theta, \mathsf{sp}(\theta'))$, then in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{\mathsf{sp}(\theta')} e'; e''}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \langle\theta, e'\rangle \mid \langle\theta', e''\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \langle\theta, e'\rangle \mid \langle\theta', e''\rangle \mid \mathcal{T}} \qquad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \qquad \langle((\theta, \mathsf{sp}(\theta')), \phi, \phi)\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\mathsf{sp}(\theta')),\phi,\phi)} \langle\theta, e'\rangle \mid \langle\theta', e''\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

in $P \mid (\sigma; L)$ we have $P[\theta \mapsto e]$ (by *Rel*), and we consider the label $(\theta, \mathsf{sp}(\theta'))$. Then, we obtain:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad \theta' \notin \mathsf{dom}(P) \qquad e \xhookrightarrow{\mathsf{sp}(\theta')} e'; e''}{P \xrightarrow{(\theta,\mathsf{sp}(\theta'))} P[\theta \mapsto e'][\theta' \mapsto e'']} \qquad \dfrac{}{(\sigma; L) \xrightarrow{(\theta,\mathsf{sp}(\theta'))} (\sigma; L)}}{P \mid (\sigma; L) \xrightarrow{(\theta,\mathsf{sp}(\theta'))} P[\theta \mapsto e'][\theta' \mapsto e''] \mid (\sigma; L)}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi(\langle(\theta, \mathsf{ref}(r, v)), (\theta, \mathsf{ref}(r, v)), \phi\rangle) = (\theta, \mathsf{ref}(r, v))$, then in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{ref}(r,v))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{ref}(r,v))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{ref}(r,v))} \langle\theta, e'\rangle \mid \mathcal{T}} \qquad \dfrac{\dfrac{}{(r \mapsto \circ) \xrightarrow{(\theta,\mathsf{ref}(r,v))} (r \mapsto v)}}{(r \mapsto \circ) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{ref}(r,v))} (r \mapsto v) \mid \mathcal{M}} \qquad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \qquad \langle\langle(\theta, \mathsf{ref}(r, v)), (\theta, \mathsf{ref}(r, v)), \phi\rangle\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto \circ) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{\langle(\theta,\mathsf{ref}(r,v)),(\theta,\mathsf{ref}(r,v)),\phi\rangle} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

in $P \mid (\sigma; L)$ we have $P[\theta \mapsto e]$ and $r \notin \mathsf{dom}(\sigma)$ (by *Rel*), and we consider the label $(\theta, \mathsf{ref}(r, v))$. Then, we obtain:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{ref}(r,v)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{ref}(r,v))} P[\theta \mapsto e']} \qquad \dfrac{r \notin \mathsf{dom}(\sigma)}{(\sigma; L) \xrightarrow{(\theta,\mathsf{ref}(r,v))} (\sigma[r \mapsto v], L)}}{P \mid (\sigma; L) \xrightarrow{(\theta,\mathsf{ref}(r,v))} P[\theta \mapsto e'] \mid (\sigma[r \mapsto v]; L)}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi(\langle(\theta, \mathsf{rd}(r,v)), (\theta, \mathsf{rd}(r,v)), \phi\rangle) = (\theta, \mathsf{rd}(r,v))$, then in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{rd}(r,v))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \dfrac{\dfrac{}{(r \mapsto v) \xrightarrow{(\theta,\mathsf{rd}(r,v))} (r \mapsto v)}}{(r \mapsto v) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{rd}(r,v))} (r \mapsto v) \mid \mathcal{M}} \quad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \quad \langle((\theta, \mathsf{rd}(r,v)), (\theta, \mathsf{rd}(r,v)), \phi)\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\mathsf{rd}(r,v)),(\theta,\mathsf{rd}(r,v)),\phi)} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

in $P \mid (\sigma; L)$ we have $P[\theta \mapsto e]$ and $\sigma[r \mapsto v]$ (by *Rel*), and we consider the label $(\theta, \mathsf{rd}(r,v))$. Then, we obtain:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{rd}(r,v)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{rd}(r,v))} P[\theta \mapsto e']} \qquad \dfrac{\sigma(r) = v}{(\sigma; L) \xrightarrow{(\theta,\mathsf{rd}(r,v))} (\sigma; L)}}{P \mid (\sigma; L) \xrightarrow{(\theta,\mathsf{rd}(r,v))} P[\theta \mapsto e'] \mid (\sigma; L)}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi(\langle(\theta, \mathsf{wr}(r,v)), (\theta, \mathsf{wr}(r,v)), \phi\rangle) = (\theta, \mathsf{wr}(r,v))$, then in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{wr}(r,v))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{wr}(r,v))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{wr}(r,v))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \dfrac{\dfrac{}{(r \mapsto v_{old}) \xrightarrow{(\theta,\mathsf{wr}(r,v))} (r \mapsto v)}}{(r \mapsto v_{old}) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{wr}(r,v))} (r \mapsto v) \mid \mathcal{M}} \quad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \quad \langle((\theta, \mathsf{wr}(r,v)), (\theta, \mathsf{wr}(r,v)), \phi)\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v_{old}) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\mathsf{wr}(r,v)),(\theta,\mathsf{wr}(r,v)),\phi)} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

in $P \mid (\sigma; L)$ we have $P[\theta \mapsto e]$ and $\sigma[r \mapsto v_{old}]$ (by *Rel*), and we consider the label $(\theta, \mathsf{wr}(r,v))$. Then, we obtain:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{wr}(r,v)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{wr}(r,v))} P[\theta \mapsto e']} \qquad \dfrac{}{(\sigma; L) \xrightarrow{(\theta,\mathsf{wr}(r,v))} (\sigma[r \mapsto v]; L)}}{P \mid (\sigma; L) \xrightarrow{(\theta,\mathsf{wr}(r,v))} P[\theta \mapsto e'] \mid (\sigma[r \mapsto v]; L)}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi(\langle(\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l))\rangle) = (\theta, \mathsf{acq}(l))$, then in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$

we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{acq}(l))} e'}{\langle\theta,e\rangle \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta,e'\rangle}}{\langle\theta,e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta,e'\rangle \mid \mathcal{T}} \quad \dfrac{\dfrac{}{(l \mapsto \circ) \xrightarrow{(\theta,\mathsf{acq}(l))} (l \mapsto \theta)}}{(l \mapsto \circ) \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{acq}(l))} (l \mapsto \theta) \mid \mathcal{S}} \quad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \quad \langle((\theta,\mathsf{acq}(l)),\phi,(\theta,\mathsf{acq}(l)))\rangle \in \mathcal{R}}{\langle\theta,e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \circ) \mid \mathcal{S} \xrightarrow{((\theta,\mathsf{acq}(l)),\phi,(\theta,\mathsf{acq}(l)))} \langle\theta,e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \theta) \mid \mathcal{S}}$$

in $P \mid (\sigma; L)$ we have $P[\theta \mapsto e]$ and $l \notin \mathsf{dom}(L)$ (by *Rel*), and we consider the label $(\theta, \mathsf{acq}(l))$. Then, we obtain:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{acq}(l)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{acq}(l))} P[\theta \mapsto e']} \qquad \dfrac{l \notin dom(L)}{(\sigma; L) \xrightarrow{(\theta,\mathsf{acq}(l))} (\sigma; L[l \mapsto \theta])}}{P \mid (\sigma; L) \xrightarrow{(\theta,\mathsf{acq}(l))} P[\theta \mapsto e'] \mid (\sigma; L[l \mapsto \theta])}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi(\langle(\theta,\mathsf{rel}(l)),\phi,(\theta,\mathsf{rel}(l))\rangle) = (\theta,\mathsf{rel}(l))$, then in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{rel}(l))} e'}{\langle\theta,e\rangle \xrightarrow{(\theta,\mathsf{rel}(l))} \langle\theta,e'\rangle}}{\langle\theta,e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{rel}(l))} \langle\theta,e'\rangle \mid \mathcal{T}} \quad \dfrac{\dfrac{}{(l \mapsto \theta) \xrightarrow{(\theta,\mathsf{rel}(l))} (l \mapsto \circ)}}{(l \mapsto \theta) \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{rel}(l))} (l \mapsto \circ) \mid \mathcal{S}} \quad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \quad \langle((\theta,\mathsf{rel}(l)),\phi,(\theta,\mathsf{rel}(l)))\rangle \in \mathcal{R}}{\langle\theta,e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \theta) \mid \mathcal{S} \xrightarrow{((\theta,\mathsf{rel}(l)),\phi,(\theta,\mathsf{rel}(l)))} \langle\theta,e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \circ) \mid \mathcal{S}}$$

in $P \mid (\sigma; L)$ we have $P[\theta \mapsto e]$ and $L[l \mapsto \theta]$ (by *Rel*), and we consider the label $(\theta, \mathsf{rel}(l))$. Then, we obtain:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{rel}(l)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{rel}(l))} P[\theta \mapsto e']} \qquad \dfrac{L(l) = \theta}{(\sigma; L) \xrightarrow{(\theta,\mathsf{rel}(l))} (\sigma; L_{|l})}}{P \mid (\sigma; L) \xrightarrow{(\theta,\mathsf{rel}(l))} P[\theta \mapsto e'] \mid (\sigma; L_{|l})}$$

we can easily see that the final states are still related.

Now we consider the case starting from $P \mid (\sigma; L)$ and we consider then Rel^{-1} and φ^{-1}. We proceed by the case for each possible labels:

- We consider the case where $\varphi^{-1}((\theta,\mathsf{pure})) = \langle(\theta,\mathsf{pure}),\phi,\phi\rangle$, then in $P \mid (\sigma; L)$ we have

that:

$$\textsc{Program}\ \frac{\textsc{PrgmStep}\ \dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{pure}} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{pure})} P[\theta \mapsto e']}}{P \mid (\sigma; L) \xrightarrow{(\theta,\mathsf{pure})} P[\theta \mapsto e'] \mid (\sigma; L)}$$

in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ we have $(\theta \mapsto e) \in T$ (by Rel^{-1}), and we consider the label $\langle(\theta, \mathsf{pure}), \phi, \phi\rangle$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{\mathsf{pure}} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{pure})} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{pure})} \langle\theta, e'\rangle \mid \mathcal{T}} \qquad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \qquad \langle((\theta, \mathsf{pure}), \phi, \phi)\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\mathsf{pure}),\phi,\phi)} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi^{-1}((\theta, \mathsf{sp}(\theta'))) = \langle(\theta, \mathsf{sp}(\theta')), \phi, \phi\rangle$, then in $P \mid (\sigma; L)$ we have that:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad \theta' \notin \mathsf{dom}(P) \qquad e \xhookrightarrow{\mathsf{sp}(\theta')} e'; e''}{P \xrightarrow{(\theta,\mathsf{sp}(\theta'))} P[\theta \mapsto e'][\theta' \mapsto e'']} \qquad \frac{}{(\sigma; L) \xrightarrow{(\theta,\mathsf{sp}(\theta'))} (\sigma; L)}}{P \mid (\sigma; L) \xrightarrow{(\theta,\mathsf{sp}(\theta'))} P[\theta \mapsto e'][\theta' \mapsto e''] \mid (\sigma; L)}$$

in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ we have $(\theta \mapsto e) \in T$ (by Rel^{-1}), and we consider the label $\langle(\theta, \mathsf{sp}(\theta')), \phi, \phi\rangle$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{\mathsf{sp}(\theta')} e'; e''}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \langle\theta, e'\rangle \mid \langle\theta', e''\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \langle\theta, e'\rangle \mid \langle\theta', e''\rangle \mid \mathcal{T}} \qquad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \qquad \langle((\theta, \mathsf{sp}(\theta')), \phi, \phi)\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\mathsf{sp}(\theta')),\phi,\phi)} \langle\theta, e'\rangle \mid \langle\theta', e''\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi^{-1}((\theta, \mathsf{ref}(r, v))) = \langle(\theta, \mathsf{ref}(r, v)), (\theta, \mathsf{ref}(r, v)), \phi\rangle$, then in $P \mid (\sigma; L)$ we have that:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{ref}(r,v)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{ref}(r,v))} P[\theta \mapsto e']} \qquad \frac{r \notin \mathsf{dom}(\sigma)}{(\sigma; L) \xrightarrow{(\theta,\mathsf{ref}(r,v))} (\sigma[r \mapsto v]; L)}}{P \mid (\sigma; L) \xrightarrow{(\theta,\mathsf{ref}(r,v))} P[\theta \mapsto e'] \mid (\sigma[r \mapsto v]; L)}$$

in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ we have $(\theta \mapsto e) \in T$ and $(r \mapsto \circ) \in M$ (by Rel^{-1}), and we consider the label $\langle(\theta, \mathsf{ref}(r,v)), (\theta, \mathsf{ref}(r,v)), \phi\rangle$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{ref}(r,v))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{ref}(r,v))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{ref}(r,v))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \frac{\overline{(r \mapsto \circ) \xrightarrow{(\theta,\mathsf{ref}(r,v))} (r \mapsto v)}}{(r \mapsto \circ) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{ref}(r,v))} (r \mapsto v) \mid \mathcal{M}} \quad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \quad \langle\langle(\theta, \mathsf{ref}(r,v)), (\theta, \mathsf{ref}(r,v)), \phi\rangle\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto \circ) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{\langle(\theta,\mathsf{ref}(r,v)),(\theta,\mathsf{ref}(r,v)),\phi\rangle} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi^{-1}((\theta, \mathsf{rd}(r,v))) = \langle(\theta, \mathsf{rd}(r,v)), (\theta, \mathsf{rd}(r,v)), \phi\rangle$, then in $P \mid (\sigma; L)$ we have that:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{rd}(r,v)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{rd}(r,v))} P[\theta \mapsto e']} \qquad \frac{\sigma(r) = v}{(\sigma; L) \xrightarrow{(\theta,\mathsf{rd}(r,v))} (\sigma; L)}}{P \mid (\sigma; L) \xrightarrow{(\theta,\mathsf{rd}(r,v))} P[\theta \mapsto e'] \mid (\sigma; L)}$$

in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ we have $(\theta \mapsto e) \in T$ and $(r \mapsto v) \in M$ (by Rel^{-1}), and we consider the label $\langle(\theta, \mathsf{rd}(r,v)), (\theta, \mathsf{rd}(r,v)), \phi\rangle$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{rd}(r,v))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \frac{\overline{(r \mapsto v) \xrightarrow{(\theta,\mathsf{rd}(r,v))} (r \mapsto v)}}{(r \mapsto v) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{rd}(r,v))} (r \mapsto v) \mid \mathcal{M}} \quad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \quad \langle\langle(\theta, \mathsf{rd}(r,v)), (\theta, \mathsf{rd}(r,v)), \phi\rangle\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{\langle(\theta,\mathsf{rd}(r,v)),(\theta,\mathsf{rd}(r,v)),\phi\rangle} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi^{-1}((\theta, \mathsf{wr}(r,v))) = \langle(\theta, \mathsf{wr}(r,v)), (\theta, \mathsf{wr}(r,v)), \phi\rangle$, then in $P \mid (\sigma; L)$ we have that:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{wr}(r,v)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{wr}(r,v))} P[\theta \mapsto e']} \qquad \frac{}{(\sigma; L) \xrightarrow{(\theta,\mathsf{wr}(r,v))} (\sigma[r \mapsto v]; L)}}{P(\sigma; L) \mid \xrightarrow{(\theta,\mathsf{wr}(r,v))} P[\theta \mapsto e'] \mid (\sigma[r \mapsto v]; L)}$$

in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ we have $(\theta \mapsto e) \in T$ and $(r \mapsto v_{old}) \in M$ (by Rel^{-1}), and we consider

the label $\langle(\theta, \mathsf{wr}(r,v)), (\theta, \mathsf{wr}(r,v)), \phi\rangle$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{wr}(r,v))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{wr}(r,v))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{wr}(r,v))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \dfrac{\dfrac{}{(r \mapsto v_{old}) \xrightarrow{(\theta,\mathsf{wr}(r,v))} (r \mapsto v)}}{(r \mapsto v_{old}) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{wr}(r,v))} (r \mapsto v) \mid \mathcal{M}} \quad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \quad \langle((\theta, \mathsf{wr}(r,v)), (\theta, \mathsf{wr}(r,v)), \phi)\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v_{old}) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\mathsf{wr}(r,v)),(\theta,\mathsf{wr}(r,v)),\phi)} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi^{-1}((\theta, \mathsf{acq}(l))) = \langle(\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l))\rangle$, then in $P \mid (\sigma; L)$ we have that:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{acq}(l)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{acq}(l))} P[\theta \mapsto e']} \qquad \dfrac{l \notin dom(L)}{(\sigma; L) \xrightarrow{(\theta,\mathsf{acq}(l))} (\sigma; L[l \mapsto \theta])}}{P \mid (\sigma; L) \xrightarrow{(\theta,\mathsf{acq}(l))} P[\theta \mapsto e'] \mid (\sigma; L[l \mapsto \theta])}$$

in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ we have $(\theta \mapsto e) \in T$ and $(l \mapsto \circ) \in S$ (by Rel^{-1}), and we consider the label $\langle(\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l))\rangle$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{acq}(l))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \dfrac{\dfrac{}{(l \mapsto \circ) \xrightarrow{(\theta,\mathsf{acq}(l))} (l \mapsto \theta)}}{(l \mapsto \circ) \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{acq}(l))} (l \mapsto \theta) \mid \mathcal{S}} \quad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \quad \langle((\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l)))\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \circ) \mid \mathcal{S} \xrightarrow{((\theta,\mathsf{acq}(l)),\phi,(\theta,\mathsf{acq}(l)))} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \theta) \mid \mathcal{S}}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi^{-1}((\theta, \mathsf{rel}(l))) = \langle(\theta, \mathsf{rel}(l)), \phi, (\theta, \mathsf{rel}(l))\rangle$, then in $P \mid (\sigma; L)$ we have that:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{rel}(l)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{rel}(l))} P[\theta \mapsto e']} \qquad \dfrac{L(l) = \theta}{(\sigma; L) \xrightarrow{(\theta,\mathsf{rel}(l))} (\sigma; L_{|l})}}{P \mid (\sigma; L) \xrightarrow{(\theta,\mathsf{rel}(l))} P[\theta \mapsto e'] \mid (\sigma; L_{|l})}$$

in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} S$ we have $(\theta \mapsto e) \in T$ and $(l \mapsto \theta) \in S$ (by Rel^{-1}), and we consider the

label $\langle(\theta, \mathsf{rel}(l)), \phi, (\theta, \mathsf{rel}(l))\rangle$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{rel}(l))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{rel}(l))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{rel}(l))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \dfrac{}{(l \mapsto \theta) \xrightarrow{(\theta,\mathsf{rel}(l))} (l \mapsto \circ)}}{(l \mapsto \theta) \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{rel}(l))} (l \mapsto \circ) \mid \mathcal{S}} \quad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad \langle((\theta, \mathsf{rel}(l)), \phi, (\theta, \mathsf{rel}(l)))\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \theta) \mid \mathcal{S} \xrightarrow{((\theta,\mathsf{rel}(l)),\phi,(\theta,\mathsf{rel}(l)))} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \circ) \mid \mathcal{S}}$$

we can easily see that the final states are still related.

□

B.2 Write Buffering

Theorem 9. $P \mid (\sigma; L; B) \sim T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$

$$\begin{array}{ccc} P \mid (\sigma; L; B) & \underset{Rel^{-1}}{\overset{Rel}{\longleftrightarrow}} & T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}) \\ \varphi(\alpha)\downarrow & & \downarrow\alpha \\ P' \mid (\sigma'; L'; B') & \underset{Rel^{-1}}{\overset{Rel}{\longleftrightarrow}} & T' \otimes_{\mathcal{R}} M' \otimes_{\mathcal{R}} (\mathcal{L}' \mid \mathcal{B}') \end{array}$$

Proof. Firstly, we need to define the relation (Rel) between the states of $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ and of $P \mid (\sigma; L; B)$, where

$$\begin{gathered} Rel = \{(T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}); P \mid (\sigma; L; B)) \mid \forall\langle\theta, e\rangle \in T \text{ then } P(\theta) = e; \\ \forall(r \mapsto v) \in M \text{ then } \sigma(r) = v; \\ \forall(r \mapsto \circ) \in M \text{ then } r \notin \mathsf{dom}(\sigma); \\ \forall(l \mapsto \theta) \in \mathcal{L} \text{ then } L(l) = \theta; \\ \forall(l \mapsto \circ) \in \mathcal{L} \text{ then } l \notin \mathsf{dom}(L); \\ B = \mathcal{B}\} \end{gathered}$$

Now we show by case that for each transitions done by $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ then the $P \mid (\sigma; L; B)$ can do the same transitions. Now we proceed by cases on the labels:

- We consider the case where $\varphi(\langle(\theta, \mathsf{pure}), \phi, \phi\rangle) = (\theta, \mathsf{pure})$, then in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{\mathsf{pure}} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{pure})} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{pure})} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad (\mathcal{L} \mid \mathcal{B}) \xrightarrow{\phi} (\mathcal{L} \mid \mathcal{B}) \qquad \langle((\theta, \mathsf{pure}), \phi, \phi)\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}) \xrightarrow{((\theta,\mathsf{pure}),\phi,\phi)} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})}$$

in $P \mid (\sigma; L; B)$ we have $P[\theta \mapsto e]$ (by *Rel*), and we consider the label (θ, pure). Then, we obtain:

$$\textsc{Program}\ \frac{\textsc{PrgmStep}\ \dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{pure}} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{pure})} P[\theta \mapsto e']}}{P \mid (\sigma; L; B) \xrightarrow{(\theta,\mathsf{pure})} P[\theta \mapsto e'] \mid (\sigma; L; B)}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi(\langle(\theta, \mathsf{sp}(\theta')), \phi, (\theta, \mathsf{sp}(\theta'))\rangle) = (\theta, \mathsf{sp}(\theta'))$, then in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{\mathsf{sp}(\theta')} e'; e''}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \langle\theta, e'\rangle \mid \langle\theta', e''\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \langle\theta, e'\rangle \mid \langle\theta', e''\rangle \mid \mathcal{T}} \qquad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad \dfrac{\dfrac{\forall r. \mathcal{B}(\theta, r) = \bullet}{\mathcal{B} \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \mathcal{B}}}{\mathcal{L} \mid \mathcal{B} \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \mathcal{L} \mid \mathcal{B}} \qquad \langle((\theta, \mathsf{sp}(\theta')), \phi, \phi)\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}) \xrightarrow{((\theta,\mathsf{sp}(\theta')),\phi,\phi)} \langle\theta, e'\rangle \mid \langle\theta', e''\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})}$$

in $P \mid (\sigma; L; B)$ we have $P[\theta \mapsto e]$ and $\forall r. B(\theta, r) = \bullet$ (by *Rel*), and we consider the label $(\theta, \mathsf{sp}(\theta'))$. Then, we obtain:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad \theta' \notin \mathsf{dom}(P) \qquad e \xhookrightarrow{\mathsf{sp}(\theta')} e'; e''}{P \xrightarrow{(\theta,\mathsf{sp}(\theta'))} P[\theta \mapsto e'][\theta' \mapsto e'']} \qquad \dfrac{\forall r. B(\theta, r) = \bullet}{(\sigma; L; B) \xrightarrow{(\theta,\mathsf{sp}(\theta'))} (\sigma; L; B)}}{P \mid (\sigma; L; B) \xrightarrow{(\theta,\mathsf{sp}(\theta'))} P[\theta \mapsto e'][\theta' \mapsto e''] \mid (\sigma; L; B)}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi(\langle(\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l))\rangle) = (\theta, \mathsf{acq}(l))$, then in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{acq}(l))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta, e'\rangle \mid \mathcal{T}} \qquad \dfrac{\dfrac{}{(l \mapsto \circ) \xrightarrow{(\theta,\mathsf{acq}(l))} (l \mapsto \theta)}}{(l \mapsto \circ) \mid \mathcal{L} \mid \mathcal{B} \xrightarrow{(\theta,\mathsf{acq}(l))} (l \mapsto \theta) \mid \mathcal{L} \mid \mathcal{B}} \qquad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad \langle((\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l)))\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \circ) \mid \mathcal{L} \mid \mathcal{B} \xrightarrow{((\theta,\mathsf{acq}(l)),\phi,(\theta,\mathsf{acq}(l)))} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \theta) \mid \mathcal{L} \mid \mathcal{B}}$$

in $P \mid (\sigma; L; B)$ we have $P[\theta \mapsto e]$ and $l \notin \mathsf{dom}(L)$ (by *Rel*), and we consider the label

$(\theta, \mathsf{acq}(l))$. Then, we obtain:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{acq}(l)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{acq}(l))} P[\theta \mapsto e']} \qquad \dfrac{l \notin dom(L)}{(\sigma; L; B) \xrightarrow{(\theta,\mathsf{acq}(l))} (\sigma, L[l \mapsto \theta])}}{P \mid (\sigma; L; B) \xrightarrow{(\theta,\mathsf{acq}(l))} P[\theta \mapsto e'] \mid (\sigma; L[l \mapsto \theta]; B)}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi(\langle(\theta, \mathsf{rel}(l)), \phi, (\theta, \mathsf{rel}(l))\rangle) = (\theta, \mathsf{rel}(l))$, then in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{rel}(l))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{rel}(l))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{rel}(l))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \dfrac{\dfrac{\forall r. \mathcal{B}(\theta, r) = \bullet}{(l \mapsto \theta) \mid \mathcal{B} \xrightarrow{(\theta,\mathsf{rel}(l))} (l \mapsto \circ) \mid \mathcal{B}}}{(l \mapsto \theta) \mid \mathcal{L} \mid \mathcal{B} \xrightarrow{(\theta,\mathsf{rel}(l))} (l \mapsto \circ) \mid \mathcal{L} \mid \mathcal{B}} \quad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \quad \langle((\theta, \mathsf{rel}(l)), \phi, (\theta, \mathsf{rel}(l)))\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \theta) \mid \mathcal{L} \mid \mathcal{B} \xrightarrow{((\theta,\mathsf{rel}(l)),\phi,(\theta,\mathsf{rel}(l)))} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \circ) \mid \mathcal{L} \mid \mathcal{B}}$$

in $P \mid (\sigma; L; B)$ we have $P[\theta \mapsto e]$, $L[l \mapsto \theta]$ and $\forall r. B(\theta, r) = \bullet$ (by Rel), and we consider the label $(\theta, \mathsf{rel}(l))$. Then, we obtain:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{rel}(l)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{rel}(l))} P[\theta \mapsto e']} \qquad \dfrac{L(l) = \theta \qquad \forall r. B(\theta, r) = \bullet}{(\sigma; L; B) \xrightarrow{(\theta,\mathsf{rel}(l))} (\sigma; L_{|l}; B)}}{P \mid (\sigma; L; B) \xrightarrow{(\theta,\mathsf{rel}(l))} P[\theta \mapsto e'] \mid (\sigma; L_{|l}; B)}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi(\langle(\theta, \mathsf{ref}(r, v)), (\theta, \mathsf{ref}(r, v)), \phi\rangle) = (\theta, \mathsf{ref}(r, v))$, then in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{ref}(r,v))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{ref}(r,v))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{ref}(r,v))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \dfrac{\dfrac{}{(r \mapsto \circ) \xrightarrow{(\theta,\mathsf{ref}(r,v))} (r \mapsto v)}}{(r \mapsto \circ) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{ref}(r,v))} (r \mapsto v) \mid \mathcal{M}} \quad (\mathcal{L} \mid \mathcal{B}) \xrightarrow{\phi} (\mathcal{L} \mid \mathcal{B}) \quad \langle\langle(\theta, \mathsf{ref}(r, v)), (\theta, \mathsf{ref}(r, v)), \phi\rangle\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto \circ) \mid \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}) \xrightarrow{\langle(\theta,\mathsf{ref}(r,v)),(\theta,\mathsf{ref}(r,v)),\phi\rangle} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})}$$

in $P \mid (\sigma; L; B)$ we have $P[\theta \mapsto e]$ and $r \notin \mathsf{dom}(\sigma)$ (by Rel), and we consider the label

$(\theta, \mathsf{ref}(r,v))$. Then, we obtain:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{ref}(r,v)} e';\cdot}{P \xrightarrow{(\theta,\mathsf{ref}(r,v))} P[\theta \mapsto e']} \qquad \dfrac{r \notin \mathsf{dom}(\sigma)}{(\sigma; L; B) \xrightarrow{(\theta,\mathsf{ref}(r,v))} (\sigma[r \mapsto v]; L; B)}}{P \mid (\sigma; L; B) \xrightarrow{(\theta,\mathsf{ref}(r,v))} P[\theta \mapsto e'] \mid (\sigma[r \mapsto v]; L; B)}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi(\langle(\theta, \mathsf{rdM}(r,v)), (\theta, \mathsf{rdM}(r,v)), (\theta, \mathsf{rdM}(r,v))\rangle) = (\theta, \mathsf{rd}(r,v))$, then in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{rd}(r,v))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{rdM}(r,v))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{rdM}(r,v))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \dfrac{}{(r \mapsto v) \xrightarrow{(\theta,\mathsf{rdM}(r,v))} (r \mapsto v)}}{(r \mapsto v) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{rdM}(r,v))} (r \mapsto v) \mid \mathcal{M}} \quad \dfrac{\dfrac{\mathcal{B}(\theta, r) = \bullet}{\mathcal{B} \xrightarrow{(\theta,\mathsf{rdM}(r,v))} \mathcal{B}}}{\mathcal{L} \mid \mathcal{B} \xrightarrow{(\theta,\mathsf{rdM}(r,v))} \mathcal{L} \mid \mathcal{B}} \quad \langle((\theta, \mathsf{rdM}(r,v)), (\theta, \mathsf{rdM}(r,v)), (\theta, \mathsf{rdM}(r,v)))\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}) \xrightarrow{((\theta,\mathsf{rdM}(r,v)),(\theta,\mathsf{rdM}(r,v)),(\theta,\mathsf{rdM}(r,v)))} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})}$$

in $P \mid (\sigma; L; B)$ we have $P[\theta \mapsto e]$, $\sigma[r \mapsto v]$ and $B(\theta, r) = \bullet$ (by *Rel*), and we consider the label $(\theta, \mathsf{rd}(r,v))$. Then, we obtain:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{rd}(r,v)} e';\cdot}{P \xrightarrow{(\theta,\mathsf{rd}(r,v))} P[\theta \mapsto e']} \qquad \dfrac{\sigma(r) = v \qquad B(\theta, r) = \bullet}{(\sigma; L; B) \xrightarrow{(\theta,\mathsf{rd}(r,v))} (\sigma; L; B)}}{P \mid (\sigma; L; B) \xrightarrow{(\theta,\mathsf{rd}(r,v))} P[\theta \mapsto e'] \mid (\sigma; L; B)}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi(\langle(\theta, \mathsf{rdB}(r,v)), (\theta, \mathsf{rdB}(r,v)), (\theta, \mathsf{rdB}(r,v))\rangle) = (\theta, \mathsf{rd}(r,v))$, then in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{rd}(r,v))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{rdB}(r,v))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{rdB}(r,v))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \dfrac{}{(r \mapsto v_1) \xrightarrow{(\theta,\mathsf{rdB}(r,v))} (r \mapsto v_1)}}{(r \mapsto v_1) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{rdB}(r,v))} (r \mapsto v_1) \mid \mathcal{M}} \quad \dfrac{\dfrac{\mathcal{B}(\theta, r) = q, v}{\mathcal{B} \xrightarrow{(\theta,\mathsf{rdB}(r,v))} \mathcal{B}}}{\mathcal{L} \mid \mathcal{B} \xrightarrow{(\theta,\mathsf{rdB}(r,v))} \mathcal{L} \mid \mathcal{B}} \quad \langle((\theta, \mathsf{rdB}(r,v)), (\theta, \mathsf{rdB}(r,v)), (\theta, \mathsf{rdB}(r,v)))\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v_1) \mid \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}) \xrightarrow{((\theta,\mathsf{rdB}(r,v)),(\theta,\mathsf{rdB}(r,v)),(\theta,\mathsf{rdB}(r,v)))} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v_1) \mid \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})}$$

in $P \mid (\sigma; L; B)$ we have $P[\theta \mapsto e]$, $\sigma[r \mapsto v_1]$ and $B(\theta, r) = q, v$ (by *Rel*), and we consider the label $(\theta, \mathsf{rd}(r, v))$. Then, we obtain:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{rd}(r,v)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{rd}(r,v))} P[\theta \mapsto e']} \qquad \dfrac{B(\theta, r) = q, v}{(\sigma; L; B) \xrightarrow{(\theta,\mathsf{rd}(r,v))} (\sigma; L; B)}}{P \mid (\sigma; L; B) \xrightarrow{(\theta,\mathsf{rd}(r,v))} P[\theta \mapsto e'] \mid (\sigma; L; B)}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi(\langle(\theta, \mathsf{wr}(r, v)), \phi, (\theta, \mathsf{wr}(r, v))\rangle) = (\theta, \mathsf{wr}(r, v))$, then in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{wr}(r,v))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{wr}(r,v))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{wr}(r,v))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad \dfrac{\dfrac{\mathcal{B}(\theta, r) = q}{\mathcal{B} \xrightarrow{(\theta,\mathsf{wr}(r,v))} \mathcal{B}[(\theta, r) \mapsto q, v]}}{\mathcal{L} \mid \mathcal{B} \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \mathcal{L} \mid \mathcal{B}[(\theta, r) \mapsto q, v]} \quad \langle((\theta, \mathsf{wr}(r, v)), (\theta, \mathsf{wr}(r, v)), \phi)\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}) \xrightarrow{((\theta,\mathsf{wr}(r,v)),(\theta,\mathsf{wr}(r,v)),\phi)} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}[(\theta, r) \mapsto q, v])}$$

in $P \mid (\sigma; L; B)$ we have $P[\theta \mapsto e]$ and $B[(\theta, r) \mapsto q]$ (by *Rel*), and we consider the label $(\theta, \mathsf{wr}(r, v))$. Then, we obtain:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{wr}(r,v)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{wr}(r,v))} P[\theta \mapsto e']} \qquad \dfrac{B[(\theta, r) \mapsto q]}{(\sigma; L) \xrightarrow{(\theta,\mathsf{wr}(r,v))} (\sigma; L; B[(\theta, r) \mapsto q, v])}}{P \mid (\sigma; L; B) \xrightarrow{(\theta,\mathsf{wr}(r,v))} P[\theta \mapsto e'] \mid (\sigma; L; B[(\theta, r) \mapsto q, v])}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi(\langle\phi, (\theta, \mathsf{mem}(r, v)), (\theta, \mathsf{mem}(r, v))\rangle) = \epsilon$, then in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have that:

$$\textsc{Sync}\ \frac{\mathcal{T} \xrightarrow{\phi} \mathcal{T} \qquad \dfrac{}{(r \mapsto v_{old}) \xrightarrow{(\theta,\mathsf{mem}(r,v))} (r \mapsto v)} \qquad \dfrac{\dfrac{\mathcal{B}(\theta, r) = v, q}{\mathcal{B} \xrightarrow{\mathsf{mem}(r,v)} \mathcal{B}[(\theta, r) \mapsto q]}}{\mathcal{L} \mid \mathcal{B} \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \mathcal{L} \mid \mathcal{B}[(\theta, r) \mapsto q]} \quad \langle\phi, (\theta, \mathsf{mem}(r, v)), (\theta, \mathsf{mem}(r, v))\rangle \in \mathcal{R}}{\mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v_{old}) \mid \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}) \xrightarrow{((\theta,\mathsf{wr}(r,v)),(\theta,\mathsf{wr}(r,v)),\phi)} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})}$$

in $P \mid (\sigma; L; B)$ we have $P[\theta \mapsto e]$, $\sigma[r \mapsto v_{old}]$ and $B(\theta, r) = v, q$(by *Rel*), and we consider

the label ϵ. Then, we obtain:

$$\textsc{Heap}\ \frac{\dfrac{B(\theta,r) = v, q}{(\sigma; L; B) \xrightarrow{\epsilon} (\sigma[r \mapsto v]; L; B[(\theta,r) \mapsto q])}}{P \mid (\sigma; L; B) \xrightarrow{\epsilon} P[\theta \mapsto e'] \mid ((\sigma[r \mapsto v]; L; B[(\theta,r) \mapsto q])}$$

we can easily see that the final states are still related.

Now we consider the case starting from $P \mid (\sigma; L; B)$ and we consider then Rel^{-1} and φ^{-1}. We proceed by the case for each possible labels:

- We consider the case where $\varphi^{-1}((\theta, \mathsf{pure})) = \langle(\theta, \mathsf{pure}), \phi, \phi\rangle$, then in $P \mid (\sigma; L; B)$ we have that:

$$\textsc{Program}\ \frac{\textsc{PrgmStep}\ \dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{pure}} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{pure})} P[\theta \mapsto e']}}{P \mid (\sigma; L; B) \xrightarrow{(\theta,\mathsf{pure})} P[\theta \mapsto e'] \mid (\sigma; L; B)}$$

in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have $\langle\theta, e\rangle \in \mathcal{T}$ (by Rel^{-1}), and we consider the label $\langle(\theta, \mathsf{pure}), \phi, \phi\rangle$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{\mathsf{pure}} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{pure})} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{pure})} \langle\theta, e'\rangle \mid \mathcal{T}} \qquad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad (\mathcal{L} \mid \mathcal{B}) \xrightarrow{\phi} (\mathcal{L} \mid \mathcal{B}) \qquad \langle((\theta, \mathsf{pure}), \phi, \phi)\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}) \xrightarrow{((\theta,\mathsf{pure}),\phi,\phi)} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi^{-1}((\theta, \mathsf{sp}(\theta'))) = \langle(\theta, \mathsf{sp}(\theta')), \phi, (\theta, \mathsf{sp}(\theta'))\rangle$, then in $P \mid (\sigma; L; B)$ we have that:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad \theta' \notin \mathsf{dom}(P) \qquad e \xhookrightarrow{\mathsf{sp}(\theta')} e'; e''}{P \xrightarrow{(\theta,\mathsf{sp}(\theta'))} P[\theta \mapsto e'][\theta' \mapsto e'']} \qquad \frac{\forall r. B(\theta, r) = \bullet}{(\sigma; L; B) \xrightarrow{(\theta,\mathsf{sp}(\theta'))} (\sigma; L; B)}}{P \mid (\sigma; L; B) \xrightarrow{(\theta,\mathsf{sp}(\theta'))} P[\theta \mapsto e'][\theta' \mapsto e''] \mid (\sigma; L; B)}$$

in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have $\langle\theta, e\rangle \in \mathcal{T}$ and $\forall r. \mathcal{B}(\theta, r) = \bullet$ (by Rel^{-1}), and we consider

the label $\langle(\theta, \mathsf{sp}(\theta')), \phi, (\theta, \mathsf{sp}(\theta'))\rangle$. Then, we obtain:

$$\textsc{Sync}\ \frac{\mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad \dfrac{\dfrac{e \xhookrightarrow{\mathsf{sp}(\theta')} e'; e''}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \langle\theta, e'\rangle \mid \langle\theta', e''\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \langle\theta, e'\rangle \mid \langle\theta', e''\rangle \mid \mathcal{T}} \qquad \dfrac{\forall r.\mathcal{B}(\theta, r) = \bullet}{\mathcal{B} \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \mathcal{B}} \qquad \langle((\theta, \mathsf{sp}(\theta')), \phi, \phi)\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}) \xrightarrow{((\theta,\mathsf{sp}(\theta')),\phi,\phi)} \langle\theta, e'\rangle \mid \langle\theta', e''\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi^{-1}((\theta, \mathsf{acq}(l))) = \langle(\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l))\rangle$, then in $P \mid (\sigma; L; B)$ we have that:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{acq}(l)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{acq}(l))} P[\theta \mapsto e']} \qquad \dfrac{l \notin dom(L)}{(\sigma; L; B) \xrightarrow{(\theta,\mathsf{acq}(l))} (\sigma, L[l \mapsto \theta])}}{P \mid (\sigma; L; B) \xrightarrow{(\theta,\mathsf{acq}(l))} P[\theta \mapsto e'] \mid (\sigma; L[l \mapsto \theta]; B)}$$

in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have $\langle\theta, e\rangle \in \mathcal{T}$ and $(l \mapsto \circ) \in \mathcal{L}$ (by Rel^{-1}), and we consider the label $\langle(\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l))\rangle$. Then, we obtain:

$$\textsc{Sync}\ \frac{\mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad \dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{acq}(l))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta, e'\rangle \mid \mathcal{T}} \qquad \dfrac{\dfrac{}{(l \mapsto \circ) \xrightarrow{(\theta,\mathsf{acq}(l))} (l \mapsto \theta)}}{(l \mapsto \circ) \mid \mathcal{L} \mid \mathcal{B} \xrightarrow{(\theta,\mathsf{acq}(l))} (l \mapsto \theta) \mid \mathcal{L} \mid \mathcal{B}} \qquad \langle((\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l)))\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \circ) \mid \mathcal{L} \mid \mathcal{B} \xrightarrow{((\theta,\mathsf{acq}(l)),\phi,(\theta,\mathsf{acq}(l)))} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \theta) \mid \mathcal{L} \mid \mathcal{B}}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi^{-1}((\theta, \mathsf{rel}(l))) = \langle(\theta, \mathsf{rel}(l)), \phi, (\theta, \mathsf{rel}(l))\rangle$, then in $P \mid (\sigma; L; B)$ we have that:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{rel}(l)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{rel}(l))} P[\theta \mapsto e']} \qquad \dfrac{L(l) = \theta \qquad \forall r.B(\theta, r) = \bullet}{(\sigma; L; B) \xrightarrow{(\theta,\mathsf{rel}(l))} (\sigma; L_{|l}; B)}}{P \mid (\sigma; L; B) \xrightarrow{(\theta,\mathsf{rel}(l))} P[\theta \mapsto e'] \mid (\sigma; L_{|l}; B)}$$

in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have $\langle\theta, e\rangle \in \mathcal{T}$, $(l \mapsto \theta) \in \mathcal{L}$ and $\forall r.\mathcal{B}(\theta, r) = \bullet$ (by Rel^{-1}), and

we consider the label $\langle(\theta, \mathsf{rel}(l)), \phi, (\theta, \mathsf{rel}(l))\rangle$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{rel}(l))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{rel}(l))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{rel}(l))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \dfrac{\dfrac{\forall r. \mathcal{B}(\theta, r) = \bullet}{(l \mapsto \theta) \mid \mathcal{B} \xrightarrow{(\theta,\mathsf{rel}(l))} (l \mapsto \circ) \mid \mathcal{B}}}{(l \mapsto \theta) \mid \mathcal{L} \mid \mathcal{B} \xrightarrow{(\theta,\mathsf{rel}(l))} (l \mapsto \circ) \mid \mathcal{L} \mid \mathcal{B}} \quad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \quad \langle((\theta, \mathsf{rel}(l)), \phi, (\theta, \mathsf{rel}(l)))\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \theta) \mid \mathcal{L} \mid \mathcal{B} \xrightarrow{((\theta,\mathsf{rel}(l)),\phi,(\theta,\mathsf{rel}(l)))} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \circ) \mid \mathcal{L} \mid \mathcal{B}}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi^{-1}((\theta, \mathsf{ref}(r, v))) = \langle(\theta, \mathsf{ref}(r, v)), (\theta, \mathsf{ref}(r, v)), \phi\rangle$, then in $P \mid (\sigma; L; B)$ we have that:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{ref}(r,v)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{ref}(r,v))} P[\theta \mapsto e']} \qquad \dfrac{r \notin \mathsf{dom}(\sigma)}{(\sigma; L; B) \xrightarrow{(\theta,\mathsf{ref}(r,v))} (\sigma[r \mapsto v]; L; B)}}{P \mid (\sigma; L; B) \xrightarrow{(\theta,\mathsf{ref}(r,v))} P[\theta \mapsto e'] \mid (\sigma[r \mapsto v]; L; B)}$$

in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have $\langle\theta, e\rangle \in \mathcal{T}$ and $(r \mapsto \circ) \in M$ (by Rel^{-1}), and we consider the label $\langle(\theta, \mathsf{ref}(r, v)), (\theta, \mathsf{ref}(r, v)), \phi\rangle$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{ref}(r,v))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{ref}(r,v))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{ref}(r,v))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \dfrac{\dfrac{}{(r \mapsto \circ) \xrightarrow{(\theta,\mathsf{ref}(r,v))} (r \mapsto v)}}{(r \mapsto \circ) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{ref}(r,v))} (r \mapsto v) \mid \mathcal{M}} \quad (\mathcal{L} \mid \mathcal{B}) \xrightarrow{\phi} (\mathcal{L} \mid \mathcal{B}) \quad \langle\langle(\theta, \mathsf{ref}(r, v)), (\theta, \mathsf{ref}(r, v)), \phi\rangle\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto \circ) \mid \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}) \xrightarrow{\langle(\theta,\mathsf{ref}(r,v)),(\theta,\mathsf{ref}(r,v)),\phi\rangle} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})}$$

we can easily see that the final states are still related.

- We consider the case where the label is $(\theta, \mathsf{rd}(r, v))$ and we have two possible cases:

  - when the buffer is empty then when $B(\theta, r) = \bullet$ and we read from the memory, in this case we have $\varphi^{-1}((\theta, \mathsf{rd}(r, v))) = \langle(\theta, \mathsf{rdM}(r, v)), (\theta, \mathsf{rdM}(r, v)), (\theta, \mathsf{rdM}(r, v))\rangle$, then in $P \mid (\sigma; L; B)$ we have that:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{rd}(r,v)} e'; \cdot}{P \xrightarrow{(\theta,\mathsf{rd}(r,v))} P[\theta \mapsto e']} \qquad \dfrac{\sigma(r) = v \qquad B(\theta, r) = \bullet}{(\sigma; L; B) \xrightarrow{(\theta,\mathsf{rd}(r,v))} (\sigma; L; B)}}{P \mid (\sigma; L; B) \xrightarrow{(\theta,\mathsf{rd}(r,v))} P[\theta \mapsto e'] \mid (\sigma; L; B)}$$

in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have $\langle\theta, e\rangle \in \mathcal{T}$, $(r \mapsto v) \in M$ and $\mathcal{B}(\theta, r) = \bullet$ (by Rel^{-1}), and we consider the label $\langle(\theta, \mathsf{rdM}(r, v)), (\theta, \mathsf{rdM}(r, v)), (\theta, \mathsf{rdM}(r, v))\rangle$. Then,

we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{rd}(r,v))} e'}{\langle\theta,e\rangle \xrightarrow{(\theta,\mathsf{rdM}(r,v))} \langle\theta,e'\rangle}}{\langle\theta,e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{rdM}(r,v))} \langle\theta,e'\rangle \mid \mathcal{T}} \quad \frac{}{(r\mapsto v) \xrightarrow{(\theta,\mathsf{rdM}(r,v))} (r\mapsto v)}}{(r\mapsto v) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{rdM}(r,v))} (r\mapsto v) \mid \mathcal{M}} \quad \frac{\dfrac{\mathcal{B}(\theta,r) = \bullet}{\mathcal{B} \xrightarrow{(\theta,\mathsf{rdM}(r,v))} \mathcal{B}}}{\mathcal{L} \mid \mathcal{B} \xrightarrow{(\theta,\mathsf{rdM}(r,v))} \mathcal{L} \mid \mathcal{B}} \quad \langle((\theta,\mathsf{rdM}(r,v)),(\theta,\mathsf{rdM}(r,v)),(\theta,\mathsf{rdM}(r,v)))\rangle \in \mathcal{R}}{\langle\theta,e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r\mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}) \xrightarrow{((\theta,\mathsf{rdM}(r,v)),(\theta,\mathsf{rdM}(r,v)),(\theta,\mathsf{rdM}(r,v)))} \langle\theta,e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r\mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})}$$

we can easily see that the final states are still related.

– when the buffer is not empty then when $B(\theta,r) \neq \bullet$ and we read from the buffer in this case we have $\varphi^{-1}((\theta,\mathsf{rd}(r,v))) = \langle(\theta,\mathsf{rdB}(r,v)),(\theta,\mathsf{rdB}(r,v)),(\theta,\mathsf{rdB}(r,v))\rangle$, then in $P \mid (\sigma;L;B)$ we have that:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{rd}(r,v)} e';\cdot}{P \xrightarrow{(\theta,\mathsf{rd}(r,v))} P[\theta\mapsto e']} \qquad \dfrac{B(\theta,r) = q,v}{(\sigma;L;B) \xrightarrow{(\theta,\mathsf{rd}(r,v))} (\sigma;L;B)}}{P \mid (\sigma;L;B) \xrightarrow{(\theta,\mathsf{rd}(r,v))} P[\theta\mapsto e'] \mid (\sigma;L;B)}$$

in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have $\langle\theta,e\rangle \in \mathcal{T}$, and $\mathcal{B}(\theta,r) = q,v$ (by Rel^{-1}), and we consider the label $\langle(\theta,\mathsf{rdB}(r,v)),(\theta,\mathsf{rdB}(r,v)),(\theta,\mathsf{rdB}(r,v))\rangle$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{rd}(r,v))} e'}{\langle\theta,e\rangle \xrightarrow{(\theta,\mathsf{rdB}(r,v))} \langle\theta,e'\rangle}}{\langle\theta,e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{rdB}(r,v))} \langle\theta,e'\rangle \mid \mathcal{T}} \quad \frac{}{(r\mapsto v_1) \xrightarrow{(\theta,\mathsf{rdB}(r,v))} (r\mapsto v_1)}}{(r\mapsto v_1) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{rdB}(r,v))} (r\mapsto v_1) \mid \mathcal{M}} \quad \frac{\dfrac{\mathcal{B}(\theta,r) = q,v}{\mathcal{B} \xrightarrow{(\theta,\mathsf{rdB}(r,v))} \mathcal{B}}}{\mathcal{L} \mid \mathcal{B} \xrightarrow{(\theta,\mathsf{rdB}(r,v))} \mathcal{L} \mid \mathcal{B}} \quad \langle((\theta,\mathsf{rdB}(r,v)),(\theta,\mathsf{rdB}(r,v)),(\theta,\mathsf{rdB}(r,v)))\rangle \in \mathcal{R}}{\langle\theta,e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r\mapsto v_1) \mid \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}) \xrightarrow{((\theta,\mathsf{rdB}(r,v)),(\theta,\mathsf{rdB}(r,v)),(\theta,\mathsf{rdB}(r,v)))} \langle\theta,e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r\mapsto v_1) \mid \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi^{-1}((\theta,\mathsf{wr}(r,v))) = \langle(\theta,\mathsf{wr}(r,v)),\phi,(\theta,\mathsf{wr}(r,v))\rangle$, then in $P \mid (\sigma;L;B)$ we have that:

$$\textsc{Mutual}\ \frac{\dfrac{P(\theta) = e \qquad e \xhookrightarrow{\mathsf{wr}(r,v)} e';\cdot}{P \xrightarrow{(\theta,\mathsf{wr}(r,v))} P[\theta\mapsto e']} \qquad \dfrac{B[(\theta,r)\mapsto q]}{(\sigma;L) \xrightarrow{(\theta,\mathsf{wr}(r,v))} (\sigma;L;B[(\theta,r)\mapsto q,v])}}{P \mid (\sigma;L;B) \xrightarrow{(\theta,\mathsf{wr}(r,v))} P[\theta\mapsto e'] \mid (\sigma;L;B[(\theta,r)\mapsto q,v])}$$

in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have $\langle \theta, e \rangle \in \mathcal{T}$ and $\mathcal{B}[(\theta, r) \mapsto q]$ (by Rel^{-1}), and we consider the label $\langle (\theta, \mathsf{wr}(r, v)), \phi, (\theta, \mathsf{wr}(r, v)) \rangle$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{wr}(r,v))} e'}{\langle \theta, e \rangle \xrightarrow{(\theta,\mathsf{wr}(r,v))} \langle \theta, e' \rangle}}{\langle \theta, e \rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{wr}(r,v))} \langle \theta, e' \rangle \mid \mathcal{T}} \quad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad \dfrac{\dfrac{\mathcal{B}(\theta, r) = q}{\mathcal{B} \xrightarrow{(\theta,\mathsf{wr}(r,v))} \mathcal{B}[(\theta, r) \mapsto q, v]}}{\mathcal{L} \mid \mathcal{B} \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \mathcal{L} \mid \mathcal{B}[(\theta, r) \mapsto q, v]} \quad \langle ((\theta, \mathsf{wr}(r, v)), (\theta, \mathsf{wr}(r, v)), \phi) \rangle \in \mathcal{R}}{\langle \theta, e \rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}) \xrightarrow{((\theta,\mathsf{wr}(r,v)),(\theta,\mathsf{wr}(r,v)),\phi)} \langle \theta, e' \rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}[(\theta, r) \mapsto q, v])}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi^{-1}(\epsilon) = \langle \phi, (\theta, \mathsf{mem}(r, v)), (\theta, \mathsf{mem}(r, v)) \rangle$, then in $P \mid (\sigma; L; B)$ we have that:

$$\textsc{Heap}\ \frac{\dfrac{B(\theta, r) = v, q}{(\sigma; L; B) \xrightarrow{\epsilon} (\sigma[r \mapsto v]; L; B[(\theta, r) \mapsto q])}}{P \mid (\sigma; L; B) \xrightarrow{\epsilon} P[\theta \mapsto e'] \mid ((\sigma[r \mapsto v]; L; B[(\theta, r) \mapsto q])}$$

in $T \otimes_{\mathcal{R}} M \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})$ we have $\langle \theta, e \rangle \in \mathcal{T}$ and $\mathcal{B}(\theta, r) = v, q$(by Rel^{-1}), and we consider the label $\langle \phi, (\theta, \mathsf{mem}(r, v)), (\theta, \mathsf{mem}(r, v)) \rangle$. Then, we obtain:

$$\textsc{Sync}\ \frac{\mathcal{T} \xrightarrow{\phi} \mathcal{T} \quad \dfrac{}{(r \mapsto v_{old}) \xrightarrow{(\theta,\mathsf{mem}(r,v))} (r \mapsto v)} \qquad \dfrac{\dfrac{\mathcal{B}(\theta, r) = v, q}{\mathcal{B} \xrightarrow{\mathsf{mem}(r,v)} \mathcal{B}[(\theta, r) \mapsto q]}}{\mathcal{L} \mid \mathcal{B} \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \mathcal{L} \mid \mathcal{B}[(\theta, r) \mapsto q]} \quad \langle \phi, (\theta, \mathsf{mem}(r, v)), (\theta, \mathsf{mem}(r, v)) \rangle \in \mathcal{R}}{\mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v_{old}) \mid \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B}) \xrightarrow{((\theta,\mathsf{wr}(r,v)),(\theta,\mathsf{wr}(r,v)),\phi)} \langle \theta, e' \rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} (\mathcal{L} \mid \mathcal{B})}$$

we can easily see that the final states are still related.

□

B.3 Automatic Mutual Exclusion

B.3.1 Strong Semantics

In Fig. B.1 we show the original strong semantics for AME described in [16].

For a deeper discussion about the semantics we refer to [16].

Instead in Fig. B.2 and Fig. B.3 we refer to the updated semantics decorated with labels and with unique identifier for expressions. We can note that for each protected action we consider to

$$\textsc{Trans Appl P}\quad \langle \sigma, T, \mathcal{P}[(\lambda\ x.e)\ V]\rangle \longmapsto \langle \sigma, T, \mathcal{P}[e[V/x]]\rangle$$

$$\textsc{Trans Appl U}\quad \langle \sigma, T.\mathcal{U}[(\lambda\ x.e)\ V].T', \mathsf{unit}\rangle \longmapsto \langle \sigma, T.\mathcal{U}[e[V/x]].T', \mathsf{unit}\rangle$$

$$\textsc{Trans Ref P}\quad \langle \sigma, T, \mathcal{P}[\mathsf{ref}\ V]\rangle \longmapsto \langle \sigma[r \mapsto V], T, \mathcal{P}[r]\rangle \qquad \text{if } r \in RefLoc\ -\ dom(\sigma)$$

$$\textsc{Trans Ref U}\quad \langle \sigma, T.\mathcal{U}[\mathsf{ref}\ V].T', \mathsf{unit}\rangle \longmapsto \langle \sigma[r \mapsto V], T.\mathcal{U}[r].T', \mathsf{unit}\rangle \qquad \text{if } r \in RefLoc\ -\ dom(\sigma)$$

$$\textsc{Trans Deref P}\quad \langle \sigma, T, \mathcal{P}[!r]\rangle \longmapsto \langle \sigma, T, \mathcal{P}[V]\rangle \qquad \text{if } \sigma(r) = V$$

$$\textsc{Trans Deref U}\quad \langle \sigma, T.\mathcal{U}[!r].T', \mathsf{unit}\rangle \longmapsto \langle \sigma, T.\mathcal{U}[V].T', \mathsf{unit}\rangle \qquad \text{if } \sigma(r) = V$$

$$\textsc{Trans Set P}\quad \langle \sigma, T, \mathcal{P}[r := V]\rangle \longmapsto \langle \sigma[r \mapsto V], T, \mathcal{P}[\mathsf{unit}]\rangle$$

$$\textsc{Trans Set U}\quad \langle \sigma, T.\mathcal{U}[r := V].T', \mathsf{unit}\rangle \longmapsto \langle \sigma[r \mapsto V], T.\mathcal{U}[\mathsf{unit}].T', \mathsf{unit}\rangle$$

$$\textsc{Trans Async P}\quad \langle \sigma, T, \mathcal{P}[\mathsf{async}\ e]\rangle \longmapsto \langle \sigma, T.e, \mathcal{P}[\mathsf{unit}]\rangle$$

$$\textsc{Trans Async U}\quad \langle \sigma, T.\mathcal{U}[\mathsf{async}\ e].T', \mathsf{unit}\rangle \longmapsto \langle \sigma, T.e.\mathcal{U}[\mathsf{unit}].T', \mathsf{unit}\rangle$$

$$\textsc{Trans Block P}\quad \langle \sigma, T, \mathcal{P}[\mathsf{blockUntil\ true}]\rangle \longmapsto \langle \sigma, T, \mathcal{P}[\mathsf{unit}]\rangle$$

$$\textsc{Trans Block U}\quad \langle \sigma, T.\mathcal{U}[\mathsf{blockUntil\ true}].T', \mathsf{unit}\rangle \longmapsto \langle \sigma, T.\mathcal{U}[\mathsf{unit}].T', \mathsf{unit}\rangle$$

$$\textsc{Trans Unprotected}\quad \langle \sigma, T, \mathcal{P}[\mathsf{unprotected}\ e]\rangle \longmapsto \langle \sigma, T.\mathcal{P}[\mathsf{unprotected}\ e], \mathsf{unit}\rangle$$

$$\textsc{Trans Close}\quad \langle \sigma, T.\mathcal{E}[\mathsf{unprotected}\ V].T', \mathsf{unit}\rangle \longmapsto \langle \sigma, T.\mathcal{E}[V].T', \mathsf{unit}\rangle$$

$$\textsc{Trans Activate}\quad \langle \sigma, T.e.T', \mathsf{unit}\rangle \longmapsto \langle \sigma, T.T', e\rangle$$

Figure B.1: Strong Semantics Abadi [16]

transition one if the evaluated expression is evaluated to unit and one for the other cases. This is because we need to distinguish these cases but it does not change the result of the original semantics, in fact if we delete the labels and the unique identifiers for the expression we will obtain the semantics described in [16].

Theorem 10. $\langle \sigma_A, E, e\rangle \sim (T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma)$

$$\begin{array}{ccc} \langle \sigma_A, E, e\rangle & \xleftarrow[Rel^{-1}]{Rel} & (T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma) \\ {\scriptstyle \varphi(\alpha)}\downarrow & & \downarrow{\scriptstyle \alpha} \\ \langle \sigma'_A, E', e'\rangle & \xleftarrow[Rel^{-1}]{Rel} & (T' \otimes_{\mathcal{R}} S' \otimes_{\mathcal{R}} \sigma') \end{array}$$

Proof. We start by defining Rel and then show that it is a strong similarity relation, after which we will show that Rel^{-1} is also a strong similarity relation.

Given $\{\langle \sigma_A, E, (\theta, e)\rangle, \langle T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\}$ then this tuple is in Rel if:

- $\sigma = \sigma_A$
- $\forall e_i \in E$ exist θ_i such that $t_i \in T$ and $t_i.\mathsf{id} = \theta_i \wedge t_i.\mathsf{exp} = e_i$
- if $e = \mathsf{unit}$ then $S = \bullet$ otherwise (if $e \neq \mathsf{unit}$) exist $t \in T$ where $t.\mathsf{id} = \theta \wedge t.\mathsf{exp} = e$ and $S = \theta$

After the definition of Rel we start to give for each label the proof that there is a strong similarity between Abadi system and our system:

- if in Abadi.system we have this transition:

  $$\textsc{Trans Activate}$$
  $$\langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{no\ unit,act})} \langle \sigma_A, E.E', (\theta, e)\rangle$$

  in our system we have:

  $$\frac{\dfrac{\textsc{Activate1}\ \dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = e}{T \mid t \xrightarrow{(\theta,\mathsf{no\ unit,act})} T \mid t}\ e \neq \mathsf{unit} \qquad \textsc{Activate1}\ \dfrac{S = \bullet}{S \xrightarrow{(\theta,\mathsf{no\ unit,act})} \theta}}{T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{no\ unit,act}),\phi,(\theta,\mathsf{no\ unit,act})\rangle} T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma}$$

  We know that $\{\langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle, \langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$.

  Then after the transitions we have that $\{\langle \sigma_A, E.E', (\theta, e)\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

  This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in $E.E'$ have a correspond thread in T (none of the rules applied to the two systems changed either $E.E'$ or T) and the thread t correspond to e (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$).

$$\textsc{Trans Appl P}\quad \langle \sigma, T, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle \xmapsto{(\theta,\text{no unit,pure})} \langle \sigma, T, (\theta, \mathcal{P}[e[V/x]])\rangle$$

$$\textsc{Trans Appl P}\quad \langle \sigma, T, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle \xmapsto{(\theta,\text{unit,pure})} \langle \sigma, T, \text{unit}\rangle$$

$$\textsc{Trans Appl U}\quad \langle \sigma, T.(\theta, \mathcal{U}[(\lambda\ x.e)\ V]).T', \text{unit}\rangle \xmapsto{(\theta,\text{unp(pure)})} \langle \sigma, T.(\theta, \mathcal{U}[e[V/x]]).T', \text{unit}\rangle$$

$$\textsc{Trans Ref P}\quad \langle \sigma, T, (\theta, \mathcal{P}[\text{ref } V])\rangle \xmapsto{(\theta,\text{no unit,ref}(r,v))} \langle \sigma[r \mapsto V], T, (\theta, \mathcal{P}[r])\rangle \qquad \text{if } r \in RefLoc\ -\ dom(\sigma)$$

$$\textsc{Trans Ref P}\quad \langle \sigma, T, (\theta, \mathcal{P}[\text{ref } V])\rangle \xmapsto{(\theta,\text{unit,ref}(r,v))} \langle \sigma[r \mapsto V], T, \text{unit}\rangle \qquad \text{if } r \in RefLoc\ -\ dom(\sigma)$$

$$\textsc{Trans Ref U}\quad \langle \sigma, T.(\theta, \mathcal{U}[\text{ref } V]).T', \text{unit}\rangle \xmapsto{(\theta,\text{unp(ref}(r,v)))} \langle \sigma[r \mapsto V], T.(\theta, \mathcal{U}[r]).T', \text{unit}\rangle \quad \text{if } r \in RefLoc\ -\ dom(\sigma)$$

$$\textsc{Trans Deref P}\quad \langle \sigma, T, (\theta, \mathcal{P}[!r])\rangle \xmapsto{(\theta,\text{no unit,rd}(r,v))} \langle \sigma, T, (\theta, \mathcal{P}[V])\rangle \qquad \text{if } \sigma(r) = V$$

$$\textsc{Trans Deref P}\quad \langle \sigma, T, (\theta, \mathcal{P}[!r])\rangle \xmapsto{(\theta,\text{unit,rd}(r,v))} \langle \sigma, T, \text{unit}\rangle \qquad \text{if } \sigma(r) = V$$

$$\textsc{Trans Deref U}\quad \langle \sigma, T.(\theta, \mathcal{U}[!r]).T', \text{unit}\rangle \xmapsto{(\theta,\text{unp(rd}(r,v)))} \langle \sigma, T.(\theta, \mathcal{U}[V]).T', \text{unit}\rangle \qquad \text{if } \sigma(r) = V$$

$$\textsc{Trans Set P}\quad \langle \sigma, T, (\theta, \mathcal{P}[r := V])\rangle \xmapsto{(\theta,\text{no unit,wr}(r,v))} \langle \sigma[r \mapsto V], T, (\theta, \mathcal{P}[\text{unit}])\rangle$$

$$\textsc{Trans Set P}\quad \langle \sigma, T, (\theta, \mathcal{P}[r := V])\rangle \xmapsto{(\theta,\text{unit,wr}(r,v))} \langle \sigma[r \mapsto V], T, \text{unit}\rangle$$

$$\textsc{Trans Set U}\quad \langle \sigma, T.(\theta, \mathcal{U}[r := V]).T', \text{unit}\rangle \xmapsto{(\theta,\text{unp(wr}(r,v)))} \langle \sigma[r \mapsto V], T.(\theta, \mathcal{U}[\text{unit}]).T', \text{unit}\rangle$$

$$\textsc{Trans Async P}\quad \langle \sigma, T, (\theta, \mathcal{P}[\text{async } e])\rangle \xmapsto{(\theta,\text{no unit,async}(e))} \langle \sigma, T.(\theta', e), (\theta, \mathcal{P}[\text{unit}])\rangle$$

$$\textsc{Trans Async P}\quad \langle \sigma, T, (\theta, \mathcal{P}[\text{async } e])\rangle \xmapsto{(\theta,\text{unit,async}(e))} \langle \sigma, T.(\theta', e), \text{unit}\rangle$$

$$\textsc{Trans Async U}\quad \langle \sigma, T.(\theta, \mathcal{U}[\text{async } e]).T', \text{unit}\rangle \xmapsto{(\theta,\text{unp(async}(e)))} \langle \sigma, T.(\theta', e).(\theta, \mathcal{U}[\text{unit}]).T', \text{unit}\rangle$$

$$\textsc{Trans Block P}\quad \langle \sigma, T, (\theta, \mathcal{P}[\text{blockUntil true}])\rangle \xmapsto{(\theta,\text{no unit,blockUntil(true)})} \langle \sigma, T, (\theta, \mathcal{P}[\text{unit}])\rangle$$

$$\textsc{Trans Block P}\quad \langle \sigma, T, (\theta, \mathcal{P}[\text{blockUntil true}])\rangle \xmapsto{(\theta,\text{unit,blockUntil(true)})} \langle \sigma, T, \text{unit}\rangle$$

$$\textsc{Trans Block U}\quad \langle \sigma, T.(\theta, \mathcal{U}[\text{blockUntil true}]).T', \text{unit}\rangle \xmapsto{(\theta,\text{unp(blockUntil(true)))}} \langle \sigma, T.(\theta, \mathcal{U}[\text{unit}]).T', \text{unit}\rangle$$

Figure B.2: LTS Abadi strong sem. [16]

$$\textsc{Trans Unprotected}\quad \langle \sigma, T, (\theta, \mathcal{P}[\mathsf{unprotected}\ e])\rangle \xmapsto{(\theta,\mathsf{unprotect})} \langle \sigma, T.(\theta, \mathcal{P}[\mathsf{unprotected}\ e]), \mathsf{unit}\rangle$$

$$\textsc{Trans Close}\quad \langle \sigma, T.(\theta, \mathcal{E}[\mathsf{unprotected}\ V]).T', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{unp}(V))} \langle \sigma, T.(\theta, \mathcal{E}[V]).T', \mathsf{unit}\rangle$$

$$\textsc{Trans Activate}\quad \langle \sigma, T.(\theta, e).T', \mathsf{unit}\rangle \xmapsto{(\theta,\text{no unit,act})} \langle \sigma, T.T', (\theta, e)\rangle$$

$$\textsc{Trans Activate}\quad \langle \sigma, T.(\theta, e).T', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{unit,act})} \langle \sigma, T.T', \mathsf{unit}\rangle$$

Figure B.3: LTS Abadi strong sem, [16]

- if in Abadi.system we have this transition:

$$\textsc{Trans Activate}\quad \langle \sigma_A, E.(\theta, e).T', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{unit,act})} \langle \sigma_A, E.E', \mathsf{unit}\rangle$$

  in our system we have:

$$\frac{\dfrac{\textsc{Activate2}\quad \dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathsf{unit}}{T \mid t \xrightarrow{(\theta,\mathsf{unit,act})} T}\ e = \mathsf{unit} \qquad \dfrac{\textsc{Activate2}\quad S = \bullet}{S \xrightarrow{(\theta,\mathsf{unit,act})} S}}{T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit,act}),\phi,(\theta,\mathsf{unit,act})\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma}$$

  We know that $\{\langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle, \langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathsf{unit}$.

  Then after the transitions we have that $\{\langle \sigma_A, E.E', \mathsf{unit}\rangle, \langle T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

  This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in $E.E'$ have a correspond thread in T (none of the rules applied to the two systems changed either $E.E'$ or T) and $S = \bullet$ because $e = \mathsf{unit}$ (and in fact also $t.\mathsf{exp} = \mathsf{unit}$).

- if in Abadi.system we have this transition:

$$\textsc{Trans Appl P}\quad \langle \sigma_A, E, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle \xmapsto{(\theta,\text{no unit,pure})} \langle \sigma_A, E, (\theta, \mathcal{P}[e[V/x]])\rangle$$

  in our system we have:

$$\dfrac{\dfrac{\textsc{Step P1}\quad t.\mathsf{id}=\theta \qquad t.\mathsf{exp}=\mathcal{P}[(\lambda\ x.e)\ V] \qquad (\lambda\ x.e)\ V \xhookrightarrow{\mathsf{pure}} e[V/x]}{T \mid t \xrightarrow{(\theta,\text{no unit,pure})} T \mid t[exp \leftarrowtail \mathcal{P}[e[V/x]]]}\ \mathcal{P}[e[V/x]] \neq \mathsf{unit} \qquad \dfrac{\textsc{Step P1}\quad S=\theta}{S \xrightarrow{(\theta,\text{no unit,pure})} S}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\text{no unit,pure}),\phi,(\theta,\text{no unit,pure})\rangle} T \mid t[\mathsf{exp} \leftarrowtail \mathcal{P}[e[V/x]]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle\sigma_A, E, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[(\lambda\ x.e)\ V]$.

Then after the transitions we have that $\{\langle\sigma_A, E, (\theta, \mathcal{P}[e[V/x]])\rangle, \langle T \mid t[\mathsf{exp} \leftarrowtail \mathcal{P}[e[V/x]]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $\mathcal{P}[e[V/x]]$ (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[e[V/x]]$).

- if in Abadi.system we have this transition:

$$\textsc{Trans Appl P}\quad \langle\sigma_A, E, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{pure})} \langle\sigma_A, E, \mathsf{unit}\rangle$$

in our system we have:

$$\dfrac{\dfrac{\textsc{Step P2}\quad t.\mathsf{id}=\theta \qquad t.\mathsf{exp}=\mathcal{P}[(\lambda\ x.e)\ V] \qquad (\lambda\ x.e)\ V \xhookrightarrow{\mathsf{pure}} e[V/x]}{T \mid t \xrightarrow{(\theta,\mathsf{unit},\mathsf{pure})} T}\ \mathcal{P}[e[V/x]] = \mathsf{unit} \qquad \dfrac{\textsc{Step P2}\quad S=\theta}{S \xrightarrow{(\theta,\mathsf{unit},\mathsf{pure})} \bullet}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{pure}),\phi,(\theta,\mathsf{unit},\mathsf{pure})\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle\sigma_A, E, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[(\lambda\ x.e)\ V]$.

Then after the transitions we have that $\{\langle\sigma_A, E, \mathsf{unit}\rangle, \langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ because $\mathcal{P}[e[V/x]] = \mathsf{unit}$.

- if in Abadi.system we have this transition:

$$\frac{}{\langle \sigma_A, E.(\theta, \mathcal{U}[(\lambda\ x.e)\ V]).E', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{unp(pure)})} \langle \sigma_A, E.(\theta, \mathcal{U}[e[V/x]]).E', \mathsf{unit}\rangle}\ \textsc{Trans Appl U}$$

in our system we have:

$$\frac{\dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{U}[(\lambda\ x.e)\ V] \qquad (\lambda\ x.e)\ V \xhookrightarrow{\mathsf{pure}} e[V/x]}{T \mid t \xrightarrow{(\theta,\mathsf{unp(pure)})} T \mid t[exp \leftarrow \mathcal{U}[e[V/x]]]}\ \textsc{Step U} \qquad \frac{S = \bullet}{S \xrightarrow{(\theta,\mathsf{unp(pure)})} S}\ \textsc{Step U/Close}}{T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp(pure)}),\phi,(\theta,\mathsf{unp(pure)})\rangle} T \mid t[exp \leftarrow \mathcal{U}[e[V/x]]] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle \sigma_A, E.(\theta, \mathcal{U}[(\lambda\ x.e)\ V]).E', \mathsf{unit}\rangle, \langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[(\lambda\ x.e)\ V]$.

Then after the transitions we have that $\{\langle \sigma_A, E.(\theta, \mathcal{U}[e[V/x]]).E', \mathsf{unit}\rangle,\ \langle T \mid t[\mathsf{exp} \leftarrow \mathcal{U}[e[V/x]]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[e[V/x]]$ (because $t.\mathsf{exp} = \mathcal{U}[e[V/x]]$) and S is equal to $\bullet$ in fact the last field in Abadi.system is equal to unit.

- if in Abadi.system we have this transition:

$$\frac{}{\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{ref}\ V])\rangle \xmapsto{(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v))} \langle \sigma_A[r \mapsto V], E, (\theta, \mathcal{P}[r])\rangle\ \mathsf{if}\ r \in RefLoc\ -\ dom(\sigma)}\ \textsc{Trans Ref P}$$

in our system we have:

$$\frac{\dfrac{\dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[\mathsf{ref}\ v] \qquad \mathsf{ref}\ v \xrightarrow{\mathsf{ref}(r,v)} r}{T \mid t \xrightarrow{(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v))} T \mid t[exp \leftarrow \mathcal{P}[r]]}\ \mathcal{P}[r] \neq \mathsf{unit}\ \textsc{Step P1} \qquad \dfrac{S = \theta}{S \xrightarrow{(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v))} S}\ \textsc{Step P1} \qquad \dfrac{r \in (RefLoc - \mathsf{dom}(\sigma))}{\sigma \xrightarrow{(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v))} \sigma[r \mapsto v]}\ \textsc{Ref}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v)),(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v)),(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v))\rangle} T \mid t[\mathsf{exp} \leftarrow \mathcal{P}[r]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma[r \mapsto v]}$$

We know that $\{\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{ref}\ v])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{ref}\ v]$.

Then after the transitions we have that $\{\langle \sigma_A[r \mapsto v], E, (\theta, \mathcal{P}[r])\rangle,\ \langle T \mid t[\mathsf{exp} \leftarrow \mathcal{P}[r]] \otimes_{\mathcal{R}}$

$\theta \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $\mathcal{P}[r]$ (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[r]$).

- if in Abadi.system we have this transition:

  Trans Ref P
  $$\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{ref}\ V])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{ref}(r,v))} \langle \sigma_A[r \mapsto V], E, \mathsf{unit}\rangle \qquad \text{if } r \in RefLoc\ -\ dom(\sigma)$$

  in our system we have:

$$\dfrac{\dfrac{\dfrac{}{S \xrightarrow{(\theta,\mathsf{unit},\mathsf{ref}(r,v))} \bullet}\ \text{Step P2} \qquad \dfrac{r \in (RefLoc - \mathsf{dom}(\sigma))}{\sigma \xrightarrow{(\theta,\mathsf{unit},\mathsf{ref}(r,v))} \sigma[r \mapsto v]}\ \text{Ref}}{T \mid t \xrightarrow{(\theta,\mathsf{unit},\mathsf{ref}(r,v))} T}\ \text{Step P2}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{ref}(r,v)),(\theta,\mathsf{unit},\mathsf{ref}(r,v)),(\theta,\mathsf{unit},\mathsf{ref}(r,v))\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]}$$

  We know that $\{\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{ref}\ v])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{ref}\ v]$.

  Then after the transitions we have that $\{\langle \sigma_A[r \mapsto v], E, \mathsf{unit}\rangle, \langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle\} \in Rel$.

  This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ because $\mathcal{P}[r] = \mathsf{unit}$.

- if in Abadi.system we have this transition:

  Trans Ref U
  $$\langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{ref}\ V]).E', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{ref}(r,v)))} \langle \sigma_A[r \mapsto V], E.(\theta, \mathcal{U}[r]).E', \mathsf{unit}\rangle$$
  $$\text{if } r \in RefLoc\ -\ dom(\sigma)$$

  in our system we have:

$$\frac{\textsc{Step U}\quad t.\mathsf{id}=\theta \qquad t.\mathsf{exp}=\mathcal{U}[\mathsf{ref}\ V] \qquad \mathsf{ref}\ V \xhookrightarrow{\mathsf{ref}(r,v)} r}{T \mid t \xrightarrow{(\theta,\mathsf{unp}(\mathsf{ref}(r,v)))} T \mid t[exp \hookleftarrow \mathcal{U}[r]]} \qquad \frac{\textsc{Step U/Close}\quad S=\bullet}{S \xrightarrow{(\theta,\mathsf{unp}(\mathsf{ref}(r,v)))} S} \qquad \frac{\textsc{URef}\quad r \in (RefLoc - \mathsf{dom}(\sigma))}{\sigma \xrightarrow{(\theta,\mathsf{unp}(\mathsf{ref}(r,v)))} \sigma[r \mapsto v]}$$

$$T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{ref}(r,v))),(\theta,\mathsf{unp}(\mathsf{ref}(r,v))),(\theta,\mathsf{unp}(\mathsf{ref}(r,v)))\rangle} T \mid t[exp \hookleftarrow \mathcal{U}[r]] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]$$

We know that $\{\langle \sigma_A, E.(\theta,\mathcal{U}[\mathsf{ref}\ v]).E', \mathsf{unit}\rangle, \langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id}=\theta$ and $t.\mathsf{exp}=\mathcal{U}[\mathsf{ref}\ v]$.

Then after the transitions we have that $\{\langle \sigma_A[r \mapsto v], E.(\theta,\mathcal{U}[r]).E', \mathsf{unit}\rangle,\ \langle T \mid t[exp \hookleftarrow \mathcal{U}[r]] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[r]$ (because $t.\mathsf{exp}=\mathcal{U}[r]$) and S is equal to $\bullet$ in fact the last field in Abadi.system is equal to unit.

- if in Abadi.system we have this transition:

$$\frac{\textsc{Trans Deref P}}{\langle \sigma_A, E, (\theta, \mathcal{P}[!r])\rangle \xmapsto{(\theta,\text{no unit},\mathsf{rd}(r,v))} \langle \sigma_A, E, (\theta, \mathcal{P}[V])\rangle \qquad \text{if } \sigma(r)=V}$$

in our system we have:

$$\frac{\textsc{Step P1}\quad t.\mathsf{id}=\theta \qquad t.\mathsf{exp}=\mathcal{P}[!r] \qquad !r \xhookrightarrow{\mathsf{rd}(r,v)} v}{T \mid t \xrightarrow{(\theta,\text{no unit},\mathsf{rd}(r,v))} T \mid t[exp \hookleftarrow \mathcal{P}[v]]}\ \mathcal{P}[v] \neq \mathsf{unit} \qquad \frac{\textsc{Step P1}\quad S=\theta}{S \xrightarrow{(\theta,\text{no unit},\mathsf{rd}(r,v))} S} \qquad \frac{\textsc{DeRef (Read)}\quad \sigma(r)=v}{\sigma \xrightarrow{(\theta,\mathsf{rd}(r,v))} \sigma}$$

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\text{no unit},\mathsf{rd}(r,v)),(\theta,\text{no unit},\mathsf{rd}(r,v)),(\theta,\text{no unit},\mathsf{rd}(r,v))\rangle} T \mid t[\mathsf{exp} \hookleftarrow \mathcal{P}[v]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma$$

We know that $\{\langle \sigma_A, E, (\theta, \mathcal{P}[!r])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id}=\theta$, $t.\mathsf{exp}=\mathcal{P}[!r]$.

Then after the transitions we have that $\{\langle \sigma_A, E, (\theta, \mathcal{P}[V])\rangle,\ \langle T \mid t[\mathsf{exp} \hookleftarrow \mathcal{P}[V]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two

systems changed either E or T) and the thread t correspond to $\mathcal{P}[v]$ (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[v]$).

- if in Abadi.system we have this transition:

$$\textsc{Trans Deref P}\quad \langle \sigma_A, E, (\theta, \mathcal{P}[!r])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{rd}(r,v))} \langle \sigma_A, E, \mathsf{unit}\rangle \qquad \text{if } \sigma(r) = V$$

in our system we have:

$$\dfrac{\dfrac{\textsc{Step P2}\quad t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[!r] \qquad !r \xhookrightarrow{\mathsf{rd}(r,v)} v}{T \mid t \xrightarrow{(\theta,\mathsf{unit},\mathsf{rd}(r,v))} T}\ \mathcal{P}[v] = \mathsf{unit} \qquad \dfrac{\dfrac{\textsc{Step P2}\quad S = \theta}{S \xrightarrow{(\theta,\mathsf{unit},\mathsf{rd}(r,v))} \bullet} \qquad \dfrac{\textsc{DeRef (Read)}\quad \sigma(r) = v}{\sigma \xrightarrow{(\theta,\mathsf{rd}(r,v))} \sigma}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{rd}(r,v)),(\theta,\mathsf{unit},\mathsf{rd}(r,v)),(\theta,\mathsf{unit},\mathsf{rd}(r,v))\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle \sigma_A, E, (\theta, \mathcal{P}[!r])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in \mathit{Rel}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[!r]$.

Then after the transitions we have that $\{\langle \sigma_A, E, \mathsf{unit}\rangle, \langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in \mathit{Rel}$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ because $\mathcal{P}[v] = \mathsf{unit}$.

- if in Abadi.system we have this transition:

$$\textsc{Trans Deref U}\quad \langle \sigma_A, E.(\theta,\mathcal{U}[!r]).E', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{rd}(r,v)))} \langle \sigma_A, E.(\theta,\mathcal{U}[V]).E', \mathsf{unit}\rangle \qquad \text{if } \sigma(r) = V$$

in our system we have:

$$\dfrac{\dfrac{\textsc{Step U}\quad t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{U}[!r] \qquad !r \xhookrightarrow{\mathsf{rd}(r,v)} v}{T \mid t \xrightarrow{(\theta,\mathsf{unp}(\mathsf{rd}(r,v)))} T \mid t[exp \leftarrow \mathcal{U}[v]]} \qquad \dfrac{\textsc{Step U/Close}\quad S = \bullet}{S \xrightarrow{(\theta,\mathsf{unp}(\mathsf{rd}(r,v)))} S} \qquad \dfrac{\textsc{UDeRef (Read)}\quad \sigma(r) = v}{\sigma \xrightarrow{(\theta,\mathsf{unp}(\mathsf{rd}(r,v)))} \sigma}}{T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{rd}(r,v))),(\theta,\mathsf{unp}(\mathsf{rd}(r,v))),(\theta,\mathsf{unp}(\mathsf{rd}(r,v)))\rangle} T \mid t[\mathsf{exp} \leftarrow \mathcal{U}[v]] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle \sigma_A, E.(\theta,\mathcal{U}[!r]).E', \mathsf{unit}\rangle, \langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in \mathit{Rel}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[!r]$.

Then after the transitions we have that $\{\langle \sigma_A, E.(\theta,\mathcal{U}[V]).E', \mathsf{unit}\rangle, \langle T \mid t[\mathsf{exp} \leftarrow \mathcal{U}[v]] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in \mathit{Rel}$.

This couple belongs to the relation *Rel* because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[v]$ (because $t.\mathsf{exp} = \mathcal{U}[v]$) and S is equal to $\bullet$ in fact the last field in Abadi.system is equal to unit.

- if in Abadi.system we have this transition:

$$\textsc{Trans Set P}\quad \langle \sigma_A, E, (\theta, \mathcal{P}[r := V])\rangle \xmapsto{(\theta,\text{no unit},\mathsf{wr}(r,v))} \langle \sigma_A[r \mapsto V], E, (\theta, \mathcal{P}[\mathsf{unit}])\rangle$$

in our system we have:

$$\dfrac{\dfrac{\dfrac{}{S = \theta}}{S \xrightarrow{(\theta,\text{no unit},\mathsf{wr}(r,v))} S}\textsc{Step P1} \qquad \dfrac{\dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[r := v] \qquad r := v \xrightarrow{\mathsf{wr}(r,v)} \mathsf{unit}}{T \mid t \xrightarrow{(\theta,\text{no unit},\mathsf{wr}(r,v))} T \mid t[exp \leftarrow\!\shortmid \mathcal{P}[\mathsf{unit}]]}\textsc{Step P1}}{\mathcal{P}[\mathsf{unit}] \neq \mathsf{unit}} \qquad \dfrac{}{\sigma \xrightarrow{(\theta,\text{no unit},\mathsf{wr}(r,v))} \sigma[r \mapsto v]}\textsc{Set (Write)}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\text{no unit},\mathsf{wr}(r,v)),(\theta,\text{no unit},\mathsf{wr}(r,v)),(\theta,\text{no unit},\mathsf{wr}(r,v))\rangle} T \mid t[\mathsf{exp} \leftarrow\!\shortmid \mathcal{P}[\mathsf{unit}]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma[r \mapsto v]}$$

We know that $\{\langle \sigma_A, E, (\theta, \mathcal{P}[r := V])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[r := v]$.

Then after the transitions we have that $\{\langle \sigma_A[r \mapsto V], E, (\theta, \mathcal{P}[\mathsf{unit}])\rangle,\ \langle T \mid t[\mathsf{exp} \leftarrow\!\shortmid \mathcal{P}[\mathsf{unit}]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle\} \in Rel$.

This couple belongs to the relation *Rel* because we have that $\sigma = \sigma_A$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $\mathcal{P}[\mathsf{unit}]$ (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$).

- if in Abadi.system we have this transition:

$$\textsc{Trans Set P}\quad \langle \sigma_A, E, (\theta, \mathcal{P}[r := V])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{wr}(r,v))} \langle \sigma_A[r \mapsto V], E, \mathsf{unit}\rangle$$

in our system we have:

$$\dfrac{\dfrac{\dfrac{}{S \xrightarrow{(\theta,\mathsf{unit},\mathsf{wr}(r,v))} \bullet}\ \text{Step P2}\quad S=\theta \qquad \dfrac{}{\sigma \xrightarrow{(\theta,\mathsf{unit},\mathsf{wr}(r,v))} \sigma[r \mapsto v]}\ \text{Set (Write)}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{wr}(r,v)),(\theta,\mathsf{unit},\mathsf{wr}(r,v)),(\theta,\mathsf{unit},\mathsf{wr}(r,v))\rangle} T \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]}\ \text{Step P2}$$

$$\dfrac{t.\mathsf{id}=\theta \qquad t.\mathsf{exp}=\mathcal{P}[r:=v] \qquad r:=v \xrightarrow{\mathsf{wr}(r,v)} \mathsf{unit}}{T \mid t \xrightarrow{(\theta,\mathsf{unit},\mathsf{wr}(r,v))} T}\ \mathcal{P}[\mathsf{unit}]=\mathsf{unit}$$

We know that $\{\langle \sigma_A, E, (\theta, \mathcal{P}[r := V])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[r := v]$.

Then after the transitions we have that $\{\langle \sigma_A[r \mapsto V], E, \mathsf{unit}\rangle, \langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ because $\mathcal{P}[\mathsf{unit}] = \mathsf{unit}$.

- if in Abadi.system we have this transition:

$$\text{Trans Set U}\quad \langle \sigma_A, E.(\theta, \mathcal{U}[r := V]).E', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{wr}(r,v)))} \langle \sigma_A[r \mapsto V], E.(\theta, \mathcal{U}[\mathsf{unit}]).E', \mathsf{unit}\rangle$$

in our system we have:

$$\dfrac{\dfrac{\dfrac{}{S \xrightarrow{(\theta,\mathsf{unp}(\mathsf{wr}(r,v)))} S}\ \text{Step U/Close}\quad S=\bullet \qquad \dfrac{}{\sigma \xrightarrow{(\theta,\mathsf{unp}(\mathsf{wr}(r,v)))} \sigma[r \mapsto v]}\ \text{Set (Write)}}{T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{wr}(r,v))),(\theta,\mathsf{unp}(\mathsf{wr}(r,v))),(\theta,\mathsf{unp}(\mathsf{wr}(r,v)))\rangle} T \mid t[\mathsf{exp} \leftarrow \mathcal{U}[\mathsf{unit}]] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]}\ \text{Step U}$$

$$\dfrac{t.\mathsf{id}=\theta \qquad t.\mathsf{exp}=\mathcal{U}[r:=v] \qquad r:=v \xrightarrow{\mathsf{wr}(r,v)} \mathsf{unit}}{T \mid t \xrightarrow{(\theta,\mathsf{unp}(\mathsf{wr}(r,v)))} T \mid t[exp \leftarrow \mathcal{U}[\mathsf{unit}]]}$$

We know that $\{\langle \sigma_A, E.(\theta, \mathcal{U}[r := V]).E', \mathsf{unit}\rangle, \langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[r := v]$.

Then after the transitions we have that $\{\langle \sigma_A[r \mapsto V], E.(\theta, \mathcal{U}[\mathsf{unit}]).E', \mathsf{unit}\rangle, \langle T \mid t[\mathsf{exp} \leftarrow \mathcal{U}[\mathsf{unit}]] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[\mathsf{unit}]$ (because $t.\mathsf{exp} = \mathcal{U}[\mathsf{unit}]$) and S is equal to $\bullet$ in fact the last field in Abadi.system is equal to unit.

- if in Abadi.system we have this transition:

$$\textsc{Trans Async P}\quad \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{async}\ e'])\rangle \xmapsto{(\theta,\mathsf{no\ unit},\mathsf{async}(e'))} \langle \sigma_A, E.(\theta', e'), (\theta, \mathcal{P}[\mathsf{unit}])\rangle$$

in our system we have:

$$\dfrac{\dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[\mathsf{async}\ e'] \qquad t', \theta'\ fresh}{T \mid t \xrightarrow{(\theta,\mathsf{no\ unit},\mathsf{async}(e'))} T \mid t[exp \leftarrow\!\!\rightarrow \mathcal{P}[\mathsf{unit}]] \mid t'[id \leftarrow\!\!\rightarrow \theta', exp \leftarrow\!\!\rightarrow e']}\ \textsc{Spawn P1}\ \mathcal{P}[\mathsf{unit}] \neq \mathsf{unit} \qquad \dfrac{S = \theta}{S \xrightarrow{(\theta,\mathsf{no\ unit},\mathsf{async}(e'))} S}\ \textsc{Step P1}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{no\ unit},\mathsf{async}(e')),\phi,(\theta,\mathsf{no\ unit},\mathsf{async}(e'))\rangle} T \mid t[exp \leftarrow\!\!\rightarrow \mathcal{P}[\mathsf{unit}]] \mid t'[id \leftarrow\!\!\rightarrow \theta', exp \leftarrow\!\!\rightarrow e'] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{async}\ e'])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{async}\ e']$.

Then after the transitions we have that $\{\langle \sigma_A, E.(\theta', e'), (\theta, \mathcal{P}[\mathsf{unit}])\rangle, \langle T \mid t[\mathsf{exp} \leftarrow\!\!\rightarrow \mathcal{P}[\mathsf{unit}]] \mid t'[id \leftarrow\!\!\rightarrow \theta', exp \leftarrow\!\!\rightarrow e'] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T), e correspond to t' (because $t'.\mathsf{exp} = e$ and both have been added with the last transition) and the thread t correspond to $\mathcal{P}[\mathsf{unit}]$ (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$).

- if in Abadi.system we have this transition:

$$\textsc{Trans Async P}\quad \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{async}\ e'])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{async}(e'))} \langle \sigma_A, E.(\theta', e'), \mathsf{unit}\rangle$$

in our system we have:

$$\dfrac{\dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[\mathsf{async}\ e'] \qquad t', \theta'\ fresh}{T \mid t \xrightarrow{(\theta,\mathsf{unit},\mathsf{async}(e'))} T \mid t'[id \leftarrow\!\!\rightarrow \theta', exp \leftarrow\!\!\rightarrow e']}\ \textsc{Spawn P2}\ \mathcal{P}[\mathsf{unit}] = \mathsf{unit} \qquad \dfrac{S = \theta}{S \xrightarrow{(\theta,\mathsf{unit},\mathsf{async}(e'))} \bullet}\ \textsc{Step P2}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{async}(e')),\phi,(\theta,\mathsf{unit},\mathsf{async}(e'))\rangle} T \mid t'[id \leftarrow\!\!\rightarrow \theta', exp \leftarrow\!\!\rightarrow e'] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{async}\ e'])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} =$

$\mathcal{P}[\mathsf{async}\ e']$.

Then after the transitions we have that $\{\langle\sigma_A, E.(\theta', e'), \mathsf{unit}\rangle, \langle T \mid t'[id \hookleftarrow \theta', exp \hookleftarrow e'] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T), e correspond to t' (because $t'.\mathsf{exp} = e$ and both have been added with the last transition) and $S = \bullet$ because $\mathcal{P}[\mathsf{unit}] = \mathsf{unit}$.

- if in Abadi.system we have this transition:

  Trans Async U
  $$\langle\sigma_A, E.(\theta, \mathcal{U}[\mathsf{async}\ e']).E', \mathsf{unit}\rangle \xmapsto{(\theta, \mathsf{unp}(\mathsf{async}(e')))} \langle\sigma_A, E.(\theta', e').(\theta, \mathcal{U}[\mathsf{unit}]).E', \mathsf{unit}\rangle$$

  in our system we have:

$$\frac{\text{Spawn U}\quad \dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{U}[\mathsf{async}\ e'] \qquad t', \theta'\ fresh}{T \mid t \xrightarrow{(\theta, \mathsf{unp}(\mathsf{async}(e')))} T \mid t[exp \hookleftarrow \mathcal{U}[\mathsf{unit}]] \mid t'[id \hookleftarrow \theta', exp \hookleftarrow e']} \qquad \text{Step U/Close}\quad \dfrac{S = \bullet}{S \xrightarrow{(\theta, \mathsf{unp}(\mathsf{async}(e')))} S}}{T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta, \mathsf{unp}(\mathsf{async}(e'))), \phi, (\theta, \mathsf{unp}(\mathsf{async}(e')))\rangle} T \mid t[exp \hookleftarrow \mathcal{U}[\mathsf{unit}]] \mid t'[id \hookleftarrow \theta', exp \hookleftarrow e'] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma}$$

  We know that $\{\langle\sigma_A, E.(\theta, \mathcal{U}[\mathsf{async}\ e']).E', \mathsf{unit}\rangle, \langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[\mathsf{async}\ e']$.

  Then after the transitions we have that $\{\langle\sigma_A, E.(\theta', e').(\theta, \mathcal{U}[\mathsf{unit}]).E', \mathsf{unit}\rangle, \langle T \mid t[exp \hookleftarrow \mathcal{U}[\mathsf{unit}]] \mid t'[id \hookleftarrow \theta', exp \hookleftarrow e'] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

  This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory.

  We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[\mathsf{unit}]$ (because $t.\mathsf{exp} = \mathcal{U}[\mathsf{unit}]$), e' correspond to t' (because $t'.\mathsf{exp} = e'$ and both have been added with the last transition) and S is equal to $\bullet$ in fact the last field in Abadi.system is equal to unit.

- if in Abadi.system we have this transition:

  Trans Block P
  $$\langle\sigma_A, E, (\theta, \mathcal{P}[\mathsf{blockUntil\ true}])\rangle \xmapsto{(\theta, \mathsf{no\ unit}, \mathsf{blockUntil(true)})} \langle\sigma_A, E, (\theta, \mathcal{P}[\mathsf{unit}])\rangle$$

  in our system we have:

$$\dfrac{\dfrac{\textsc{Step P1}\quad t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[\mathsf{blockUntil\ true}] \qquad \mathsf{blockUntil\ true} \xhookrightarrow{\mathsf{blockUntil(true)}} \mathsf{unit}}{T \mid t \xrightarrow{(\theta,\mathsf{no\ unit},\mathsf{blockUntil(true)})} T \mid t[exp \leftarrow\!\shortmid \mathcal{P}[\mathsf{unit}]]}\ \mathcal{P}[\mathsf{unit}] \neq \mathsf{unit} \qquad \dfrac{\textsc{Step P1}\quad S = \theta}{S \xrightarrow{(\theta,\mathsf{no\ unit},\mathsf{blockUntil(true)})} S}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{no\ unit},\mathsf{blockUntil(true)}),\phi,(\theta,\mathsf{no\ unit},\mathsf{blockUntil(true)})\rangle} T \mid t[\mathsf{exp} \leftarrow\!\shortmid \mathcal{P}[\mathsf{unit}]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle\sigma_A, E, (\theta, \mathcal{P}[\mathsf{blockUntil\ true}])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{blockUntil\ true}]$.

Then after the transitions we have that $\{\langle\sigma_A, E, (\theta, \mathcal{P}[\mathsf{unit}])\rangle,\ \langle T \mid t[\mathsf{exp} \leftarrow\!\shortmid \mathcal{P}[\mathsf{unit}]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $\mathcal{P}[\mathsf{unit}]$ (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$).

- if in Abadi.system we have this transition:

$$\dfrac{\textsc{Trans Block P}}{\langle\sigma_A, E, (\theta, \mathcal{P}[\mathsf{blockUntil\ true}])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{blockUntil(true)})} \langle\sigma_A, E, \mathsf{unit}\rangle}$$

in our system we have:

$$\dfrac{\dfrac{\textsc{Step P2}\quad t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[\mathsf{blockUntil\ true}] \qquad \mathsf{blockUntil\ true} \xhookrightarrow{\mathsf{blockUntil(true)}} \mathsf{unit}}{T \mid t \xrightarrow{(\theta,\mathsf{unit},\mathsf{blockUntil(true)})} T}\ \mathcal{P}[\mathsf{unit}] = \mathsf{unit} \qquad \dfrac{\textsc{Step P2}\quad S = \theta}{S \xrightarrow{(\theta,\mathsf{unit},\mathsf{blockUntil(true)})} \bullet}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{blockUntil(true)}),\phi,(\theta,\mathsf{unit},\mathsf{blockUntil(true)})\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle\sigma_A, E, (\theta, \mathcal{P}[\mathsf{blockUntil\ true}])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{blockUntil\ true}]$.

Then after the transitions we have that $\{\langle\sigma_A, E, \mathsf{unit}\rangle,\ \langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ because $\mathcal{P}[\mathsf{unit}] = \mathsf{unit}$.

- if in Abadi.system we have this transition:

  TRANS BLOCK U
  $$\langle \sigma_A, E.\mathcal{U}[\mathsf{blockUntil\ true}].E', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{unp(blockUntil(true))})} \langle \sigma_A, E.\mathcal{U}[\mathsf{unit}].E', \mathsf{unit}\rangle$$

  in our system we have:

  $$\dfrac{\dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{U}[\mathsf{blockUntil\ true}] \qquad \mathsf{blockUntil\ true} \xhookrightarrow{\mathsf{blockUntil\ true}} \mathsf{unit}}{T \mid t \xrightarrow{(\theta,\mathsf{unp(blockUntil\ true)})} T \mid t[exp \leftarrow\!\shortmid \mathcal{U}[\mathsf{unit}]]}\ \textsc{Step U} \qquad \dfrac{S = \bullet}{S \xrightarrow{(\theta,\mathsf{unp(blockUntil\ true)})} S}\ \textsc{Step U/Close}}{T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp(blockUntil\ true)}),\phi,(\theta,\mathsf{unp(blockUntil\ true)})\rangle} T \mid t[exp \leftarrow\!\shortmid \mathcal{U}[\mathsf{unit}] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma}$$

  We know that $\{\langle \sigma_A, E.\mathcal{U}[\mathsf{blockUntil\ true}].E', \mathsf{unit}\rangle, \langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[\mathsf{blockUntil\ true}]$.

  Then after the transitions we have that $\{\langle \sigma_A, E.\mathcal{U}[\mathsf{unit}].E', \mathsf{unit}\rangle, \langle T \mid t[exp \leftarrow\!\shortmid \mathcal{U}[\mathsf{unit}] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

  This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory.

  We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[\mathsf{unit}]$ (because $t.\mathsf{exp} = \mathcal{U}[\mathsf{unit}]$) and S is equal to $\bullet$ in fact the last field in Abadi.system is equal to unit.

- if in Abadi.system we have this transition:

  TRANS UNPROTECTED
  $$\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{unprotected}\ e])\rangle \xmapsto{(\theta,\mathsf{unprotect})} \langle \sigma_A, E.(\theta, \mathcal{P}[\mathsf{unprotected}\ e]), \mathsf{unit}\rangle$$

  in our system we have:

  $$\dfrac{\dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[\mathsf{unprotected}\ e]}{T \mid t \xrightarrow{(\theta,\mathsf{unprtect})} T \mid t}\ \textsc{Unprotected} \qquad \dfrac{S = \theta}{S \xrightarrow{(\theta,\mathsf{unprotect})} \bullet}\ \textsc{Unprotected}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unprtect}),\phi,(\theta,\mathsf{unprtect})\rangle} T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma}$$

  We know that $\{\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{unprotected}\ e])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{unprotected}\ e]$.

Then after the transitions we have that $\{\langle \sigma_A, E.(\theta, \mathcal{P}[\mathsf{unprotected}\ e]), \mathsf{unit}\rangle, \langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T), the thread t correspond to $\mathcal{P}[\mathsf{unprotected}\ e]$ (because $t.\mathsf{exp} = \mathcal{P}[\mathsf{unprotected}\ e]$) and S is equal to $\bullet$ in fact the last field in Abadi.system is equal to unit.

- if in Abadi.system we have this transition:

  TRANS CLOSE
  $$\langle \sigma_A, E.(\theta, \mathcal{E}[\mathsf{unprotected}\ V]).E', \mathsf{unit}\rangle \xmapsto{(\theta, \mathsf{unp}(V))} \langle \sigma_A, E.(\theta, \mathcal{E}[V]).E', \mathsf{unit}\rangle$$

  in our system we have:

  $$\frac{\dfrac{\text{CLOSE}\quad t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{E}[\mathsf{unprotected}\ V]}{T \mid t \xrightarrow{(\theta, \mathsf{unp}(V))} T \mid t[exp \leftarrowtail \mathcal{E}[V]]} \qquad \dfrac{\text{STEP U/CLOSE}\quad S = \bullet}{S \xrightarrow{(\theta, \mathsf{unp}(V))} S}}{T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle (\theta, \mathsf{unp}(V)), \phi, (\theta, \mathsf{unp}(V))\rangle} T \mid t[exp \leftarrowtail \mathcal{E}[V]] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma}$$

  We know that $\{\langle \sigma_A, E.(\theta, \mathcal{E}[\mathsf{unprotected}\ V]).E', \mathsf{unit}\rangle, \langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{E}[\mathsf{unprotected}\ V]$.

  Then after the transitions we have that $\{\langle \sigma_A, E.(\theta, \mathcal{E}[V]).E', \mathsf{unit}\rangle, \langle T \mid t[exp \leftarrowtail \mathcal{E}[V]] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

  This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{E}[V]$ (because $t.\mathsf{exp} = \mathcal{E}[V]$) and S is equal to $\bullet$ in fact the last field in Abadi.system is equal to unit.

To proof the bi-simulation we can prove that given Rel as a strong simulation relation then if also Rel^{-1} is a strong simulation relation then Rel is a bi-simulation relation. Now we show the relationship Rel^{-1} derived directly from Rel.

Given $\{\langle T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, e\rangle\}$ then this tuple is in Rel^{-1} if:

- $\sigma_A = \sigma$
- $\forall t_i \in T$ such that $t_i.\mathsf{id} = \theta_i$ and $\theta_i \neq S$ then exist $e_i = t_i.\mathsf{exp}$ and $e_i \in E$
- if $S = \bullet$ then $e = \mathsf{unit}$ otherwise (if $S \neq \bullet$) then exist $t \in T$ such that $t.\mathsf{id} = S$ and we have that $e = t.\mathsf{exp}$

Now we start to analyses all the possible case:

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\text{no unit,act}),\phi,(\theta,\text{no unit,act})\rangle} T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma$$

  in Abadi.system we have:

$$\textsc{Trans Activate}$$
$$\langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle \xmapsto{(\theta,\text{no unit,act})} \langle \sigma_A, E.E', (\theta, e)\rangle$$

  We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$.

  Then after the transitions we have that $\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \{\langle \sigma_A, E.E', (\theta, e)\rangle\} \in Rel^{-1}$.

  This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T) and e correspond to the thread t (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$).

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{act}),\phi,(\theta,\mathsf{unit},\mathsf{act})\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \qquad \text{if } e = \mathsf{unit}$$

  in Abadi.system we have:

$$\textsc{Trans Activate}$$
$$\langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{act})} \langle \sigma_A, E.E', \mathsf{unit}\rangle$$

  We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$.

  Then after the transitions we have that $\langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \{\langle \sigma_A, E.E', \mathsf{unit}\rangle\} \in Rel^{-1}$.

  This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\text{no unit,pure}),\phi,(\theta,\text{no unit,pure})\rangle} T \mid t[\mathsf{exp} \leftarrow\!\shortmid \mathcal{P}[e[V/x]]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma$$

  in Abadi.system we have:

$$\textsc{Trans Appl P}$$
$$\langle \sigma_A, E, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle \xmapsto{(\theta,\text{no unit,pure})} \langle \sigma_A, E, (\theta, \mathcal{P}[e[V/x]])\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[(\lambda\ x.e)\ V]$.

Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \hookleftarrow \mathcal{P}[e[V/x]]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[e[V/x]])\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $\mathcal{P}[e[V/x]]$ correspond to the thread t (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[e[V/x]]$).

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{pure}),\phi,(\theta,\mathsf{unit},\mathsf{pure})\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

$$\textsc{Trans Appl P}$$
$$\langle \sigma_A, E, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{pure})} \langle \sigma_A, E, \mathsf{unit}\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[(\lambda\ x.e)\ V]$.

Then after the transitions we have that $\{\langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, \mathsf{unit}\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{pure})),\phi,(\theta,\mathsf{unp}(\mathsf{pure}))\rangle} T \mid t[exp \hookleftarrow \mathcal{U}[e[V/x]]] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

$$\textsc{Trans Appl U}$$
$$\langle \sigma_A, E.(\theta,\mathcal{U}[(\lambda\ x.e)\ V]).E', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{pure}))} \langle \sigma_A, E.(\theta,\mathcal{U}[e[V/x]]).E', \mathsf{unit}\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta,\mathcal{U}[(\lambda\ x.e)\ V]).E', \mathsf{unit}\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[(\lambda\ x.e)\ V]$.

Then after the transitions we have that $\{\langle T \mid t[exp \hookleftarrow \mathcal{U}[e[V/x]]] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta,\mathcal{U}[e[V/x]]).E', \mathsf{unit}\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the

expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[e[V/x]]$ (because $t.\mathsf{exp} = \mathcal{U}[e[V/x]]$) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v)),(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v)),(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v))\rangle} T \mid t[\mathsf{exp} \leftarrowtail \mathcal{P}[r]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma[r \mapsto v]$$

in Abadi.system we have:

$$\textsc{Trans Ref P}$$
$$\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{ref}\ V])\rangle \xmapsto{(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v))} \langle \sigma_A[r \mapsto V], E, (\theta, \mathcal{P}[r])\rangle \quad \mathsf{if}\ r \in RefLoc\ -\ dom(\sigma)$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{ref}\ V])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{ref}\ V]$.

Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \leftarrowtail \mathcal{P}[r]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle, \langle \sigma_A[r \mapsto V], E, (\theta, \mathcal{P}[r])\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $\mathcal{P}[r]$ correspond to the thread t (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[r]$).

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{ref}(r,v)),(\theta,\mathsf{unit},\mathsf{ref}(r,v)),(\theta,\mathsf{unit},\mathsf{ref}(r,v))\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]$$

in Abadi.system we have:

$$\textsc{Trans Ref P}$$
$$\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{ref}\ V])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{ref}(r,v))} \langle \sigma_A[r \mapsto V], E, \mathsf{unit}\rangle \qquad \mathsf{if}\ r \in RefLoc\ -\ dom(\sigma)$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{ref}\ V])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{ref}\ V]$.

Then after the transitions we have that $\{\langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle, \langle \sigma_A[r \mapsto V], E, \mathsf{unit}\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{ref}(r,v))),(\theta,\mathsf{unp}(\mathsf{ref}(r,v))),(\theta,\mathsf{unp}(\mathsf{ref}(r,v)))\rangle} T \mid t[exp \leftarrow \mathcal{U}[r]] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]$$

in Abadi.system we have:

$$\textsc{Trans Ref U}$$
$$\langle \sigma_A, E.(\theta,\mathcal{U}[\mathsf{ref}\ V]).E', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{ref}(r,v)))} \langle \sigma_A[r \mapsto V], E.(\theta,\mathcal{U}[r]).E', \mathsf{unit}\rangle \quad \mathsf{if}\ r \in RefLoc\ -\ dom(\sigma)$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta,\mathcal{U}[\mathsf{ref}\ V]).E', \mathsf{unit}\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[\mathsf{ref}\ V]$.

Then after the transitions we have that $\{\langle T \mid t[exp \leftarrow \mathcal{U}[r]] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle, \langle \sigma_A[r \mapsto V], E.(\theta,\mathcal{U}[r]).E', \mathsf{unit}\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), $\mathcal{U}[r]$ correspond to the thread t (because $t.\mathsf{exp} = \mathcal{U}[r]$) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{no\ unit},\mathsf{rd}(r,v)),(\theta,\mathsf{no\ unit},\mathsf{rd}(r,v)),(\theta,\mathsf{no\ unit},\mathsf{rd}(r,v))\rangle} T \mid t[\mathsf{exp} \leftarrow \mathcal{P}[v]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

$$\textsc{Trans Deref P}$$
$$\langle \sigma_A, E, (\theta, \mathcal{P}[!r])\rangle \xmapsto{(\theta,\mathsf{no\ unit},\mathsf{rd}(r,v))} \langle \sigma_A, E, (\theta, \mathcal{P}[V])\rangle \qquad \mathsf{if}\ \sigma(r) = V$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[!r])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[!r]$.

Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \leftarrow \mathcal{P}[v]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[V])\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $\mathcal{P}[v]$ (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[v]$).

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{rd}(r,v)),(\theta,\mathsf{unit},\mathsf{rd}(r,v)),(\theta,\mathsf{unit},\mathsf{rd}(r,v))\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

$$\textsc{Trans Deref P}$$
$$\langle \sigma_A, E, (\theta, \mathcal{P}[!r])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{rd}(r,v))} \langle \sigma_A, E, \mathsf{unit}\rangle \qquad \text{if } \sigma(r) = V$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[!r])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[!r]$.

Then after the transitions we have that $\{\langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, \mathsf{unit}\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{rd}(r,v))),(\theta,\mathsf{unp}(\mathsf{rd}(r,v))),(\theta,\mathsf{unp}(\mathsf{rd}(r,v)))\rangle} T \mid t[\mathsf{exp} \leftarrowtail \mathcal{U}[v]] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

$$\textsc{Trans Deref U}$$
$$\langle \sigma_A, E.(\theta,\mathcal{U}[!r]).E', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{rd}(r,v)))} \langle \sigma_A, E.(\theta,\mathcal{U}[V]).E', \mathsf{unit}\rangle \qquad \text{if } \sigma(r) = V$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta,\mathcal{U}[!r]).E', \mathsf{unit}\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[!r]$.

Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \leftarrowtail \mathcal{U}[v]] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta,\mathcal{U}[V]).E', \mathsf{unit}\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), $\mathcal{U}[v]$ correspond to the thread t (because $t.\mathsf{exp} = \mathcal{U}[v]$) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{no\ unit},\mathsf{wr}(r,v)),(\theta,\mathsf{no\ unit},\mathsf{wr}(r,v)),(\theta,\mathsf{no\ unit},\mathsf{wr}(r,v))\rangle} T \mid t[\mathsf{exp} \leftarrowtail \mathcal{P}[\mathsf{unit}]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma[r \mapsto v]$$

in Abadi.system we have:

$$\textsc{Trans Set P}$$
$$\langle \sigma_A, E, (\theta, \mathcal{P}[r := V])\rangle \xmapsto{(\theta,\mathsf{no\ unit},\mathsf{wr}(r,v))} \langle \sigma_A[r \mapsto V], E, (\theta, \mathcal{P}[\mathsf{unit}])\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[r := V])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[r := v]$.

Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \leftarrow\!\!\shortmid \mathcal{P}[\mathsf{unit}]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle, \langle \sigma_A[r \mapsto V], E, (\theta, \mathcal{P}[\mathsf{unit}])\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $\mathcal{P}[\mathsf{unit}]$ (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$).

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{wr}(r,v)),(\theta,\mathsf{unit},\mathsf{wr}(r,v)),(\theta,\mathsf{unit},\mathsf{wr}(r,v))\rangle} T \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]$$

in Abadi.system we have:

$$\textsc{Trans Set P}$$
$$\langle \sigma_A, E, (\theta, \mathcal{P}[r := V])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{wr}(r,v))} \langle \sigma_A[r \mapsto V], E, \mathsf{unit}\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[r := V])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[r := v]$.

Then after the transitions we have that $\{\langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle, \langle \sigma_A[r \mapsto V], E, \mathsf{unit}\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{wr}(r,v))),(\theta,\mathsf{unp}(\mathsf{wr}(r,v))),(\theta,\mathsf{unp}(\mathsf{wr}(r,v)))\rangle} T \mid t[\mathsf{exp} \leftarrow\!\!\shortmid \mathcal{U}[\mathsf{unit}]] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]$$

in Abadi.system we have:

$$\textsc{Trans Set U}$$
$$\langle \sigma_A, E.(\theta,\mathcal{U}[r := V]).E', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{wr}(r,v)))} \langle \sigma_A[r \mapsto V], E.(\theta,\mathcal{U}[\mathsf{unit}]).E', \mathsf{unit}\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta,\mathcal{U}[r := V]).E', \mathsf{unit}\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[r := v]$.

Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \leftarrow\!\!\shortmid \mathcal{U}[\mathsf{unit}]] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle, \langle \sigma_A[r \mapsto V], E.(\theta,\mathcal{U}[\mathsf{unit}]).E', \mathsf{unit}\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the

two systems changed either E and E' or T), $\mathcal{U}[\mathsf{unit}]$ correspond to the thread t (because $t.\mathsf{exp} = \mathcal{U}[\mathsf{unit}]$) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{no\ unit},\mathsf{async}(e')),\phi,(\theta,\mathsf{no\ unit},\mathsf{async}(e'))\rangle} T \mid t[exp \leftarrow \mathcal{P}[\mathsf{unit}]] \mid t'[id \leftarrow \theta', exp \leftarrow e'] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

$$\begin{array}{l}\textsc{Trans Async P}\\ \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{async}\ e'])\rangle \xmapsto{(\theta,\mathsf{no\ unit},\mathsf{async}(e'))} \langle \sigma_A, E.(\theta', e'), (\theta, \mathcal{P}[\mathsf{unit}])\rangle\end{array}$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{async}\ e'])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{async}\ e']$.

Then after the transitions we have that $\{\langle T \mid t[exp \leftarrow \mathcal{P}[\mathsf{unit}]] \mid t'[id \leftarrow \theta', exp \leftarrow e'] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta', e'), (\theta, \mathcal{P}[\mathsf{unit}])\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T), e' correspond to t' (because $t'.\mathsf{exp} = e'$ and both have been added with the last transition) and $\mathcal{P}[\mathsf{unit}]$ correspond to the thread t (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$).

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{async}(e')),\phi,(\theta,\mathsf{unit},\mathsf{async}(e'))\rangle} T \mid t'[id \leftarrow \theta', exp \leftarrow e'] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

$$\begin{array}{l}\textsc{Trans Async P}\\ \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{async}\ e'])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{async}(e'))} \langle \sigma_A, E.(\theta', e'), \mathsf{unit}\rangle\end{array}$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{async}\ e'])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{async}\ e']$.

Then after the transitions we have that $\{\langle T \mid t[exp \leftarrow \mathcal{P}[\mathsf{unit}]] \mid t'[id \leftarrow \theta', exp \leftarrow e'] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta', e'), \mathsf{unit}\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T), e' correspond to t' (because $t'.\mathsf{exp} = e'$ and both have been added with the last transition) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

  $$T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{async}(e'))),\phi,(\theta,\mathsf{unp}(\mathsf{async}(e')))\rangle} T \mid t[exp \hookleftarrow \mathcal{U}[\mathsf{unit}]] \mid t'[id \hookleftarrow \theta', exp \hookleftarrow e'] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma$$

  in Abadi.system we have:

  Trans Async U
  $$\langle \sigma_A, E.(\theta,\mathcal{U}[\mathsf{async}\ e']).E', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{async}(e')))} \langle \sigma_A, E.(\theta', e').(\theta,\mathcal{U}[\mathsf{unit}]).E', \mathsf{unit}\rangle$$

  We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta,\mathcal{U}[\mathsf{async}\ e']).E', \mathsf{unit}\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[\mathsf{async}\ e']$.

  Then after the transitions we have that $\{\langle T \mid t[exp \hookleftarrow \mathcal{U}[\mathsf{unit}]] \mid t'[id \hookleftarrow \theta', exp \hookleftarrow e'] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta', e').(\theta,\mathcal{U}[\mathsf{unit}]).E', \mathsf{unit}\rangle\} \in Rel^{-1}$.

  This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), $\mathcal{U}[\mathsf{unit}]$ correspond to the thread t (because $t.\mathsf{exp} = \mathcal{U}[\mathsf{unit}]$), e' correspond to t' (because $t'.\mathsf{exp} = e'$ and both have been added with the last transition) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

  $$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{no\ unit},\mathsf{blockUntil}(\mathsf{true})),\phi,(\theta,\mathsf{no\ unit},\mathsf{blockUntil}(\mathsf{true}))\rangle} T \mid t[\mathsf{exp} \hookleftarrow \mathcal{P}[\mathsf{unit}]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma$$

  in Abadi.system we have:

  Trans Block P
  $$\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{blockUntil\ true}])\rangle \xmapsto{(\theta,\mathsf{no\ unit},\mathsf{blockUntil}(\mathsf{true}))} \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{unit}])\rangle$$

  We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{blockUntil\ true}])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{blockUntil\ true}]$.

  Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \hookleftarrow \mathcal{P}[\mathsf{unit}]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{unit}])\rangle\} \in Rel^{-1}$.

  This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $\mathcal{P}[\mathsf{unit}]$ correspond to the thread t (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$).

- if in our system we have this transition:

  $$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{blockUntil}(\mathsf{true})),\phi,(\theta,\mathsf{unit},\mathsf{blockUntil}(\mathsf{true}))\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

$$\textsc{Trans Block P}$$
$$\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{blockUntil\ true}]) \rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{blockUntil(true)})} \langle \sigma_A, E, \mathsf{unit} \rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{blockUntil\ true}]) \rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{blockUntil\ true}]$.

Then after the transitions we have that $\{\langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \rangle, \langle \sigma_A, E, \mathsf{unit} \rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp(blockUntil\ true)}),\phi,(\theta,\mathsf{unp(blockUntil\ true)})\rangle} T \mid t[exp \leftarrow\!\!\shortmid \mathcal{U}[\mathsf{unit}] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

$$\textsc{Trans Block U}$$
$$\langle \sigma_A, E.(\theta,\mathcal{U}[\mathsf{blockUntil\ true}]).E', \mathsf{unit} \rangle \xmapsto{(\theta,\mathsf{unp(blockUntil(true))})} \langle \sigma_A, E.(\theta,\mathcal{U}[\mathsf{unit}]).E', \mathsf{unit} \rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \rangle, \langle \sigma_A, E.(\theta,\mathcal{U}[\mathsf{blockUntil\ true}]).E', \mathsf{unit} \rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[\mathsf{blockUntil\ true}]$.

Then after the transitions we have that $\{\langle T \mid t[exp \leftarrow\!\!\shortmid \mathcal{U}[\mathsf{unit}] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \rangle, \langle \sigma_A, E.(\theta,\mathcal{U}[\mathsf{unit}]).E', \mathsf{unit} \rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), $\mathcal{U}[\mathsf{unit}]$ correspond to the thread t (because $t.\mathsf{exp} = \mathcal{U}[\mathsf{unit}]$) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unprtect}),\phi,(\theta,\mathsf{unprtect})\rangle} T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

$$\textsc{Trans Unprotected}$$
$$\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{unprotected}\ e]) \rangle \xmapsto{(\theta,\mathsf{unprotect})} \langle \sigma_A, E.(\theta, \mathcal{P}[\mathsf{unprotected}\ e]), \mathsf{unit} \rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{unprotected}\ e]) \rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{unprotected}\ e]$.

Then after the transitions we have that $\{\langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta, \mathcal{P}[\mathsf{unprotected}\ e]), \mathsf{unit}\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T), $\mathcal{P}[\mathsf{unprotected}\ e]$ correspond to the thread t (because $t.\mathsf{exp} = \mathcal{P}[\mathsf{unprotected}\ e]$) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(V)),\phi,(\theta,\mathsf{unp}(V))\rangle} T \mid t[exp \leftarrow\!\shortmid \mathcal{E}[V]] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

Trans Close
$$\langle \sigma_A, E.(\theta, \mathcal{E}[\mathsf{unprotected}\ V]).E', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{unp}(V))} \langle \sigma_A, E.(\theta, \mathcal{E}[V]).E', \mathsf{unit}\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta, \mathcal{E}[\mathsf{unprotected}\ V]).E', \mathsf{unit}\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{E}[\mathsf{unprotected}\ V]$.

Then after the transitions we have that $\{\langle T \mid t[exp \leftarrow\!\shortmid \mathcal{E}[V]] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta, \mathcal{E}[V]).E', \mathsf{unit}\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), $\mathcal{E}[V]$ correspond to the thread t (because $t.\mathsf{exp} = \mathcal{E}[V]$) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

□

B.3.2 Weak Semantics

In Fig. B.4 we show the original weak semantics for AME described in [16].

For a deeper discussion about the semantics we refer to [16].

While in Fig. B.5 and Fig. B.6 we refer to the updated semantics decorated with labels and with unique identifier for expressions. We can note that for each protected action we consider to transition one if the evaluated expression is evaluated to unit and one for the other cases. This because we need to distinguish these cases but it does not change the result of the original semantics, in fact if we delete the labels and the unique identifiers for the expression we will obtain the semantics described in [16].

Trans Appl P
$\langle \sigma, T, \mathcal{P}[(\lambda\ x.e)\ V] \rangle \longmapsto \langle \sigma, T, \mathcal{P}[e[V/x]] \rangle$

Trans Appl U
$\langle \sigma, T.\mathcal{U}[(\lambda\ x.e)\ V].T', e \rangle \longmapsto \langle \sigma, T.\mathcal{U}[e[V/x]].T', e \rangle$

Trans Ref P
$\langle \sigma, T, \mathcal{P}[\mathsf{ref}\ V] \rangle \longmapsto \langle \sigma[r \mapsto V], T, \mathcal{P}[r] \rangle \quad \text{if } r \in RefLoc\ -\ dom(\sigma)$

Trans Ref U
$\langle \sigma, T.\mathcal{U}[\mathsf{ref}\ V].T', e \rangle \longmapsto \langle \sigma[r \mapsto V], T.\mathcal{U}[r].T', e \rangle \quad \text{if } r \in RefLoc\ -\ dom(\sigma)$

Trans Deref P
$\langle \sigma, T, \mathcal{P}[!r] \rangle \longmapsto \langle \sigma, T, \mathcal{P}[V] \rangle \quad \text{if } \sigma(r) = V$

Trans Deref U
$\langle \sigma, T.\mathcal{U}[!r].T', e \rangle \longmapsto \langle \sigma, T.\mathcal{U}[V].T', e \rangle \quad \text{if } \sigma(r) = V$

Trans Set P
$\langle \sigma, T, \mathcal{P}[r := V] \rangle \longmapsto \langle \sigma[r \mapsto V], T, \mathcal{P}[\mathsf{unit}] \rangle$

Trans Set U
$\langle \sigma, T.\mathcal{U}[r := V].T', e \rangle \longmapsto \langle \sigma[r \mapsto V], T.\mathcal{U}[\mathsf{unit}].T', e \rangle$

Trans Async P
$\langle \sigma, T, \mathcal{P}[\mathsf{async}\ e] \rangle \longmapsto \langle \sigma, T.e, \mathcal{P}[\mathsf{unit}] \rangle$

Trans Async U
$\langle \sigma, T.\mathcal{U}[\mathsf{async}\ e].T', e \rangle \longmapsto \langle \sigma, T.e.\mathcal{U}[\mathsf{unit}].T', e \rangle$

Trans Block P
$\langle \sigma, T, \mathcal{P}[\mathsf{blockUntil\ true}] \rangle \longmapsto \langle \sigma, T, \mathcal{P}[\mathsf{unit}] \rangle$

Trans Block U
$\langle \sigma, T.\mathcal{U}[\mathsf{blockUntil\ true}].T', e \rangle \longmapsto \langle \sigma, T.\mathcal{U}[\mathsf{unit}].T', e \rangle$

Trans Unprotected
$\langle \sigma, T, \mathcal{P}[\mathsf{unprotected}\ e] \rangle \longmapsto \langle \sigma, T.\mathcal{P}[\mathsf{unprotected}\ e], \mathsf{unit} \rangle$

Trans Close
$\langle \sigma, T.\mathcal{E}[\mathsf{unprotected}\ V].T', e \rangle \longmapsto \langle \sigma, T.\mathcal{E}[V].T', e \rangle$

Trans Activate
$\langle \sigma, T.e.T', \mathsf{unit} \rangle \longmapsto \langle \sigma, T.T', e \rangle$

Figure B.4: Weak Semantics Abadi [16]

$$\textsc{Trans Appl P}\quad \langle \sigma, T, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle \xmapsto{(\theta,\text{no unit,pure})} \langle \sigma, T, (\theta, \mathcal{P}[e[V/x]])\rangle$$

$$\textsc{Trans Appl P}\quad \langle \sigma, T, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle \xmapsto{(\theta,\text{unit,pure})} \langle \sigma, T, \text{unit}\rangle$$

$$\textsc{Trans Appl U}\quad \langle \sigma, T.(\theta,\mathcal{U}[(\lambda\ x.e)\ V]).T', (\theta', e')\rangle \xmapsto{(\theta,\text{unp(pure)})} \langle \sigma, T.(\theta,\mathcal{U}[e[V/x]]).T', (\theta', e')\rangle$$

$$\textsc{Trans Ref P}\quad \langle \sigma, T, (\theta, \mathcal{P}[\text{ref } V])\rangle \xmapsto{(\theta,\text{no unit,ref}(r,v))} \langle \sigma[r \mapsto V], T, (\theta, \mathcal{P}[r])\rangle \text{ if } r \in RefLoc\ -\ dom(\sigma)$$

$$\textsc{Trans Ref P}\quad \langle \sigma, T, (\theta, \mathcal{P}[\text{ref } V])\rangle \xmapsto{(\theta,\text{unit,ref}(r,v))} \langle \sigma[r \mapsto V], T, \text{unit}\rangle \quad \text{if } r \in RefLoc\ -\ dom(\sigma)$$

$$\textsc{Trans Ref U}\quad \langle \sigma, T.(\theta,\mathcal{U}[\text{ref } V]).T', (\theta', e)\rangle \xmapsto{(\theta,\text{unp(ref}(r,v)))} \langle \sigma[r \mapsto V], T.(\theta,\mathcal{U}[r]).T', (\theta', e)\rangle \text{ if } r \in RefLoc\ -\ dom(\sigma)$$

$$\textsc{Trans Deref P}\quad \langle \sigma, T, (\theta, \mathcal{P}[!r])\rangle \xmapsto{(\theta,\text{no unit,rd}(r,v))} \langle \sigma, T, (\theta, \mathcal{P}[V])\rangle \quad \text{if } \sigma(r) = V$$

$$\textsc{Trans Deref P}\quad \langle \sigma, T, (\theta, \mathcal{P}[!r])\rangle \xmapsto{(\theta,\text{unit,rd}(r,v))} \langle \sigma, T, \text{unit}\rangle \quad \text{if } \sigma(r) = V$$

$$\textsc{Trans Deref U}\quad \langle \sigma, T.(\theta,\mathcal{U}[!r]).T', (\theta', e)\rangle \xmapsto{(\theta,\text{unp(rd}(r,v)))} \langle \sigma, T.(\theta,\mathcal{U}[V]).T', (\theta', e)\rangle \quad \text{if } \sigma(r) = V$$

$$\textsc{Trans Set P}\quad \langle \sigma, T, (\theta, \mathcal{P}[r := V])\rangle \xmapsto{(\theta,\text{no unit,wr}(r,v))} \langle \sigma[r \mapsto V], T, (\theta, \mathcal{P}[\text{unit}])\rangle$$

$$\textsc{Trans Set P}\quad \langle \sigma, T, (\theta, \mathcal{P}[r := V])\rangle \xmapsto{(\theta,\text{unit,wr}(r,v))} \langle \sigma[r \mapsto V], T, \text{unit}\rangle$$

$$\textsc{Trans Set U}\quad \langle \sigma, T.(\theta,\mathcal{U}[r := V]).T', (\theta', e)\rangle \xmapsto{(\theta,\text{unp(wr}(r,v)))} \langle \sigma[r \mapsto V], T.(\theta,\mathcal{U}[\text{unit}]).T', (\theta', e)\rangle$$

$$\textsc{Trans Async P}\quad \langle \sigma, T, (\theta, \mathcal{P}[\text{async } e])\rangle \xmapsto{(\theta,\text{no unit,async}(e))} \langle \sigma, T.(\theta', e), (\theta, \mathcal{P}[\text{unit}])\rangle$$

$$\textsc{Trans Async P}\quad \langle \sigma, T, (\theta, \mathcal{P}[\text{async } e])\rangle \xmapsto{(\theta,\text{unit,async}(e))} \langle \sigma, T.(\theta', e), \text{unit}\rangle$$

$$\textsc{Trans Async U}\quad \langle \sigma, T.(\theta,\mathcal{U}[\text{async } e]).T', (\theta'', e'')\rangle \xmapsto{(\theta,\text{unp(async}(e)))} \langle \sigma, T.(\theta', e).(\theta,\mathcal{U}[\text{unit}]).T', (\theta'', e'')\rangle$$

$$\textsc{Trans Block P}\quad \langle \sigma, T, (\theta, \mathcal{P}[\text{blockUntil true}])\rangle \xmapsto{(\theta,\text{no unit,blockUntil(true)})} \langle \sigma, T, (\theta, \mathcal{P}[\text{unit}])\rangle$$

$$\textsc{Trans Block P}\quad \langle \sigma, T, (\theta, \mathcal{P}[\text{blockUntil true}])\rangle \xmapsto{(\theta,\text{unit,blockUntil(true)})} \langle \sigma, T, \text{unit}\rangle$$

$$\textsc{Trans Block U}\quad \langle \sigma, T.(\theta,\mathcal{U}[\text{blockUntil true}]).T', (\theta', e)\rangle \xmapsto{(\theta,\text{unp(blockUntil(true)))}} \langle \sigma, T.(\theta,\mathcal{U}[\text{unit}]).T', (\theta', e)\rangle$$

Figure B.5: LTS Abadi weak sem, [16]

$$\textsc{Trans Unprotected}\quad \langle \sigma, T, (\theta, \mathcal{P}[\mathsf{unprotected}\ e])\rangle \xmapsto{(\theta,\mathsf{unprotect})} \langle \sigma, T.(\theta, \mathcal{P}[\mathsf{unprotected}\ e]), \mathsf{unit}\rangle$$

$$\textsc{Trans Close}\quad \langle \sigma, T.(\theta, \mathcal{E}[\mathsf{unprotected}\ V]).T', (\theta', e)\rangle \xmapsto{(\theta,\mathsf{unp}(V))} \langle \sigma, T.(\theta, \mathcal{E}[V]).T', (\theta', e)\rangle$$

$$\textsc{Trans Activate}\quad \langle \sigma, T.(\theta, e).T', \mathsf{unit}\rangle \xmapsto{(\theta,\text{no unit,act})} \langle \sigma, T.T', (\theta, e)\rangle$$

$$\textsc{Trans Activate}\quad \langle \sigma, T.(\theta, e).T', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{unit,act})} \langle \sigma, T.T', \mathsf{unit}\rangle$$

Figure B.6: LTS Abadi weak sem, [16]

Theorem 11. $\langle \sigma_A, E, e\rangle \sim (T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma)$

$$\begin{array}{ccc} \langle \sigma_A, E, e\rangle & \underset{Rel^{-1}}{\overset{Rel}{\longleftrightarrow}} & (T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma) \\ \Big\downarrow{\scriptstyle \varphi(\alpha)} & & \Big\downarrow{\scriptstyle \alpha} \\ \langle \sigma'_A, E', e'\rangle & \underset{Rel^{-1}}{\overset{Rel}{\longleftrightarrow}} & (T' \otimes_{\mathcal{R}} S' \otimes_{\mathcal{R}} \sigma') \end{array}$$

Proof. We start by defining Rel and then show that it is a strong similarity relation, after which we will show that Rel^{-1} is also a strong similarity relation.

Given $\{\langle \sigma_A, E, e\rangle, \langle T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\}$ then this tuple is in Rel if:

- $\sigma = \sigma_A$
- $\forall e_i \in E$ exist θ_i such that $t_i \in T$ and $t_i.\mathsf{id} = \theta_i \wedge t_i.\mathsf{exp} = e_i$
- if $e = \mathsf{unit}$ then $S = \bullet$ otherwise (if $e \neq \mathsf{unit}$) exist $t \in T$ where $t.\mathsf{id} = \theta \wedge t.\mathsf{exp} = e$ and $S = \theta$
- if in Abadi.system we have this transition:

  $$\textsc{Trans Activate}\quad \langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle \xmapsto{(\theta,\text{no unit,act})} \langle \sigma_A, E.E', (\theta, e)\rangle$$

  in our system we have:

  $$\frac{\dfrac{\textsc{Activate1}\quad t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = e}{T \mid t \xrightarrow{(\theta,\text{no unit,act})} T \mid t}\ e \neq \mathsf{unit} \qquad \dfrac{\textsc{Activate1}\quad S = \bullet}{S \xrightarrow{(\theta,\text{no unit,act})} \theta}}{T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\text{no unit,act}),\phi,(\theta,\text{no unit,act})\rangle} T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma}$$

  We know that $\{\langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle, \langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$.

  Then after the transitions we have that $\{\langle \sigma_A, E.E', (\theta, e)\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

  This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the

expressions in $E.E'$ have a correspond thread in T (none of the rules applied to the two systems changed either $E.E'$ or T) and the thread t correspond to e (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$).

- if in Abadi.system we have this transition:

$$\frac{}{\langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{act})} \langle \sigma_A, E.E', \mathsf{unit}\rangle}\ \textsc{Trans Activate}$$

in our system we have:

$$\frac{\dfrac{\textsc{Activate2}\quad t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathsf{unit}}{T \mid t \xrightarrow{(\theta,\mathsf{unit},\mathsf{act})} T}\ e = \mathsf{unit} \qquad \dfrac{\textsc{Activate2}\quad S = \bullet}{S \xrightarrow{(\theta,\mathsf{unit},\mathsf{act})} S}}{T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{act}),\phi,(\theta,\mathsf{unit},\mathsf{act})\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle, \langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathsf{unit}$.

Then after the transitions we have that $\{\langle \sigma_A, E.E', \mathsf{unit}\rangle, \langle T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in $E.E'$ have a correspond thread in T (none of the rules applied to the two systems changed either $E.E'$ or T) and $S = \bullet$ because $e = \mathsf{unit}$ (and in fact also $t.\mathsf{exp} = \mathsf{unit}$).

- if in Abadi.system we have this transition:

$$\frac{}{\langle \sigma_A, E, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle \xmapsto{(\theta,\text{no unit},\mathsf{pure})} \langle \sigma_A, E, (\theta, \mathcal{P}[e[V/x]])\rangle}\ \textsc{Trans Appl P}$$

in our system we have:

$$\frac{\dfrac{\textsc{Step P1}\quad t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[(\lambda\ x.e)\ V] \qquad (\lambda\ x.e)\ V \xrightarrow{\mathsf{pure}} e[V/x]}{T \mid t \xrightarrow{(\theta,\text{no unit},\mathsf{pure})} T \mid t[exp \leftarrowtail \mathcal{P}[e[V/x]]]}\ \mathcal{P}[e[V/x]] \neq \mathsf{unit} \qquad \dfrac{\textsc{Step P1}\quad S = \theta}{S \xrightarrow{(\theta,\text{no unit},\mathsf{pure})} S}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\text{no unit},\mathsf{pure}),\phi,(\theta,\text{no unit},\mathsf{pure})\rangle} T \mid t[\mathsf{exp} \leftarrowtail \mathcal{P}[e[V/x]]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle \sigma_A, E, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[(\lambda\ x.e)\ V]$.

Then after the transitions we have that $\{\langle \sigma_A, E, (\theta, \mathcal{P}[e[V/x]])\rangle, \langle T \mid t[\mathsf{exp} \leftarrowtail \mathcal{P}[e[V/x]]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $\mathcal{P}[e[V/x]]$ (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[e[V/x]]$).

- if in Abadi.system we have this transition:

$$\frac{}{\langle \sigma_A, E, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{pure})} \langle \sigma_A, E, \mathsf{unit}\rangle}\ \textsc{Trans Appl P}$$

in our system we have:

$$\frac{\dfrac{\textsc{Step P2}\quad t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[(\lambda\ x.e)\ V] \qquad (\lambda\ x.e)\ V \xhookrightarrow{\mathsf{pure}} e[V/x]}{T \mid t \xrightarrow{(\theta,\mathsf{unit},\mathsf{pure})} T} \qquad \dfrac{\textsc{Step P2}\quad S = \theta \qquad \mathcal{P}[e[V/x]] = \mathsf{unit}}{S \xrightarrow{(\theta,\mathsf{unit},\mathsf{pure})} \bullet}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{pure}),\phi,(\theta,\mathsf{unit},\mathsf{pure})\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle \sigma_A, E, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[(\lambda\ x.e)\ V]$.

Then after the transitions we have that $\{\langle \sigma_A, E, \mathsf{unit}\rangle, \langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ because $\mathcal{P}[e[V/x]] = \mathsf{unit}$.

- if in Abadi.system we have this transition:

$$\frac{}{\langle \sigma_A, E.(\theta,\mathcal{U}[(\lambda\ x.e)\ V]).E', (\theta', e')\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{pure}))} \langle \sigma_A, E.(\theta,\mathcal{U}[e[V/x]]).E', (\theta', e')\rangle}\ \textsc{Trans Appl U}$$

in our system we have:

$$\frac{\dfrac{\textsc{Step U}\quad t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{U}[(\lambda\ x.e)\ V] \qquad (\lambda\ x.e)\ V \xhookrightarrow{\mathsf{pure}} e[V/x]}{T \mid t \xrightarrow{(\theta,\mathsf{unp}(\mathsf{pure}))} T \mid t[exp \leftarrow \mathcal{U}[e[V/x]]]} \qquad \dfrac{\textsc{Step U/Close}\quad S \neq \theta}{S \xrightarrow{(\theta,\mathsf{unp}(\mathsf{pure}))} S}}{T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{pure})),\phi,(\theta,\mathsf{unp}(\mathsf{pure}))\rangle} T \mid t[exp \leftarrow \mathcal{U}[e[V/x]]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle \sigma_A, E.(\theta,\mathcal{U}[(\lambda\ x.e)\ V]).E', (\theta', e')\rangle, \langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[(\lambda\ x.e)\ V]$.

Then after the transitions we have that $\{\langle \sigma_A, E.(\theta,\mathcal{U}[e[V/x]]).E', (\theta', e')\rangle, \langle T \mid t[\mathsf{exp} \leftarrow \mathcal{U}[e[V/x]]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation *Rel* because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[e[V/x]]$ (because $t.\mathsf{exp} = \mathcal{U}[e[V/x]]$) and S correspond to e' in fact the two rules do not change them.

- if in Abadi.system we have this transition:

$$\textsc{Trans Ref P}\quad \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{ref}\ V])\rangle \xmapsto{(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v))} \langle \sigma_A[r \mapsto V], E, (\theta, \mathcal{P}[r])\rangle \quad \mathsf{if}\ r \in RefLoc\ -\ dom(\sigma)$$

in our system we have:

$$\dfrac{\dfrac{\textsc{Step P1}\quad \dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[\mathsf{ref}\ v] \qquad \mathsf{ref}\ v \xhookrightarrow{\mathsf{ref}(r,v)} r}{T \mid t \xrightarrow{(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v))} T \mid t[exp \leftarrow\!\shortmid \mathcal{P}[r]]}\ \mathcal{P}[r] \neq \mathsf{unit} \qquad \textsc{Step P1}\quad \dfrac{S = \theta}{S \xrightarrow{(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v))} S} \qquad \textsc{Ref}\quad \dfrac{r \in (RefLoc - \mathsf{dom}(\sigma))}{\sigma \xrightarrow{(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v))} \sigma[r \mapsto v]}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v)),(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v)),(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v))\rangle} T \mid t[\mathsf{exp} \leftarrow\!\shortmid \mathcal{P}[r]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma[r \mapsto v]}$$

We know that $\{\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{ref}\ v])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{ref}\ v]$.

Then after the transitions we have that $\{\langle \sigma_A[r \mapsto v], E, (\theta, \mathcal{P}[r])\rangle,\ \langle T \mid t[\mathsf{exp} \leftarrow\!\shortmid \mathcal{P}[r]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle\} \in Rel$.

This couple belongs to the relation *Rel* because we have that $\sigma = \sigma_A$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $\mathcal{P}[r]$ (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[r]$).

- if in Abadi.system we have this transition:

$$\textsc{Trans Ref P}\quad \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{ref}\ V])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{ref}(r,v))} \langle \sigma_A[r \mapsto V], E, \mathsf{unit}\rangle \qquad \mathsf{if}\ r \in RefLoc\ -\ dom(\sigma)$$

in our system we have:

$$\dfrac{\dfrac{\textsc{Step P2}}{t.\mathsf{id}=\theta \qquad t.\mathsf{exp}=\mathcal{P}[\mathsf{ref}\ v] \qquad \mathsf{ref}\ v \xhookrightarrow{\mathsf{ref}(r,v)} r}{T \mid t \xrightarrow{(\theta,\mathsf{unit},\mathsf{ref}(r,v))} T}\ \mathcal{P}[r]=\mathsf{unit} \qquad \dfrac{\dfrac{\textsc{Step P2}}{S=\theta}}{S \xrightarrow{(\theta,\mathsf{unit},\mathsf{ref}(r,v))} \bullet} \qquad \dfrac{\dfrac{\textsc{Ref}}{r \in (RefLoc - \mathsf{dom}(\sigma))}}{\sigma \xrightarrow{(\theta,\mathsf{unit},\mathsf{ref}(r,v))} \sigma[r \mapsto v]}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{ref}(r,v)),(\theta,\mathsf{unit},\mathsf{ref}(r,v)),(\theta,\mathsf{unit},\mathsf{ref}(r,v))\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]}$$

We know that $\{\langle\sigma_A, E, (\theta, \mathcal{P}[\mathsf{ref}\ v])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{ref}\ v]$.

Then after the transitions we have that $\{\langle\sigma_A[r \mapsto v], E, \mathsf{unit}\rangle, \langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ because $\mathcal{P}[r] = \mathsf{unit}$.

- if in Abadi.system we have this transition:

$$\textsc{Trans Ref U}$$
$$\langle\sigma_A, E.(\theta,\mathcal{U}[\mathsf{ref}\ V]).E', (\theta', e)\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{ref}(r,v)))} \langle\sigma_A[r \mapsto V], E.(\theta,\mathcal{U}[r]).E', (\theta', e)\rangle \quad \mathsf{if}\ r \in RefLoc\ -\ dom(\sigma)$$

in our system we have:

$$\dfrac{\dfrac{\textsc{Step U}}{t.\mathsf{id}=\theta \qquad t.\mathsf{exp}=\mathcal{U}[\mathsf{ref}\ V] \qquad \mathsf{ref}\ V \xhookrightarrow{\mathsf{ref}(r,v)} r}{T \mid t \xrightarrow{(\theta,\mathsf{unp}(\mathsf{ref}(r,v)))} T \mid t[exp \leftarrowtail \mathcal{U}[r]]} \qquad \dfrac{\dfrac{\textsc{Step U/Close}}{S \neq \theta}}{S \xrightarrow{(\theta,\mathsf{unp}(\mathsf{ref}(r,v)))} S} \qquad \dfrac{\dfrac{\textsc{URef}}{r \in (RefLoc - \mathsf{dom}(\sigma))}}{\sigma \xrightarrow{(\theta,\mathsf{unp}(\mathsf{ref}(r,v)))} \sigma[r \mapsto v]}}{T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{ref}(r,v))),(\theta,\mathsf{unp}(\mathsf{ref}(r,v))),(\theta,\mathsf{unp}(\mathsf{ref}(r,v)))\rangle} T \mid t[exp \leftarrowtail \mathcal{U}[r]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma[r \mapsto v]}$$

We know that $\{\langle\sigma_A, E.(\theta,\mathcal{U}[\mathsf{ref}\ V]).E', (\theta', e)\rangle, \langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[\mathsf{ref}\ v]$.

Then after the transitions we have that $\{\langle\sigma_A[r \mapsto V], E.(\theta,\mathcal{U}[r]).E', (\theta', e)\rangle, \langle T \mid t[exp \leftarrowtail \mathcal{U}[r]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to

the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[r]$ (because $t.\mathsf{exp} = \mathcal{U}[r]$) and S correspond to e in fact the two rules do not change them.

- if in Abadi.system we have this transition:

$$\frac{}{\langle \sigma_A, E, (\theta, \mathcal{P}[!r]) \rangle \xmapsto{(\theta,\text{no unit},\mathsf{rd}(r,v))} \langle \sigma_A, E, (\theta, \mathcal{P}[V]) \rangle} \text{Trans Deref P} \quad \text{if } \sigma(r) = V$$

in our system we have:

$$\frac{\dfrac{\dfrac{}{t.\mathsf{id} = \theta} \quad \dfrac{}{t.\mathsf{exp} = \mathcal{P}[!r]} \quad \dfrac{}{!r \xhookrightarrow{\mathsf{rd}(r,v)} v}}{T \mid t \xrightarrow{(\theta,\text{no unit},\mathsf{rd}(r,v))} T \mid t[exp \leftarrowtail \mathcal{P}[v]]}\ \text{Step P1}\ \mathcal{P}[v] \neq \mathsf{unit} \qquad \dfrac{\dfrac{}{S = \theta}}{S \xrightarrow{(\theta,\text{no unit},\mathsf{rd}(r,v))} S}\ \text{Step P1} \qquad \dfrac{\dfrac{}{\sigma(r) = v}}{\sigma \xrightarrow{(\theta,\text{no unit},\mathsf{rd}(r,v))} \sigma}\ \text{DeRef (Read)}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\text{no unit},\mathsf{rd}(r,v)),(\theta,\text{no unit},\mathsf{rd}(r,v)),(\theta,\text{no unit},\mathsf{rd}(r,v))\rangle} T \mid t[\mathsf{exp} \leftarrowtail \mathcal{P}[v]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle \sigma_A, E, (\theta, \mathcal{P}[!r]) \rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[!r]$.

Then after the transitions we have that $\{\langle \sigma_A, E, (\theta, \mathcal{P}[V]) \rangle, \langle T \mid t[\mathsf{exp} \leftarrowtail \mathcal{P}[V]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $\mathcal{P}[v]$ (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[v]$).

- if in Abadi.system we have this transition:

$$\frac{}{\langle \sigma_A, E, (\theta, \mathcal{P}[!r]) \rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{rd}(r,v))} \langle \sigma_A, E, \mathsf{unit} \rangle} \text{Trans Deref P} \quad \text{if } \sigma(r) = V$$

in our system we have:

$$\frac{\dfrac{\dfrac{}{t.\mathsf{id} = \theta} \quad \dfrac{}{t.\mathsf{exp} = \mathcal{P}[!r]} \quad \dfrac{}{!r \xhookrightarrow{\mathsf{rd}(r,v)} v}}{T \mid t \xrightarrow{(\theta,\mathsf{unit},\mathsf{rd}(r,v))} T}\ \text{Step P2}\ \mathcal{P}[v] = \mathsf{unit} \qquad \dfrac{\dfrac{}{S = \theta}}{S \xrightarrow{(\theta,\mathsf{unit},\mathsf{rd}(r,v))} \bullet}\ \text{Step P2} \qquad \dfrac{\dfrac{}{\sigma(r) = v}}{\sigma \xrightarrow{(\theta,\mathsf{rd}(r,v))} \sigma}\ \text{DeRef (Read)}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{rd}(r,v)),(\theta,\mathsf{unit},\mathsf{rd}(r,v)),(\theta,\mathsf{unit},\mathsf{rd}(r,v))\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle\sigma_A, E, (\theta, \mathcal{P}[!r])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[!r]$.

Then after the transitions we have that $\{\langle\sigma_A, E, \mathsf{unit}\rangle, \langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ because $\mathcal{P}[v] = \mathsf{unit}$.

- if in Abadi.system we have this transition:

  Trans Deref U

  $$\langle\sigma_A, E.(\theta, \mathcal{U}[!r]).E', (\theta', e)\rangle \xmapsto{(\theta, \mathsf{unp}(\mathsf{rd}(r,v)))} \langle\sigma_A, E.(\theta, \mathcal{U}[V]).E', (\theta', e)\rangle \qquad \text{if } \sigma(r) = V$$

  in our system we have:

$$\frac{\dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{U}[!r] \qquad !r \xhookrightarrow{\mathsf{rd}(r,v)} v}{T \mid t \xrightarrow{(\theta, \mathsf{unp}(\mathsf{rd}(r,v)))} T \mid t[exp \leftarrowtail \mathcal{U}[v]]} \qquad \dfrac{S \neq \theta}{S \xrightarrow{(\theta, \mathsf{unp}(\mathsf{rd}(r,v)))} S} \qquad \dfrac{\sigma(r) = v}{\sigma \xrightarrow{(\theta, \mathsf{unp}(\mathsf{rd}(r,v)))} \sigma}}{T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta, \mathsf{unp}(\mathsf{rd}(r,v))), (\theta, \mathsf{unp}(\mathsf{rd}(r,v))), (\theta, \mathsf{unp}(\mathsf{rd}(r,v)))\rangle} T \mid t[\mathsf{exp} \leftarrowtail \mathcal{U}[v]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma}$$

Step U / Step U/Close / UDeRef (Read)

We know that $\{\langle\sigma_A, E.(\theta, \mathcal{U}[!r]).E', (\theta', e)\rangle, \langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[!r]$.

Then after the transitions we have that $\{\langle\sigma_A, E.(\theta, \mathcal{U}[V]).E', (\theta', e)\rangle, \langle T \mid t[\mathsf{exp} \leftarrowtail \mathcal{U}[v]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[v]$ (because $t.\mathsf{exp} = \mathcal{U}[v]$) and S correspond to e in fact the two rules do not change them.

- if in Abadi.system we have this transition:

  Trans Set P

  $$\langle\sigma_A, E, (\theta, \mathcal{P}[r := V])\rangle \xmapsto{(\theta, \text{no unit}, \mathsf{wr}(r,v))} \langle\sigma_A[r \mapsto V], E, (\theta, \mathcal{P}[\mathsf{unit}])\rangle$$

  in our system we have:

$$\dfrac{\dfrac{\dfrac{}{S \xrightarrow{(\theta,\text{no unit},\mathsf{wr}(r,v))} S}\ \text{Step P1} \qquad \dfrac{}{\sigma \xrightarrow{(\theta,\text{no unit},\mathsf{wr}(r,v))} \sigma[r \mapsto v]}\ \text{Set (Write)}}{T \mid t \xrightarrow{(\theta,\text{no unit},\mathsf{wr}(r,v))} T \mid t[exp \leftarrow\!\shortmid \mathcal{P}[\mathsf{unit}]]}\ \text{Step P1}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\text{no unit},\mathsf{wr}(r,v)),(\theta,\text{no unit},\mathsf{wr}(r,v)),(\theta,\text{no unit},\mathsf{wr}(r,v))\rangle} T \mid t[\mathsf{exp} \leftarrow\!\shortmid \mathcal{P}[\mathsf{unit}]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma[r \mapsto v]}$$

We know that $\{\langle\sigma_A, E, (\theta, \mathcal{P}[r := V])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[r := v]$.

Then after the transitions we have that $\{\langle\sigma_A[r \mapsto V], E, (\theta, \mathcal{P}[\mathsf{unit}])\rangle, \langle T \mid t[\mathsf{exp} \leftarrow\!\shortmid \mathcal{P}[\mathsf{unit}]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $\mathcal{P}[\mathsf{unit}]$ (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$).

- if in Abadi.system we have this transition:

$$\dfrac{}{\langle\sigma_A, E, (\theta, \mathcal{P}[r := V])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{wr}(r,v))} \langle\sigma_A[r \mapsto V], E, \mathsf{unit}\rangle}\ \text{Trans Set P}$$

in our system we have:

$$\dfrac{\dfrac{\dfrac{}{S \xrightarrow{(\theta,\mathsf{unit},\mathsf{wr}(r,v))} \bullet}\ \text{Step P2} \qquad \dfrac{}{\sigma \xrightarrow{(\theta,\mathsf{unit},\mathsf{wr}(r,v))} \sigma[r \mapsto v]}\ \text{Set (Write)}}{T \mid t \xrightarrow{(\theta,\mathsf{unit},\mathsf{wr}(r,v))} T}\ \text{Step P2}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{wr}(r,v)),(\theta,\mathsf{unit},\mathsf{wr}(r,v)),(\theta,\mathsf{unit},\mathsf{wr}(r,v))\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]}$$

We know that $\{\langle\sigma_A, E, (\theta, \mathcal{P}[r := v])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[r := v]$.

Then after the transitions we have that $\{\langle\sigma_A[r \mapsto v], E, \mathsf{unit}\rangle, \langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ because $\mathcal{P}[\mathsf{unit}] = \mathsf{unit}$.

- if in Abadi.system we have this transition:

  Trans Set U
  $$\langle \sigma_A, E.(\theta, \mathcal{U}[r := V]).E', (\theta', e)\rangle \xrightarrow{(\theta, \mathsf{unp}(\mathsf{wr}(r,v)))} \langle \sigma_A[r \mapsto V], E.(\theta, \mathcal{U}[\mathsf{unit}]).E', (\theta', e)\rangle$$

  in our system we have:

$$\dfrac{\dfrac{\text{Step U}\quad t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{U}[r := v] \qquad r := v \xrightarrow{\mathsf{wr}(r,v)} \mathsf{unit}}{T \mid t \xrightarrow{(\theta, \mathsf{unp}(\mathsf{wr}(r,v)))} T \mid t[exp \leftarrow \mathcal{U}[\mathsf{unit}]]} \quad \dfrac{\text{Step U/Close}\quad S \neq \theta}{S \xrightarrow{(\theta, \mathsf{unp}(\mathsf{wr}(r,v)))} S} \quad \dfrac{\text{Set (Write)}}{\sigma \xrightarrow{(\theta, \mathsf{unp}(\mathsf{wr}(r,v)))} \sigma[r \mapsto v]}}{T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle (\theta, \mathsf{unp}(\mathsf{wr}(r,v))), (\theta, \mathsf{unp}(\mathsf{wr}(r,v))), (\theta, \mathsf{unp}(\mathsf{wr}(r,v)))\rangle} T \mid t[\mathsf{exp} \leftarrow \mathcal{U}[\mathsf{unit}]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma[r \mapsto v]}$$

  We know that $\{\langle \sigma_A, E.(\theta, \mathcal{U}[r := V]).E', (\theta', e)\rangle, \langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[r := v]$.

  Then after the transitions we have that $\{\langle \sigma_A[r \mapsto V], E.(\theta, \mathcal{U}[\mathsf{unit}]).E', (\theta', e)\rangle, \langle T \mid t[\mathsf{exp} \leftarrow \mathcal{U}[\mathsf{unit}]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle\} \in Rel$.

  This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[\mathsf{unit}]$ (because $t.\mathsf{exp} = \mathcal{U}[\mathsf{unit}]$) and S correspond to e in fact the two rules do not change them.

- if in Abadi.system we have this transition:

  Trans Async P
  $$\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{async}\ e'])\rangle \xrightarrow{(\theta, \mathsf{no\ unit}, \mathsf{async}(e'))} \langle \sigma_A, E.(\theta', e'), (\theta, \mathcal{P}[\mathsf{unit}])\rangle$$

  in our system we have:

$$\dfrac{\dfrac{\text{Spawn P1}\quad t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[\mathsf{async}\ e'] \qquad t', \theta'\ fresh}{T \mid t \xrightarrow{(\theta, \mathsf{no\ unit}, \mathsf{async}(e'))} T \mid t[exp \leftarrow \mathcal{P}[\mathsf{unit}]] \mid t'[id \leftarrow \theta', exp \leftarrow e']}\ \mathcal{P}[\mathsf{unit}] \neq \mathsf{unit} \quad \dfrac{\text{Step P1}\quad S = \theta}{S \xrightarrow{(\theta, \mathsf{no\ unit}, \mathsf{async}(e'))} S}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle (\theta, \mathsf{no\ unit}, \mathsf{async}(e')), \phi, (\theta, \mathsf{no\ unit}, \mathsf{async}(e'))\rangle} T \mid t[exp \leftarrow \mathcal{P}[\mathsf{unit}]] \mid t'[id \leftarrow \theta', exp \leftarrow e'] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma}$$

  We know that $\{\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{async}\ e'])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{async}\ e']$.

Then after the transitions we have that $\{\langle\sigma_A, E.(\theta', e'), (\theta, \mathcal{P}[\mathsf{unit}])\rangle, \langle T \mid t[\mathsf{exp} \leftarrow\!\!\shortmid \mathcal{P}[\mathsf{unit}]] \mid t'[id \leftarrow\!\!\shortmid \theta', exp \leftarrow\!\!\shortmid e'] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T), e correspond to t' (because $t'.\mathsf{exp} = e$ and both have been added with the last transition) and the thread t correspond to $\mathcal{P}[\mathsf{unit}]$ (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$).

- if in Abadi.system we have this transition:

$$\textsc{Trans Async P}\quad \langle\sigma_A, E, (\theta, \mathcal{P}[\mathsf{async}\ e'])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{async}(e'))} \langle\sigma_A, E.(\theta', e'), \mathsf{unit}\rangle$$

in our system we have:

$$\dfrac{\dfrac{\textsc{Spawn P2}\quad t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[\mathsf{async}\ e'] \qquad t', \theta'\ fresh}{T \mid t \xrightarrow{(\theta,\mathsf{unit},\mathsf{async}(e'))} T \mid t'[id \leftarrow\!\!\shortmid \theta', exp \leftarrow\!\!\shortmid e']}\ \mathcal{P}[\mathsf{unit}] = \mathsf{unit} \qquad \dfrac{\textsc{Step P2}\quad S = \theta}{S \xrightarrow{(\theta,\mathsf{unit},\mathsf{async}(e'))} \bullet}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{async}(e')),\phi,(\theta,\mathsf{unit},\mathsf{async}(e'))\rangle} T \mid t'[id \leftarrow\!\!\shortmid \theta', exp \leftarrow\!\!\shortmid e'] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle\sigma_A, E, (\theta, \mathcal{P}[\mathsf{async}\ e'])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{async}\ e']$.

Then after the transitions we have that $\{\langle\sigma_A, E.(\theta', e'), \mathsf{unit}\rangle, \langle T \mid t'[id \leftarrow\!\!\shortmid \theta', exp \leftarrow\!\!\shortmid e'] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T), e correspond to t' (because $t'.\mathsf{exp} = e$ and both have been added with the last transition) and $S = \bullet$ because $\mathcal{P}[\mathsf{unit}] = \mathsf{unit}$.

- if in Abadi.system we have this transition:

$$\textsc{Trans Async U}\quad \langle\sigma_A, E.(\theta, \mathcal{U}[\mathsf{async}\ e']).E', (\theta'', e'')\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{async}(e')))} \langle\sigma_A, E.(\theta', e').(\theta, \mathcal{U}[\mathsf{unit}]).E', (\theta'', e'')\rangle$$

in our system we have:

$$\frac{\dfrac{\textsc{Spawn U}\quad t.\mathsf{id}=\theta \qquad t.\mathsf{exp}=\mathcal{U}[\mathsf{async}\ e'] \qquad t',\theta'\ fresh}{T \mid t \xrightarrow{(\theta,\mathsf{unp}(\mathsf{async}(e')))} T \mid t[exp \leftarrow\!\shortmid \mathcal{U}[\mathsf{unit}]] \mid t'[id \leftarrow\!\shortmid \theta', exp \leftarrow\!\shortmid e']} \qquad \frac{\textsc{Step U/Close}\quad S \neq \theta}{S \xrightarrow{(\theta,\mathsf{unp}(\mathsf{async}(e')))} S}}{T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{async}(e'))),\phi,(\theta,\mathsf{unp}(\mathsf{async}(e')))\rangle} T \mid t[exp \leftarrow\!\shortmid \mathcal{U}[\mathsf{unit}]] \mid t'[id \leftarrow\!\shortmid \theta', exp \leftarrow\!\shortmid e'] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle\sigma_A, E.(\theta,\mathcal{U}[\mathsf{async}\ e']).E', (\theta'', e'')\rangle, \langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[\mathsf{async}\ e']$.

Then after the transitions we have that $\{\langle\sigma_A, E.(\theta', e').(\theta,\mathcal{U}[\mathsf{unit}]).E', (\theta'', e'')\rangle,\ \langle T \mid t[exp \leftarrow\!\shortmid \mathcal{U}[\mathsf{unit}]] \mid t'[id \leftarrow\!\shortmid \theta', exp \leftarrow\!\shortmid e'] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory.

We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[\mathsf{unit}]$ (because $t.\mathsf{exp} = \mathcal{U}[\mathsf{unit}]$), e' correspond to t' (because $t'.\mathsf{exp} = e'$ and both have been added with the last transition) and S correspond to e in fact the two rules do not change them.

- if in Abadi.system we have this transition:

$$\textsc{Trans Block P}$$
$$\langle\sigma_A, E, (\theta, \mathcal{P}[\mathsf{blockUntil\ true}])\rangle \xmapsto{(\theta,\mathsf{no\ unit},\mathsf{blockUntil(true)})} \langle\sigma_A, E, (\theta, \mathcal{P}[\mathsf{unit}])\rangle$$

in our system we have:

$$\frac{\dfrac{\dfrac{\textsc{Step P1}\quad t.\mathsf{id}=\theta \qquad t.\mathsf{exp}=\mathcal{P}[\mathsf{blockUntil\ true}] \qquad \mathsf{blockUntil\ true} \xrightarrow{\mathsf{blockUntil(true)}} \mathsf{unit}}{T \mid t \xrightarrow{(\theta,\mathsf{no\ unit},\mathsf{blockUntil(true)})} T \mid t[exp \leftarrow\!\shortmid \mathcal{P}[\mathsf{unit}]]}\ \mathcal{P}[\mathsf{unit}] \neq \mathsf{unit} \qquad \frac{\textsc{Step P1}\quad S=\theta}{S \xrightarrow{(\theta,\mathsf{no\ unit},\mathsf{blockUntil(true)})} S}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{no\ unit},\mathsf{blockUntil(true)}),\phi,(\theta,\mathsf{no\ unit},\mathsf{blockUntil(true)})\rangle} T \mid t[\mathsf{exp} \leftarrow\!\shortmid \mathcal{P}[\mathsf{unit}]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle\sigma_A, E, (\theta, \mathcal{P}[\mathsf{blockUntil\ true}])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{blockUntil\ true}]$.

Then after the transitions we have that $\{\langle\sigma_A, E, (\theta, \mathcal{P}[\mathsf{unit}])\rangle,\ \langle T \mid t[\mathsf{exp} \leftarrow\!\shortmid \mathcal{P}[\mathsf{unit}]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two

systems changed either E or T) and the thread t correspond to $\mathcal{P}[\mathsf{unit}]$ (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$).

- if in Abadi.system we have this transition:

$$\textsc{Trans Block P}\quad \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{blockUntil\ true}])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{blockUntil(true)})} \langle \sigma_A, E, \mathsf{unit}\rangle$$

in our system we have:

$$\dfrac{\dfrac{}{T \mid t \xrightarrow{(\theta,\mathsf{unit},\mathsf{blockUntil(true)})} T}\ \textsc{Step P2}\quad t.\mathsf{id} = \theta \quad t.\mathsf{exp} = \mathcal{P}[\mathsf{blockUntil\ true}] \quad \mathsf{blockUntil\ true} \xhookrightarrow{\mathsf{blockUntil(true)}} \mathsf{unit} \quad \mathcal{P}[\mathsf{unit}] = \mathsf{unit} \qquad \dfrac{}{S \xrightarrow{(\theta,\mathsf{unit},\mathsf{blockUntil(true)})} \bullet}\ \textsc{Step P2}\quad S = \theta}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{blockUntil(true)}),\phi,(\theta,\mathsf{unit},\mathsf{blockUntil(true)})\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{blockUntil\ true}])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in \mathit{Rel}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{blockUntil\ true}]$.

Then after the transitions we have that $\{\langle \sigma_A, E, \mathsf{unit}\rangle, \langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in \mathit{Rel}$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ because $\mathcal{P}[\mathsf{unit}] = \mathsf{unit}$.

- if in Abadi.system we have this transition:

$$\textsc{Trans Block U}\quad \langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{blockUntil\ true}]).E', (\theta', e)\rangle \xmapsto{(\theta,\mathsf{unp(blockUntil(true))})} \langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{unit}]).E', (\theta', e)\rangle$$

in our system we have:

$$\dfrac{\dfrac{}{T \mid t \xrightarrow{(\theta,\mathsf{unp(blockUntil\ true)})} T \mid t[\mathit{exp} \leftarrow \mathcal{U}[\mathsf{unit}]]}\ \textsc{Step U}\quad t.\mathsf{id} = \theta \quad t.\mathsf{exp} = \mathcal{U}[\mathsf{blockUntil\ true}] \quad \mathsf{blockUntil\ true} \xhookrightarrow{\mathsf{blockUntil\ true}} \mathsf{unit} \qquad \dfrac{}{S \xrightarrow{(\theta,\mathsf{unp(blockUntil\ true)})} S}\ \textsc{Step U/Close}\quad S \neq \theta}{T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp(blockUntil\ true)}),\phi,(\theta,\mathsf{unp(blockUntil\ true)})\rangle} T \mid t[\mathit{exp} \leftarrow \mathcal{U}[\mathsf{unit}] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{blockUntil\ true}]).E', (\theta', e)\rangle, \langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in \mathit{Rel}$ where

$t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[\mathsf{blockUntil\ true}]$.

Then after the transitions we have that $\{\langle \sigma_A, E.(\theta,\mathcal{U}[\mathsf{unit}]).E', (\theta', e)\rangle,\ \langle T \mid t[exp \leftarrow \mathcal{U}[\mathsf{unit}] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory.

We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[\mathsf{unit}]$ (because $t.\mathsf{exp} = \mathcal{U}[\mathsf{unit}]$) and S correspond to e in fact the two rules do not change them.

- if in Abadi.system we have this transition:

  Trans Unprotected
  $$\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{unprotected}\ e])\rangle \xmapsto{(\theta,\mathsf{unprotect})} \langle \sigma_A, E.(\theta, \mathcal{P}[\mathsf{unprotected}\ e]), \mathsf{unit}\rangle$$

  in our system we have:

$$\dfrac{\dfrac{\textsc{Unprotected}\quad t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[\mathsf{unprotected}\ e]}{T \mid t \xrightarrow{(\theta,\mathsf{unprtect})} T \mid t} \qquad \dfrac{\textsc{Unprotected}\quad S = \theta}{S \xrightarrow{(\theta,\mathsf{unprotect})} \bullet}}{T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unprtect}),\phi,(\theta,\mathsf{unprtect})\rangle} T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma}$$

  We know that $\{\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{unprotected}\ e])\rangle, \langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{unprotected}\ e]$.

  Then after the transitions we have that $\{\langle \sigma_A, E.(\theta, \mathcal{P}[\mathsf{unprotected}\ e]), \mathsf{unit}\rangle, \langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

  This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T), the thread t correspond to $\mathcal{P}[\mathsf{unprotected}\ e]$ (because $t.\mathsf{exp} = \mathcal{P}[\mathsf{unprotected}\ e]$) and S is equal to $\bullet$ in fact the last field in Abadi.system is equal to unit.

- if in Abadi.system we have this transition:

  Trans Close
  $$\langle \sigma_A, E.(\theta, \mathcal{E}[\mathsf{unprotected}\ V]).E', (\theta', e)\rangle \xmapsto{(\theta,\mathsf{unp}(V))} \langle \sigma_A, E.(\theta, \mathcal{E}[V]).E', (\theta', e)\rangle$$

  in our system we have:

$$\frac{\textsc{Close}\quad t.\mathsf{id}=\theta \qquad t.\mathsf{exp}=\mathcal{E}[\mathsf{unprotected}\ V] \qquad \textsc{Step U/Close}\quad S\neq\theta}{T \mid t \xrightarrow{(\theta,\mathsf{unp}(V))} T \mid t[exp \leftarrow\!\!\shortmid \mathcal{E}[V]] \qquad S \xrightarrow{(\theta,\mathsf{unp}(V))} S}$$
$$T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(V)),\phi,(\theta,\mathsf{unp}(V))\rangle} T \mid t[exp \leftarrow\!\!\shortmid \mathcal{E}[V]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma$$

We know that $\{\langle\sigma_A, E.(\theta,\mathcal{E}[\mathsf{unprotected}\ V]).E', (\theta', e)\rangle, \langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id}=\theta$ and $t.\mathsf{exp}=\mathcal{E}[\mathsf{unprotected}\ V]$.

Then after the transitions we have that $\{\langle\sigma_A, E.(\theta,\mathcal{E}[V]).E', (\theta', e)\rangle, \langle T \mid t[exp \leftarrow\!\!\shortmid \mathcal{E}[V]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{E}[V]$ (because $t.\mathsf{exp}=\mathcal{E}[V]$) and S correspond to e in fact the two rules do not change them.

To proof the bi-simulation we can prove that given Rel as a strong simulation relation then if also Rel^{-1} is a strong simulation relation then Rel is a bi-simulation relation. Now we show the relationship Rel^{-1} derived directly from Rel.

Given $\{\langle T \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle, \langle\sigma_A, E, e\rangle\}$ then this tuple is in Rel^{-1} if:

- $\sigma_A = \sigma$
- $\forall t_i \in T$ such that $t_i.\mathsf{id} = \theta_i$ and $\theta_i \neq S$ then exist $e_i = t_i.\mathsf{exp}$ and $e_i \in E$
- if $S = \bullet$ then $e = \mathsf{unit}$ otherwise (if $S \neq \bullet$) then exist $t \in T$ such that $t.\mathsf{id} = S$ and we have that $e = t.\mathsf{exp}$
- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{no\ unit,act}),\phi,(\theta,\mathsf{no\ unit,act})\rangle} T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \qquad \text{if } e \neq \mathsf{unit}$$

in Abadi.system we have:

$$\textsc{Trans Activate}$$
$$\langle\sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{no\ unit,act})} \langle\sigma_A, E.E', (\theta, e)\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle\sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$.

Then after the transitions we have that $\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \{\langle\sigma_A, E.E', (\theta, e)\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the

two systems changed either E and E' or T) and e correspond to the thread t (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$).

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{act}),\phi,(\theta,\mathsf{unit},\mathsf{act})\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma \qquad \text{if } e = \mathsf{unit}$$

in Abadi.system we have:

$$\textsc{Trans Activate}$$
$$\langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{act})} \langle \sigma, E.E', \mathsf{unit}\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta, e).E', \mathsf{unit}\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$.

Then after the transitions we have that $\langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \{\langle \sigma, E.E', \mathsf{unit}\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\text{no unit},\mathsf{pure}),\phi,(\theta,\text{no unit},\mathsf{pure})\rangle} T \mid t[\mathsf{exp} \leftarrow\!\shortmid \mathcal{P}[e[V/x]]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \quad \text{if } \mathcal{P}[e[V/x]] \neq \mathsf{unit}$$

in Abadi.system we have:

$$\textsc{Trans Appl P}$$
$$\langle \sigma_A, E, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle \xmapsto{(\theta,\text{no unit},\mathsf{pure})} \langle \sigma_A, E, (\theta, \mathcal{P}[e[V/x]])\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[(\lambda\ x.e)\ V]$.

Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \leftarrow\!\shortmid \mathcal{P}[e[V/x]]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[e[V/x]])\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $\mathcal{P}[e[V/x]]$ correspond to the thread t (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[e[V/x]]$).

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{pure}),\phi,(\theta,\mathsf{unit},\mathsf{pure})\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

$$\textsc{Trans Appl P}$$
$$\langle \sigma_A, E, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{pure})} \langle \sigma_A, E, \mathsf{unit}\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[(\lambda\ x.e)\ V])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[(\lambda\ x.e)\ V]$.

Then after the transitions we have that $\{\langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, \mathsf{unit}\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{pure})),\phi,(\theta,\mathsf{unp}(\mathsf{pure}))\rangle} T \mid t[exp \hookleftarrow \mathcal{U}[e[V/x]]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

$$\textsc{Trans Appl U}$$
$$\langle \sigma_A, E.(\theta,\mathcal{U}[(\lambda\ x.e)\ V]).E', (\theta', e')\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{pure}))} \langle \sigma_A, E.(\theta,\mathcal{U}[e[V/x]]).E', (\theta', e')\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta,\mathcal{U}[(\lambda\ x.e)\ V]).E', (\theta', e')\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[(\lambda\ x.e)\ V]$.

Then after the transitions we have that $\{\langle T \mid t[exp \hookleftarrow \mathcal{U}[e[V/x]]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta,\mathcal{U}[e[V/x]]).E', (\theta', e')\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[e[V/x]]$ (because $t.\mathsf{exp} = \mathcal{U}[e[V/x]]$) and e' correspond to S in fact the two rules do not change them.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v)),(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v)),(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v))\rangle} T \mid t[\mathsf{exp} \hookleftarrow \mathcal{P}[r]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma[r \mapsto v] \quad \text{if } \mathcal{P}[r] \neq \mathsf{unit}$$

in Abadi.system we have:

$$\textsc{Trans Ref P}$$
$$\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{ref}\ V])\rangle \xmapsto{(\theta,\mathsf{no\ unit},\mathsf{ref}(r,v))} \langle \sigma_A[r \mapsto V], E, (\theta, \mathcal{P}[r])\rangle$$
$$\text{if } r \in RefLoc\ -\ dom(\sigma)$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{ref}\ V])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{ref}\ V]$.

Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \leftarrow \mathcal{P}[r]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle, \langle \sigma_A[r \mapsto V], E, (\theta, \mathcal{P}[r])\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $\mathcal{P}[r]$ correspond to the thread t (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[r]$).

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{ref}(r,v)),(\theta,\mathsf{unit},\mathsf{ref}(r,v)),(\theta,\mathsf{unit},\mathsf{ref}(r,v))\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]$$

in Abadi.system we have:

Trans Ref P
$$\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{ref}\ V])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{ref}(r,v))} \langle \sigma_A[r \mapsto V], E, \mathsf{unit}\rangle \qquad \text{if } r \in RefLoc\ -\ dom(\sigma)$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{ref}\ V])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{ref}\ V]$.

Then after the transitions we have that $\{\langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle, \langle \sigma_A[r \mapsto V], E, \mathsf{unit}\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{ref}(r,v))),(\theta,\mathsf{unp}(\mathsf{ref}(r,v))),(\theta,\mathsf{unp}(\mathsf{ref}(r,v)))\rangle} T \mid t[exp \leftarrow \mathcal{U}[r]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma[r \mapsto v]$$

in Abadi.system we have:

Trans Ref U
$$\langle \sigma_A, E.(\theta,\mathcal{U}[\mathsf{ref}\ V]).E', (\theta', e)\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{ref}(r,v)))} \langle \sigma_A[r \mapsto V], E.(\theta,\mathcal{U}[r]).E', (\theta', e)\rangle \quad \text{if } r \in RefLoc\ -\ dom(\sigma)$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta,\mathcal{U}[\mathsf{ref}\ V]).E', (\theta', e)\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[\mathsf{ref}\ V]$.

Then after the transitions we have that $\{\langle T \mid t[exp \leftarrow \mathcal{U}[r]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle, \langle \sigma_A[r \mapsto V], E.(\theta,\mathcal{U}[r]).E', (\theta', e)\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to

the two systems changed either E and E' or T), $\mathcal{U}[r]$ correspond to the thread t (because $t.\mathsf{exp} = \mathcal{U}[r]$) and e correspond to S in fact the two rules do not change them.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{no\ unit},\mathsf{rd}(r,v)),(\theta,\mathsf{no\ unit},\mathsf{rd}(r,v)),(\theta,\mathsf{no\ unit},\mathsf{rd}(r,v))\rangle} T \mid t[\mathsf{exp} \leftarrowtail \mathcal{P}[v]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

$$\begin{array}{l}\textsc{Trans Deref P}\\ \langle \sigma_A, E, (\theta, \mathcal{P}[!r])\rangle \xmapsto{(\theta,\mathsf{no\ unit},\mathsf{rd}(r,v))} \langle \sigma_A, E, (\theta, \mathcal{P}[V])\rangle \qquad \mathsf{if}\ \sigma(r) = V\end{array}$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[!r])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[!r]$.

Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \leftarrowtail \mathcal{P}[v]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[V])\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $\mathcal{P}[v]$ (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[v]$).

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{rd}(r,v)),(\theta,\mathsf{unit},\mathsf{rd}(r,v)),(\theta,\mathsf{unit},\mathsf{rd}(r,v))\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

$$\begin{array}{l}\textsc{Trans Deref P}\\ \langle \sigma_A, E, (\theta, \mathcal{P}[!r])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{rd}(r,v))} \langle \sigma_A, E, \mathsf{unit}\rangle \qquad \mathsf{if}\ \sigma(r) = V\end{array}$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[!r])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[!r]$.

Then after the transitions we have that $\{\langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, \mathsf{unit}\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{rd}(r,v))),(\theta,\mathsf{unp}(\mathsf{rd}(r,v))),(\theta,\mathsf{unp}(\mathsf{rd}(r,v)))\rangle} T \mid t[\mathsf{exp} \leftarrowtail \mathcal{U}[v]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

$$\textsc{Trans Deref U}$$
$$\langle \sigma_A, E.(\theta,\mathcal{U}[!r]).E', (\theta', e)\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{rd}(r,v)))} \langle \sigma_A, E.(\theta,\mathcal{U}[V]).E', (\theta', e)\rangle \qquad \text{if } \sigma(r) = V$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta,\mathcal{U}[!r]).E', (\theta', e)\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[!r]$.

Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \hookleftarrow \mathcal{U}[v]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta,\mathcal{U}[V]).E', (\theta', e)\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), $\mathcal{U}[v]$ correspond to the thread t (because $t.\mathsf{exp} = \mathcal{U}[v]$) and e correspond to S in fact the two rules do not change them.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{no\ unit},\mathsf{wr}(r,v)),(\theta,\mathsf{no\ unit},\mathsf{wr}(r,v)),(\theta,\mathsf{no\ unit},\mathsf{wr}(r,v))\rangle} T \mid t[\mathsf{exp} \hookleftarrow \mathcal{P}[\mathsf{unit}]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma[r \mapsto v]$$

in Abadi.system we have:

$$\textsc{Trans Set P}$$
$$\langle \sigma_A, E, (\theta, \mathcal{P}[r := V])\rangle \xmapsto{(\theta,\mathsf{no\ unit},\mathsf{wr}(r,v))} \langle \sigma_A[r \mapsto V], E, (\theta, \mathcal{P}[\mathsf{unit}])\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[r := V])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[r := v]$.

Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \hookleftarrow \mathcal{P}[\mathsf{unit}]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle, \langle \sigma_A[r \mapsto V], E, (\theta, \mathcal{P}[\mathsf{unit}])\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $\mathcal{P}[\mathsf{unit}]$ (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$).

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{wr}(r,v)),(\theta,\mathsf{unit},\mathsf{wr}(r,v)),(\theta,\mathsf{unit},\mathsf{wr}(r,v))\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]$$

in Abadi.system we have:

$$\textsc{Trans Set P}$$
$$\langle \sigma_A, E, (\theta, \mathcal{P}[r := V])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{wr}(r,v))} \langle \sigma_A[r \mapsto V], E, \mathsf{unit}\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[r := V])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[r := v]$.

Then after the transitions we have that $\{\langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle, \langle \sigma_A[r \mapsto v], E, \mathsf{unit}\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{wr}(r,v))),(\theta,\mathsf{unp}(\mathsf{wr}(r,v))),(\theta,\mathsf{unp}(\mathsf{wr}(r,v)))\rangle} T \mid t[\mathsf{exp} \leftarrow \mathcal{U}[\mathsf{unit}]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma[r \mapsto v]$$

in Abadi.system we have:

Trans Set U
$$\langle \sigma_A, E.(\theta,\mathcal{U}[r := V]).E', (\theta', e)\rangle \xrightarrow{(\theta,\mathsf{unp}(\mathsf{wr}(r,v)))} \langle \sigma_A[r \mapsto V], E.(\theta,\mathcal{U}[\mathsf{unit}]).E', (\theta', e)\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta,\mathcal{U}[r := V]).E', (\theta', e)\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[r := v]$.

Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \leftarrow \mathcal{U}[\mathsf{unit}]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle, \langle \sigma_A[r \mapsto V], E.(\theta,\mathcal{U}[\mathsf{unit}]).E', (\theta', e)\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because the rules applied to the two systems changed the σ_A and σ in the same way. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), $\mathcal{U}[\mathsf{unit}]$ correspond to the thread t (because $t.\mathsf{exp} = \mathcal{U}[\mathsf{unit}]$) and e correspond to S in fact the two rules do not change them.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{no\ unit},\mathsf{async}(e')),\phi,(\theta,\mathsf{no\ unit},\mathsf{async}(e'))\rangle} T \mid t[exp \leftarrow \mathcal{P}[\mathsf{unit}]] \mid t'[id \leftarrow \theta', exp \leftarrow e'] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

Trans Async P
$$\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{async}\ e'])\rangle \xrightarrow{(\theta,\mathsf{no\ unit},\mathsf{async}(e'))} \langle \sigma_A, E.(\theta', e'), (\theta, \mathcal{P}[\mathsf{unit}])\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{async}\ e'])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{async}\ e']$.

Then after the transitions we have that $\{\langle T \mid t[exp \leftarrow \mathcal{P}[\mathsf{unit}]] \mid t'[id \leftarrow \theta', exp \leftarrow e'] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta', e'), (\theta, \mathcal{P}[\mathsf{unit}])\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T), e' correspond to t' (because $t'.\mathsf{exp} = e'$ and both have

been added with the last transition) and $\mathcal{P}[\mathsf{unit}]$ correspond to the thread t (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$).

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unit},\mathsf{async}(e')),\phi,(\theta,\mathsf{unit},\mathsf{async}(e'))\rangle} T \mid t'[id \hookleftarrow \theta', exp \hookleftarrow e'] \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

$$\textsc{Trans Async P}$$
$$\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{async}\ e'])\rangle \xmapsto{(\theta,\mathsf{unit},\mathsf{async}(e'))} \langle \sigma_A, E.(\theta', e'), \mathsf{unit}\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{async}\ e'])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{async}\ e']$.

Then after the transitions we have that $\{\langle T \mid t[exp \hookleftarrow \mathcal{P}[\mathsf{unit}]] \mid t'[id \hookleftarrow \theta', exp \hookleftarrow e'] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta', e'), \mathsf{unit}\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T), e' correspond to t' (because $t'.\mathsf{exp} = e'$ and both have been added with the last transition) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{async}(e'))),\phi,(\theta,\mathsf{unp}(\mathsf{async}(e')))\rangle} T \mid t[exp \hookleftarrow \mathcal{U}[\mathsf{unit}]] \mid t'[id \hookleftarrow \theta', exp \hookleftarrow e'] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

$$\textsc{Trans Async U}$$
$$\langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{async}\ e']).E', (\theta'', e'')\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{async}(e')))} \langle \sigma_A, E.(\theta', e').(\theta, \mathcal{U}[\mathsf{unit}]).E', (\theta'', e'')\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{async}\ e']).E', (\theta'', e'')\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[\mathsf{async}\ e]$.

Then after the transitions we have that $\{\langle T \mid t[exp \hookleftarrow \mathcal{U}[\mathsf{unit}]] \mid t'[id \hookleftarrow \theta', exp \hookleftarrow e'] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta', e').(\theta, \mathcal{U}[\mathsf{unit}]).E', (\theta'', e'')\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), $\mathcal{U}[\mathsf{unit}]$ correspond to the thread t (because $t.\mathsf{exp} = \mathcal{U}[\mathsf{unit}]$), e' correspond to t' (because $t'.\mathsf{exp} = e'$ and both have been added with the last transition) and e correspond to S in fact the two rules do not change them.

- if in our system we have this transition:

  $$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\text{no unit},\text{blockUntil(true)}),\phi,(\theta,\text{no unit},\text{blockUntil(true)})\rangle} T \mid t[\mathsf{exp} \hookleftarrow \mathcal{P}[\mathsf{unit}]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma$$

  in Abadi.system we have:

  Trans Block P
  $$\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{blockUntil\ true}])\rangle \xmapsto{(\theta,\text{no unit},\text{blockUntil(true)})} \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{unit}])\rangle$$

  We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{blockUntil\ true}])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{blockUntil\ true}]$.

  Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \hookleftarrow \mathcal{P}[\mathsf{unit}]] \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{unit}])\rangle\} \in Rel^{-1}$.

  This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $\mathcal{P}[\mathsf{unit}]$ correspond to the thread t (because $S = \theta$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$).

- if in our system we have this transition:

  $$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\text{unit},\text{blockUntil(true)}),\phi,(\theta,\text{unit},\text{blockUntil(true)})\rangle} T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma$$

  in Abadi.system we have:

  Trans Block P
  $$\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{blockUntil\ true}])\rangle \xmapsto{(\theta,\text{unit},\text{blockUntil(true)})} \langle \sigma_A, E, \mathsf{unit}\rangle$$

  We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{blockUntil\ true}])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{blockUntil\ true}]$.

  Then after the transitions we have that $\{\langle T \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, \mathsf{unit}\rangle\} \in Rel^{-1}$.

  This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

  $$T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\text{unp(blockUntil true)}),\phi,(\theta,\text{unp(blockUntil true)})\rangle} T \mid t[exp \hookleftarrow \mathcal{U}[\mathsf{unit}] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma$$

  in Abadi.system we have:

  Trans Block U
  $$\langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{blockUntil\ true}]).E', (\theta', e)\rangle \xmapsto{(\theta,\text{unp(blockUntil(true)))}} \langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{unit}]).E', (\theta', e)\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{blockUntil\ true}]).E', (\theta', e)\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{U}[\mathsf{blockUntil\ true}]$.

Then after the transitions we have that $\{\langle T \mid t[exp \hookleftarrow \mathcal{U}[\mathsf{unit}] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{unit}]).E', (\theta', e)\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), $\mathcal{U}[\mathsf{unit}]$ correspond to the thread t (because $t.\mathsf{exp} = \mathcal{U}[\mathsf{unit}]$) and e correspond to S in fact the two rules do not change them.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unprtect}),\phi,(\theta,\mathsf{unprtect})\rangle} T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

Trans Unprotected
$$\langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{unprotected}\ e])\rangle \xmapsto{(\theta,\mathsf{unprotect})} \langle \sigma_A, E.(\theta, \mathcal{P}[\mathsf{unprotected}\ e]), \mathsf{unit}\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} \theta \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, (\theta, \mathcal{P}[\mathsf{unprotected}\ e])\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{unprotected}\ e]$.

Then after the transitions we have that $\{\langle T \mid t \otimes_{\mathcal{R}} \bullet \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta, \mathcal{P}[\mathsf{unprotected}\ e]), \mathsf{unit}\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T), $\mathcal{P}[\mathsf{unprotected}\ e]$ correspond to the thread t (because $t.\mathsf{exp} = \mathcal{P}[\mathsf{unprotected}\ e]$) and $S = \bullet$ in fact the last field of the Abadi system is equal to unit.

- if in our system we have this transition:

$$T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(V)),\phi,(\theta,\mathsf{unp}(V))\rangle} T \mid t[exp \hookleftarrow \mathcal{E}[V]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have:

Trans Close
$$\langle \sigma_A, E.(\theta, \mathcal{E}[\mathsf{unprotected}\ V]).E', (\theta', e)\rangle \xmapsto{(\theta,\mathsf{unp}(V))} \langle \sigma_A, E.(\theta, \mathcal{E}[V]).E', (\theta', e)\rangle$$

We know that $\{\langle T \mid t \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta, \mathcal{E}[\mathsf{unprotected}\ V]).E', (\theta', e)\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{E}[\mathsf{unprotected}\ V]$.

Then after the transitions we have that $\{\langle T \mid t[exp \hookleftarrow \mathcal{E}[V]] \otimes_{\mathcal{R}} S \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta, \mathcal{E}[V]).E', (\theta', e)\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), $\mathcal{E}[V]$ correspond to the thread t (because $t.\mathsf{exp} = \mathcal{E}[V]$) and e correspond to S in fact the two rules do not change them.

□

B.3.3 Weak Semantics with Optimistic Rollback

In Fig. B.7 we show the original weak semantics with optimistic rollback for AME described in [16].

For a deeper discussion about the semantics we refer to [16]. While in Fig. B.8 we refer to the updated semantics decorated with labels and with unique identifier for expressions.

Theorem 12. $\langle \sigma_A, E, O, l\rangle \sim AT \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma$

$$\begin{array}{ccc} \langle \sigma_A, E, O, l\rangle & \underset{Rel^{-1}}{\overset{Rel}{\longleftrightarrow}} & (AT \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma) \\ \Big\downarrow{\scriptstyle \varphi(\alpha)} & & \Big\downarrow{\scriptstyle \alpha} \\ \langle \sigma'_A, E', O', l'\rangle & \underset{Rel^{-1}}{\overset{Rel}{\longleftrightarrow}} & (AT' \otimes_{\mathcal{R}} \{S'; A'; L'\} \otimes_{\mathcal{R}} \sigma') \end{array}$$

Proof. We start by defining Rel and then show that it is a strong similarity relation, after which we will show that Rel^{-1} is also a strong similarity relation.

Given $\{\langle \sigma_A, E, O, l\rangle, \langle T \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\}$ then this tuple is in Rel if:

- $\sigma = \sigma_A$
- $\forall e_i \in E$ exist θ_i such that $t_i \in T$ and $t_i.\mathsf{id} = \theta_i \wedge t_i.\mathsf{exp} = e_i$
- $\forall o = (e, f, a, P_o) \in O$ we have that exist θ_o such that:
  - exist $t_o \in T$ and $t_o.\mathsf{id} = \theta_o$, $t_i.\mathsf{exp} = e$ and $t_o.\mathsf{init} = f$
  - $\theta_o \in S$
  - $\forall r \in a$ then $(\theta_o, r) \in A$
  - $\forall p \in P_o$ then exist θ_p such that exist $\lfloor t[\mathsf{id} \leftarrow\!\shortmid \theta_p, \mathsf{exp} \leftarrow\!\shortmid p]\rfloor_{\theta_o} \in P$
- $L = l$

After the definition of Rel we start to give for each label the proof that there is a strong similarity between Abadi system and our system:

- if in Abadi.system we have this transition:

$$\textsc{Trans Activate}$$
$$\langle \sigma_A, E.(\theta, e).E', O, l\rangle \xrightarrow{(\theta, \mathsf{act})} \langle \sigma_A, E.E', ((\theta, e), e, \emptyset, \emptyset).O, l\rangle$$

Trans Appl P
$\langle \sigma, T, O.(\mathcal{P}[(\lambda\ x.e)\ V], f, a, P).O', l\rangle \longmapsto \langle \sigma, T, O.(\mathcal{P}[e[V/x]], f, a, P).O', l\rangle$

Trans Appl U
$\langle \sigma, T.\mathcal{U}[(\lambda\ x.e)\ V].T', O, l\rangle \longmapsto \langle \sigma, T.\mathcal{U}[e[V/x]].T', O, l\rangle$

Trans Ref P
$\langle \sigma, T, O.(\mathcal{P}[\mathsf{ref}\ V], f, a, P).O', l\rangle \longmapsto \langle \sigma[r \mapsto V], T, O.(\mathcal{P}[r], f, a, P).O', l\rangle$
if $r \in RefLoc\ -\ dom(\sigma)$

Trans Ref U
$\langle \sigma, T.\mathcal{U}[\mathsf{ref}\ V].T', O, l\rangle \longmapsto \langle \sigma[r \mapsto V], T.\mathcal{U}[r].T', O, l\rangle$ if $r \in RefLoc\ -\ dom(\sigma)$

Trans Deref P
$\langle \sigma, T, O.(\mathcal{P}[!r], f, a, P).O', l\rangle \longmapsto \langle \sigma, T, O.(\mathcal{P}[V], f, r.a, P).O', l\rangle$ if $\sigma(r) = V$

Trans Deref U
$\langle \sigma, T.\mathcal{U}[!r].T', O, l\rangle \longmapsto \langle \sigma, T.\mathcal{U}[V].T', O, l\rangle$ if $\sigma(r) = V$

Trans Set P
$\langle \sigma, T, O.(\mathcal{P}[r := V], f, a, P).O', l\rangle \longmapsto \langle \sigma[r \mapsto V], T, O.(\mathcal{P}[\mathsf{unit}], f, r.a, P).O', l'\rangle$
where $l' = \mathsf{if}\ r \in dom(l)\ \mathsf{then}\ l\ \mathsf{else}\ l.[r \mapsto V]$

Trans Set U
$\langle \sigma, T.\mathcal{U}[r := V].T', O, l\rangle \longmapsto \langle \sigma[r \mapsto V], T.\mathcal{U}[\mathsf{unit}].T', O, l\rangle$

Trans Async P
$\langle \sigma, T, O.(\mathcal{P}[\mathsf{async}\ e], f, a, P).O', l\rangle \longmapsto \langle \sigma, T, O.(\mathcal{P}[\mathsf{unit}], f, a, e.P).O', l\rangle$

Trans Async U
$\langle \sigma, T.\mathcal{U}[\mathsf{async}\ e].T', O, l\rangle \longmapsto \langle \sigma, T.e.\mathcal{U}[\mathsf{unit}].T', O, l\rangle$

Trans Block P
$\langle \sigma, T, O.(\mathcal{P}[\mathsf{blockUntil\ true}], f, a, P).O', l\rangle \longmapsto \langle \sigma, T, O.(\mathcal{P}[\mathsf{unit}], f, a, P).O', l\rangle$

Trans Block U
$\langle \sigma, T.\mathcal{U}[\mathsf{blockUntil\ true}].T', O, l\rangle \longmapsto \langle \sigma, T.\mathcal{U}[\mathsf{unit}].T', O, l\rangle$

Trans Undo
$\langle \sigma, T, O, l\rangle \longmapsto \langle \sigma l, origin(O).T, \emptyset, \emptyset\rangle$

Trans Unprotected
$\langle \sigma, T, O.(\mathcal{P}[\mathsf{unprotected}\ e], f, a, P).O', l\rangle \longmapsto \langle \sigma, T.\mathcal{P}[\mathsf{unprotected}\ e].P, O.O', l - a\rangle$
if $(\mathcal{P}[\mathsf{unprotected}\ e], f, a, P)$ does not conflict with $O.O'$

Trans Done
$\langle \sigma, T, O.(\mathsf{unit}, f, a, P).O', l\rangle \longmapsto \langle \sigma, T.P, O.O', l - a\rangle$
if (unit, f, a, P) does not conflict with $O.O'$

Trans Close
$\langle \sigma, T.\mathcal{E}[\mathsf{unprotected}\ V].T', O, l\rangle \longmapsto \langle \sigma, T.\mathcal{E}[V].T', O, l\rangle$

Trans Activate
$\langle \sigma, T.e.T', O, l\rangle \longmapsto \langle \sigma, T.T', (e, e, \emptyset, \emptyset).O, l\rangle$

Figure B.7: Optimistic Weak Semantics Abadi [16]

Trans Appl P

$$\langle \sigma, T, O.((\theta, \mathcal{P}[(\lambda\ x.e)\ V]), f, a, P).O', l\rangle \xmapsto{(\theta,\mathsf{pure})} \langle \sigma, T, O.((\theta, \mathcal{P}[e[V/x]]), f, a, P).O', l\rangle$$

Trans Appl U

$$\langle \sigma, T.(\theta,\mathcal{U}[(\lambda\ x.e)\ V]).T', O, l\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{pure}))} \langle \sigma, T.(\theta,\mathcal{U}[e[V/x]]).T', O, l\rangle$$

Trans Ref P

$$\langle \sigma, T, O.((\theta, \mathcal{P}[\mathsf{ref}\ V]), f, a, P).O', l\rangle \xmapsto{(\theta,\mathsf{ref}(r,v))} \langle \sigma[r \mapsto V], T, O.((\theta, \mathcal{P}[r]), f, a, P).O', l\rangle$$
$$\text{if } r \in RefLoc\ -\ dom(\sigma)$$

Trans Ref U

$$\langle \sigma, T.(\theta,\mathcal{U}[\mathsf{ref}\ V]).T', O, l\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{ref}(r,v)))} \langle \sigma[r \mapsto V], T.(\theta,\mathcal{U}[r]).T', O, l\rangle$$
$$\text{if } r \in RefLoc\ -\ dom(\sigma)$$

Trans Deref P

$$\langle \sigma, T, O.((\theta, \mathcal{P}[!r]), f, a, P).O', l\rangle \xmapsto{(\theta,\mathsf{rd}(r,v))} \langle \sigma, T, O.((\theta, \mathcal{P}[V]), f, r.a, P).O', l\rangle \qquad \text{if } \sigma(r) = V$$

Trans Deref U

$$\langle \sigma, T.(\theta,\mathcal{U}[!r]).T', O, l\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{rd}(r,v)))} \langle \sigma, T.(\theta,\mathcal{U}[V]).T', O, l\rangle \qquad \text{if } \sigma(r) = V$$

Trans Set P

$$\langle \sigma, T, O.((\theta, \mathcal{P}[r := V]), f, a, P).O', l\rangle \xmapsto{(\theta,\mathsf{wr}(r,v))} \langle \sigma[r \mapsto V], T, O.((\theta, \mathcal{P}[\mathsf{unit}]), f, r.a, P).O', l'\rangle$$
$$\text{where } l' = \text{if } r \in dom(l) \text{ then } l \text{ else } l.[r \mapsto V]$$

Trans Set U

$$\langle \sigma, T.(\theta,\mathcal{U}[r := V]).T', O, l\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{wr}(r,v)))} \langle \sigma[r \mapsto V], T.(\theta,\mathcal{U}[\mathsf{unit}]).T', O, l\rangle$$

Trans Async P

$$\langle \sigma, T, O.((\theta, \mathcal{P}[\mathsf{async}\ e]), f, a, P).O', l\rangle \xmapsto{(\theta,\mathsf{async}(e))} \langle \sigma, T, O.((\theta, \mathcal{P}[\mathsf{unit}]), f, a, (\theta', e).P).O', l\rangle$$

Trans Async U

$$\langle \sigma, T.(\theta,\mathcal{U}[\mathsf{async}\ e]).T', O, l\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{async}(e)))} \langle \sigma, T.(\theta', e).(\theta,\mathcal{U}[\mathsf{unit}]).T', O, l\rangle$$

Trans Block P

$$\langle \sigma, T, O.((\theta, \mathcal{P}[\mathsf{blockUntil\ true}]), f, a, P).O', l\rangle \xmapsto{(\theta,\mathsf{blockUntil}(\mathsf{true}))} \langle \sigma, T, O.((\theta, \mathcal{P}[\mathsf{unit}]), f, a, P).O', l\rangle$$

Trans Block U

$$\langle \sigma, T.(\theta,\mathcal{U}[\mathsf{blockUntil\ true}]).T', O, l\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{blockUntil}(\mathsf{true})))} \langle \sigma, T.(\theta,\mathcal{U}[\mathsf{unit}]).T', O, l\rangle$$

Trans Undo

$$\langle \sigma, T, O, l\rangle \xmapsto{(\theta_1,\ldots,\theta_n,\mathsf{undo}(l))} \langle \sigma l, origin(O).T, \emptyset, \emptyset\rangle \qquad \text{where } \theta_1, \ldots, \theta_n \text{ are the } \theta_i \text{ in } O$$

Trans Unprotected

$$\langle \sigma, T, O.((\theta, \mathcal{P}[\mathsf{unprotected}\ e]), f, a, P).O', l\rangle \xmapsto{(\theta,\mathsf{unprotect})} \langle \sigma, T.(\theta, \mathcal{P}[\mathsf{unprotected}\ e]).P, O.O', l - a\rangle$$
$$\text{if } (\mathcal{P}[\mathsf{unprotected}\ e], f, a, P) \text{ does not conflict with } O.O'$$

Trans Done

$$\langle \sigma, T, O.((\theta, \mathsf{unit}), f, a, P).O', l\rangle \xmapsto{\mathsf{done}} \langle \sigma, T.P, O.O', l - a\rangle$$
$$\text{if } (\mathsf{unit}, f, a, P) \text{ does not conflict with } O.O'$$

Trans Close

$$\langle \sigma, T.(\theta, \mathcal{E}[\mathsf{unprotected}\ V]).T', O, l\rangle \xmapsto{(\theta,\mathsf{unp}(V))} \langle \sigma, T.(\theta, \mathcal{E}[V]).T', O, l\rangle$$

Trans Activate

$$\langle \sigma, T.(\theta, e).T', O, l\rangle \xmapsto{(\theta,\mathsf{act})} \langle \sigma, T.T', ((\theta, e), e, \emptyset, \emptyset).O, l\rangle$$

in our system we have:

$$\frac{\textsc{Thread Activate}\quad \dfrac{t.\mathsf{id} = \theta}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{act})} T \mid t[\mathsf{init} \leftarrow t.\mathsf{exp}] \mid P} \qquad \textsc{Activate}\quad \dfrac{\theta \notin S}{\{S; A; L\} \xrightarrow{(\theta,\mathsf{act})} \{S \cup \{\theta\}; A; L\}}}{T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta,\mathsf{act})} T \mid t[\mathsf{init} \leftarrow t.\mathsf{exp}] \mid P \otimes_{\mathcal{R}} \{S \cup \{\theta\}; A; L\} \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle \sigma_A, E.(\theta, e).E', O, l\rangle, \langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$.

Then after the transitions we have that $\{\langle \sigma_A, E.E', ((\theta, e), e, \emptyset, \emptyset).O, l\rangle$,
$\langle T \mid t[\mathsf{init} \leftarrow t.\mathsf{exp}] \mid P \otimes_{\mathcal{R}} \{S \cup \{\theta\}; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ and $L = l$ because none of the rules applied to the two systems changed the memory and the logs.

We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T) and the thread t correspond to $(e, e, \emptyset, \emptyset)$ (because $\theta \in S$, $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = e$ and $t.\mathsf{init} = e$). A and P do not change in fact in $(e, f, a, P_o).O$ we have a and P_o equal to $\emptyset$.

- if in Abadi.system we have this transition:

  Trans Appl P
  $\langle \sigma_A, E, O.((\theta, \mathcal{P}[(\lambda\ x.e)\ V]), f, a, P_o).O', l\rangle \xmapsto{(\theta,\mathsf{pure})} \langle \sigma_A, E, O.((\theta, \mathcal{P}[e[V/x]]), f, a, P_o).O', l\rangle$

  in our system we have:

$$\frac{\textsc{Thread Step}\quad \dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[(\lambda\ x.e)\ V] \qquad (\lambda\ x.e)\ V \xrightarrow{\mathsf{pure}} e[V/x]}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{pure})} T \mid t[exp \leftarrow \mathcal{P}[e[V/x]]] \mid P} \qquad \textsc{Step}\quad \dfrac{\theta \in S}{\{S; A; L\} \xrightarrow{(\theta,\mathsf{pure})} \{S; A; L\}}}{T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta,\mathsf{pure})} T \mid t[\mathsf{exp} \leftarrow \mathcal{P}[e[V/x]]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma}$$

  We know that $\{\langle \sigma_A, E, O.((\theta, \mathcal{P}[(\lambda\ x.e)\ V]), f, a, P_o).O', l\rangle$,
  $\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{init} = f$ and $t.\mathsf{exp} = \mathcal{P}[(\lambda\ x.e)\ V]$.

  Then after the transitions we have that $\{\langle \sigma_A, E, O.((\theta, \mathcal{P}[e[V/x]]), f, a, P_o).O', l\rangle$,
  $\langle T \mid t[\mathsf{exp} \leftarrow \mathcal{P}[e[V/x]]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

  This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ and $L = l$ because none of the rules applied to the two systems changed the memory and the logs.

  We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $(\mathcal{P}[e[V/x]], f, a, P_o)$ (because $\theta \in S$, $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[e[V/x]]$ and $t.\mathsf{init} = f$). A and P do not change in fact the rule does not modify a and P_o.

- if in Abadi.system we have this transition:

  Trans Appl U
  $$\langle \sigma_A, E.(\theta, \mathcal{U}[(\lambda\ x.e)\ V]).E', O, l\rangle \xmapsto{(\theta,\mathsf{unp(pure)})} \langle \sigma_A, E.(\theta, \mathcal{U}[e[V/x]]).E', O, l\rangle$$

  in our system we have:

$$\dfrac{\dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{U}[(\lambda\ x.e)\ V] \qquad (\lambda\ x.e)\ V \xhookrightarrow{\mathsf{pure}} e[V/x]}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{unp(pure)})} T \mid t[exp \leftarrow\!\shortmid \mathcal{U}[e[V/x]]] \mid P} \qquad \dfrac{\theta \notin S}{\{S; A; L\} \xrightarrow{(\theta,\mathsf{unp(pure)})} \{S; A; L\}}}{T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta,\mathsf{unp(pure)})} T \mid t[exp \leftarrow\!\shortmid \mathcal{U}[e[V/x]]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma}$$

  Thread Step Unprotect

  U Step/Close

  We know that $\{\langle \sigma_A, E.(\theta, \mathcal{U}[(\lambda\ x.e)\ V]).E', O, l\rangle,\ \langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{U}[(\lambda\ x.e)\ V]$ and $\theta \notin S$.

  Then after the transitions we have that $\{\langle \sigma_A, E.(\theta, \mathcal{U}[e[V/x]]).E', O, l\rangle,\ \langle T \mid t[exp \leftarrow\!\shortmid \mathcal{U}[e[V/x]]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

  This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ and $L = l$ because none of the rules applied to the two systems changed the memory and the logs.

  We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[e[V/x]]$ (because $t.\mathsf{exp} = \mathcal{U}[e[V/x]]$)). We have that the element in O does not change in fact we have that all the tuple (e, f, a, P_o) in O have correspond to a thread in T, the ids of these threads are in S. Furthermore for all the tuple (e, f, a, P_o) in O each element in a and P_o has a correspondence with the elements in A and P.

- if in Abadi.system we have this transition:

  Trans Ref P
  $$\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{ref}\ V]), f, a, P_o).O', l\rangle \xmapsto{(\theta,\mathsf{ref}(r,v))} \langle \sigma_A[r \mapsto V], E, O.((\theta, \mathcal{P}[r]), f, a, P_o).O', l\rangle \quad \mathsf{if}\ r \in RefLoc\ -\ dom(\sigma)$$

  in our system we have:

$$\dfrac{\dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[\mathsf{ref}\ V] \qquad \mathsf{ref}\ V \xhookrightarrow{\mathsf{ref}(r,v)} r}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{ref}(r,v))} T \mid t[exp \leftarrow\!\shortmid \mathcal{P}[r]] \mid P} \qquad \dfrac{\dfrac{\theta \in S}{\{S; A; L\} \xrightarrow{(\theta,\mathsf{ref}(r,v))} \{S; A; L\}} \qquad \dfrac{r \in (RefLoc - \mathsf{dom}(\sigma))}{\sigma \xrightarrow{(\theta,\mathsf{ref}(r,v))} \sigma[r \mapsto v]}}{T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta,\mathsf{ref}(r,v))} T \mid t[\mathsf{exp} \leftarrow\!\shortmid \mathcal{P}[r]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma[r \mapsto v]}$$

  Thread Step

  Step

  Ref

  We know that $\{\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{ref}\ V]), f, a, P_o).O', l\rangle,$

$\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{init} = f$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{ref}\ V]$.

Then after the transitions we have that $\{\langle \sigma_A[r \mapsto V], E, O.((\theta, \mathcal{P}[r]), f, a, P_o).O', l\rangle$, $\langle T \mid t[\mathsf{exp} \leftarrow\!\shortmid \mathcal{P}[r]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because the rules add the same element to σ_A and to σ. $L = l$ because none of the rules applied to the two systems changed the logs.

We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $(\mathcal{P}[r], f, a, P_o)$ (because $\theta \in S$, $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[r]$ and $t.\mathsf{init} = f$). A and P do not change in fact the rule does not modify a and P_o.

- if in Abadi.system we have this transition:

$$\textsc{Trans Ref U}\quad \langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{ref}\ V]).E', O, l\rangle \xrightarrow{(\theta, \mathsf{unp}(\mathsf{ref}(r,v)))} \langle \sigma_A[r \mapsto V], E.(\theta, \mathcal{U}[r]).E', O, l\rangle \quad \text{if } r \in RefLoc\ -\ dom(\sigma)$$

in our system we have:

$$\textsc{Thread Step Unprotect}\quad \frac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{U}[\mathsf{ref}\ V] \quad \mathsf{ref}\ V \xrightarrow{\mathsf{ref}(r,v)} r}{T \mid t \mid P \xrightarrow{(\theta, \mathsf{unp}(\mathsf{ref}(r,v)))} T \mid t[exp \leftarrow\!\shortmid \mathcal{U}[r]] \mid P} \ \alpha \neq \mathsf{async}(e'')$$

$$\textsc{U Step/Close}\quad \frac{\theta \notin S}{\{S; A; L\} \xrightarrow{(\theta, \mathsf{unp}(\mathsf{ref}(r,v)))} \{S; A; L\}} \qquad \textsc{URef}\quad \frac{r \in (RefLoc - \mathsf{dom}(\sigma))}{\sigma \xrightarrow{(\theta, \mathsf{unp}(\mathsf{ref}(r,v)))} \sigma[r \mapsto v]}$$

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle (\theta, \mathsf{unp}(\mathsf{ref}(r,v))), (\theta, \mathsf{unp}(\mathsf{ref}(r,v))), (\theta, \mathsf{unp}(\mathsf{ref}(r,v)))\rangle} T \mid t[exp \leftarrow\!\shortmid \mathcal{U}[r]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma[r \mapsto v]$$

We know that $\{\langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{ref}\ V]).E', O, l\rangle$, $\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{U}[\mathsf{ref}\ V]$ and $\theta \notin S$.

Then after the transitions we have that $\{\langle \sigma_A[r \mapsto V], E.(\theta, \mathcal{U}[r]).E', O, l\rangle$, $\langle T \mid t[exp \leftarrow\!\shortmid \mathcal{U}[r]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because the rules add the same element to σ_A and to σ. $L = l$ because none of the rules applied to the two systems changed the logs.

We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[r]$ (because $t.\mathsf{exp} = \mathcal{U}[r]$)). We have that the element in O does not change in fact we have that all the tuple (e, f, a, P_o) in O have correspond to a thread in T, the ids of these threads are in S. Furthermore for all the tuple (e, f, a, P_o) in O each element in a and P_o has a correspondence with the elements in A and P.

- if in Abadi.system we have this transition:

  Trans Deref P
  $$\langle \sigma_A, E, O.((\theta, \mathcal{P}[!r]), f, a, P_o).O', l\rangle \xmapsto{(\theta,\mathsf{rd}(r,v))} \langle \sigma_A, E, O.((\theta, \mathcal{P}[V]), f, r.a, P_o).O', l\rangle \quad \text{if } \sigma(r) = V$$

  in our system we have:

$$\dfrac{\dfrac{\text{Thread Step}\quad t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[!r] \qquad !r \xrightarrow{\mathsf{rd}(r,v)} v}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{rd}(r,v))} T \mid t[exp \hookleftarrow \mathcal{P}[v]] \mid P} \quad \dfrac{\text{DeRef (Read)}\quad \theta \in S}{\{S; A; L\} \xrightarrow{(\theta,\mathsf{rd}(r,v))} S \mid A \cup \{\theta, r\} \mid L} \quad \dfrac{\text{DeRef (Read)}\quad \sigma(r) = v}{\sigma \xrightarrow{(\theta,\mathsf{rd}(r,v))} \sigma}}{T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{rd}(r,v)),(\theta,\mathsf{rd}(r,v)),(\theta,\mathsf{rd}(r,v))\rangle} T \mid t[\mathsf{exp} \hookleftarrow \mathcal{P}[v]] \mid P \otimes_{\mathcal{R}} \{S; A \cup \{\theta, r\}; L\} \otimes_{\mathcal{R}} \sigma}$$

  We know that $\{\langle \sigma_A, E, O.((\theta, \mathcal{P}[!r]), f, a, P_o).O', l\rangle,$
  $\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{init} = f$ and $t.\mathsf{exp} = \mathcal{P}[!r]$.

  Then after the transitions we have that $\{\langle \sigma_A, E, O.((\theta, \mathcal{P}[V]), f, r.a, P_o).O', l\rangle,$
  $\langle T \mid t[\mathsf{exp} \hookleftarrow \mathcal{P}[v]] \mid P \otimes_{\mathcal{R}} \{S; A \cup \{\theta, r\}; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

  This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ and $L = l$ because none of the rules applied to the two systems changed the memory and the logs.

  We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $(\mathcal{P}[V], f, r.a, P_o)$ (because $\theta \in S$, $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[V]$ and $t.\mathsf{init} = f$). All the other element in O and O' does not change, $(\mathcal{P}[V], f, r.a, P_o)$ add r to a in fact in our system the element $\{\theta, r\}$ is added to A. P does not change in fact the rule does not modify P_o.

- if in Abadi.system we have this transition:

  Trans Deref U
  $$\langle \sigma_A, E.(\theta, \mathcal{U}[!r]).E', O, l\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{rd}(r,v)))} \langle \sigma_A, E.(\theta, \mathcal{U}[V]).E', O, l\rangle \qquad \text{if } \sigma(r) = V$$

  in our system we have:

$$\frac{\dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{U}[!r] \quad !r \xrightarrow{\mathsf{rd}(r,v)} v}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{unp}(\mathsf{rd}(r,v)))} T \mid t[exp \leftarrowtail \mathcal{U}[v]] \mid P}\ \text{Thread Step Unprotect} \qquad \dfrac{\dfrac{\theta \notin S}{\{S;A;L\} \xrightarrow{(\theta,\mathsf{unp}(\mathsf{rd}(r,v)))} \{S;A;L\}}\ \text{U Step/Close} \qquad \dfrac{\sigma(r) = v}{\sigma \xrightarrow{(\theta,\mathsf{unp}(\mathsf{rd}(r,v)))} \sigma}\ \text{UDeRef (Read)}}{T \mid t \mid P \otimes_{\mathcal{R}} \{S;A;L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta,\mathsf{unp}(\mathsf{rd}(r,v)))} T \mid t[\mathsf{exp} \leftarrowtail \mathcal{U}[v]] \mid P \otimes_{\mathcal{R}} \{S;A;L\} \otimes_{\mathcal{R}} \sigma}$$

We know that $\{\langle \sigma_A, E.(\theta,\mathcal{U}[!r]).E', O, l\rangle,\ \langle T \mid t \mid P \otimes_{\mathcal{R}} \{S;A;L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{U}[!r]$ and $\theta \notin S$.

Then after the transitions we have that $\{\langle \sigma_A, E.(\theta,\mathcal{U}[V]).E', O, l\rangle,\ \langle T \mid t[\mathsf{exp} \leftarrowtail \mathcal{U}[v]] \mid P \otimes_{\mathcal{R}} \{S;A;L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ and $L = l$ because none of the rules applied to the two systems changed the memory and the logs.

We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[v]$ (because $t.\mathsf{exp} = \mathcal{U}[v]$)). We have that the element in O does not change in fact we have that all the tuple (e, f, a, P_o) in O have correspond to a thread in T, the ids of these threads are in S. Furthermore for all the tuple (e, f, a, P_o) in O each element in a and P_o has a correspondence with the elements in A and P.

- if in Abadi.system we have this transition:

$$\text{Trans Set P}\quad \langle \sigma_A, E, O.((\theta, \mathcal{P}[r := V]), f, a, P_o).O', l\rangle \xrightarrow{(\theta,\mathsf{wr}(r,v))} \langle \sigma_A[r \mapsto V], E, O.((\theta, \mathcal{P}[\mathsf{unit}]), f, r.a, P_o).O', l'\rangle \quad \text{where } l' = \mathsf{if}\ r \in dom(l)\ \mathsf{then}\ l\ \mathsf{else}\ l.[r \mapsto V]$$

in our system we have:

$$\frac{\dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[r := v] \quad r := v \xrightarrow{\mathsf{wr}(r,v)} \mathsf{unit}}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{wr}(r,v))} T \mid t[exp \leftarrowtail \mathcal{P}[\mathsf{unit}]] \mid P}\ \text{Thread Step} \qquad \dfrac{\dfrac{\theta \in S}{\{S;A;L\} \xrightarrow{(\theta,\mathsf{wr}(r,v))} \{S;A \cup \{\theta,r\};L.[r \mapsto \sigma(r)]\}}\ \text{Set (Write)} \qquad \dfrac{}{\sigma \xrightarrow{(\theta,\mathsf{wr}(r,v))} \sigma[r \mapsto v]}\ \text{Set (Write)}}{T \mid t \mid P \otimes_{\mathcal{R}} \{S;A;L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta,\mathsf{wr}(r,v))} T \mid t[\mathsf{exp} \leftarrowtail \mathcal{P}[\mathsf{unit}]] \mid P \otimes_{\mathcal{R}} \{S;A \cup \{\theta,r\};L.[r \mapsto \sigma(r)]\} \otimes_{\mathcal{R}} \sigma[r \mapsto v]}$$

We know that $\{\langle \sigma_A, E, O.((\theta, \mathcal{P}[r := V]), f, a, P_o).O', l\rangle,$
$\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S;A;L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{init} = f$ and $t.\mathsf{exp} = \mathcal{P}[r := V]$.

Then after the transitions we have that $\{\langle\sigma_A[r \mapsto V], E, O.((\theta, \mathcal{P}[\mathsf{unit}]), f, r.a, P_o).O', l'\rangle, \langle T \mid t[\mathsf{exp} \leftarrow\!\shortmid \mathcal{P}[\mathsf{unit}]] \mid P \otimes_{\mathcal{R}} \{S; A \cup \{\theta, r\}; L.[r \mapsto \sigma(r)]\} \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ and $L = l$ because the rules applied to the two systems changed the memory and the logs in the same way.

We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $(\mathcal{P}[\mathsf{unit}], f, a, P_o)$ (because $\theta \in S$, $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$ and $t.\mathsf{init} = f$). All the other element in O and O' does not change, $(\mathcal{P}[\mathsf{unit}], f, r.a, P_o)$ add r to a in fact in our system the element $\{\theta, r\}$ is added to A. P does not change in fact the rule does not modify P_o.

- if in Abadi.system we have this transition:

  Trans Set U

  $$\langle\sigma_A, E.(\theta, \mathcal{U}[r := V]).E', O, l\rangle \xmapsto{(\theta, \mathsf{unp}(\mathsf{wr}(r,v)))} \langle\sigma_A[r \mapsto V], E.(\theta, \mathcal{U}[\mathsf{unit}]).E', O, l\rangle$$

  in our system we have:

  $$\dfrac{\dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{U}[r := v] \quad r := v \xhookrightarrow{\mathsf{unp}(\mathsf{wr}(r,v))} \mathsf{unit}}{T \mid t \mid P \xrightarrow{(\theta, \mathsf{unp}(\mathsf{wr}(r,v)))} T \mid t[exp \leftarrow\!\shortmid \mathcal{U}[\mathsf{unit}]] \mid P} \text{ Thread Step Unprotect} \qquad \dfrac{\theta \notin S}{\{S; A; L\} \xrightarrow{(\theta, \mathsf{unp}(\mathsf{wr}(r,v)))} \{S; A; L\}} \text{ U Step/Close} \qquad \dfrac{}{\sigma \xrightarrow{(\theta, \mathsf{wr}(r,v))} \sigma[r \mapsto v]} \text{ Set (Write)}}{T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta, \mathsf{unp}(\mathsf{wr}(r,v))), (\theta, \mathsf{unp}(\mathsf{wr}(r,v))), (\theta, \mathsf{unp}(\mathsf{wr}(r,v)))\rangle} T \mid t[\mathsf{exp} \leftarrow\!\shortmid \mathcal{U}[\mathsf{unit}]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma[r \mapsto v]}$$

  We know that $\{\langle\sigma_A, E.(\theta, \mathcal{U}[r := V]).E', O, l\rangle, \langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{U}[r := V]$ and $\theta \notin S$.

  Then after the transitions we have that $\{\langle\sigma_A[r \mapsto V], E.(\theta, \mathcal{U}[\mathsf{unit}]).E', O, l\rangle, \langle T \mid t[\mathsf{exp} \leftarrow\!\shortmid \mathcal{U}[\mathsf{unit}]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle\} \in Rel$.

  This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because the rules add the same element to σ_A and to σ. $L = l$ because none of the rules applied to the two systems changed the logs.

  We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[]$ (because $t.\mathsf{exp} = \mathcal{U}[\mathsf{unit}]$). We have that the element in O does not change in fact we have that all the tuple (e, f, a, P_o) in O have correspond to a thread in T, the ids of these threads are in S. Furthermore for all the tuple (e, f, a, P_o) in O each element in a and P_o has a correspondence with the elements in A and P.

- if in Abadi.system we have this transition:

  Trans Async P
  $\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{async}\ e']), f, a, P_o).O', l\rangle \xmapsto{(\theta,\mathsf{async}(e'))} \langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{unit}]), f, a, (\theta', e').P_o).O', l\rangle$

  in our system we have:

$$\frac{\text{Thread Spawn}\quad t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[\mathsf{async}\ e'] \qquad t', \theta'\ fresh}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{async}(e'))} T \mid t[exp \hookleftarrow \mathcal{P}[\mathsf{unit}]] \mid P \mid \lfloor t'[id \hookleftarrow \theta', exp \hookleftarrow e']\rfloor_\theta} \quad \frac{\text{Step}\quad \theta \in S}{\{S; A; L\} \xrightarrow{(\theta,\mathsf{async}(e'))} \{S; A; L\}}$$

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta,\mathsf{async}(e'))} T \mid t[exp \hookleftarrow \mathcal{P}[\mathsf{unit}]] \mid P \mid \lfloor t'[id \hookleftarrow \theta', exp \hookleftarrow e']\rfloor_\theta \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma$$

  We know that $\{\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{async}\ e']), f, a, P_o).O', l\rangle,$
  $\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{init} = f$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{async}\ e']$.

  Then after the transitions we have that $\{\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{unit}]), f, a, (\theta', e').P_o).O', l\rangle,$
  $\langle T \mid t[exp \hookleftarrow \mathcal{P}[\mathsf{unit}]] \mid P \mid \lfloor t'[id \hookleftarrow \theta', exp \hookleftarrow e']\rfloor_\theta \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

  This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ and $L = l$ because none of the rules applied to the two systems changed the memory and the logs.

  We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $(\mathcal{P}[\mathsf{unit}], f, a, e'.P_o)$ (because $\theta \in S$, $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$ and $t.\mathsf{init} = f$). A does not change in fact the rule does not modify a. $\lfloor t'\rfloor_\theta$ correspond to e' in fact $t'.\mathsf{exp} = e'$, instead the other elements in P_o have a thread in P that correspond to them.

- if in Abadi.system we have this transition:

  Trans Async U
  $\langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{async}\ e']).E', O, l\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{async}(e')))} \langle \sigma_A, E.(\theta', e').(\theta, \mathcal{U}[\mathsf{unit}]).E', O, l\rangle$

  in our system we have:

$$\frac{\text{Thread Spawn Unprotect}\quad t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{U}[\mathsf{async}\ e'] \qquad t', \theta'\ fresh}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{unp}(\mathsf{async}(e')))} T \mid t[exp \hookleftarrow \mathcal{U}[\mathsf{unit}]] \mid t'[id \hookleftarrow \theta', exp \hookleftarrow e'] \mid P} \quad \frac{\text{U Step/Close}\quad \theta \notin S}{\{S; A; L\} \xrightarrow{(\theta,\mathsf{unp}(\mathsf{async}(e')))} \{S; A; L\}}$$

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta,\mathsf{unp}(\mathsf{async}(e')))} T \mid t[exp \hookleftarrow \mathcal{U}[\mathsf{unit}]] \mid t'[id \hookleftarrow \theta', exp \hookleftarrow e'] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma$$

  We know that $\{\langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{async}\ e']).E', O, l\rangle, \langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{U}[\mathsf{async}\ e']$ and $\theta \notin S$.

Then after the transitions we have that $\{\langle \sigma_A, E.(\theta', e').(\theta, \mathcal{U}[\mathsf{unit}]).E', O, l\rangle$, $\langle T \mid t[exp \leftarrow \mathcal{U}[\mathsf{unit}]] \mid t'[id \leftarrow \theta', exp \leftarrow e'] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ and $L = l$ because none of the rules applied to the two systems changed the memory and the logs.

We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[\mathsf{unit}]$ (because $t.\mathsf{exp} = \mathcal{U}[\mathsf{unit}]$)). e' correspond to t' in fact $t'.\mathsf{exp} = e'$. We have that the element in O does not change in fact we have that all the tuple (e, f, a, P_o) in O have correspond to a thread in T, the ids of these threads are in S. Furthermore for all the tuple (e, f, a, P_o) in O each element in a and P_o has a correspondence with the elements in A and P.

- if in Abadi.system we have this transition:

  Trans Block P

  $\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{blockUntil\ true}]), f, a, P_o).O', l\rangle \xmapsto{(\theta, \mathsf{blockUntil(true)})} \langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{unit}]), f, a, P_o).O', l\rangle$

  in our system we have:

$$\frac{\dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[\mathsf{blockUntil\ true}] \qquad \mathsf{blockUntil\ true} \xhookrightarrow{\mathsf{blockUntil(true)}} \mathsf{unit}}{T \mid t \mid P \xrightarrow{(\theta, \mathsf{blockUntil(true)})} T \mid t[exp \leftarrow \mathcal{P}[\mathsf{unit}]] \mid P} \text{ Thread Step} \qquad \frac{\theta \in S}{\{S; A; L\} \xrightarrow{(\theta, \mathsf{blockUntil(true)})} \{S; A; L\}} \text{ Step}}{T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta, \mathsf{blockUntil(true)})} T \mid t[exp \leftarrow \mathcal{P}[\mathsf{unit}]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma}$$

  We know that $\{\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{blockUntil\ true}]), f, a, P_o).O', l\rangle$,
  $\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{init} = f$ and mi $t.\mathsf{exp} = \mathcal{P}[\mathsf{blockUntil\ true}]$.

  Then after the transitions we have that $\{\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{unit}]), f, a, P_o).O', l\rangle$,
  $\langle T \mid t[exp \leftarrow \mathcal{P}[\mathsf{unit}]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

  This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ and $L = l$ because none of the rules applied to the two systems changed the memory and the logs.

  We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $(\mathcal{P}[\mathsf{unit}], f, a, P_o)$ (because $\theta \in S$, $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$ and $t.\mathsf{init} = f$). A and P do not change in fact the rule does not modify a and P_o.

- if in Abadi.system we have this transition:

  Trans Block U

  $\langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{blockUntil\ true}]).E', O, l\rangle \xmapsto{(\theta, \mathsf{unp(blockUntil(true))})} \langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{unit}]).E', O, l\rangle$

  in our system we have:

THREAD STEP UNPROTECT

$$\frac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{U}[\mathsf{blockUntil\ true}] \qquad \mathsf{blockUntil\ true} \xhookrightarrow{\mathsf{blockUntil(true)}} \mathsf{unit}}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{unp(blockUntil\ true)})} T \mid t[exp \leftarrow\!\shortmid \mathcal{U}[\mathsf{unit}]] \mid P}$$

U STEP/CLOSE

$$\frac{\theta \notin S}{\{S; A; L\} \xrightarrow{(\theta,\mathsf{unp(blockUntil\ true)})} \{S; A; L\}}$$

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp(blockUntil\ true)}),\phi,(\theta,\mathsf{unp(blockUntil\ true)})\rangle} T \mid t[exp \leftarrow\!\shortmid \mathcal{U}[\mathsf{unit}] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma$$

We know that $\{\langle \sigma_A, E.(\theta,\mathcal{U}[\mathsf{blockUntil\ true}]).E', O, l\rangle, \langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{U}[\mathcal{U}[\mathsf{blockUntil\ true}]]$ and $\theta \notin S$.

Then after the transitions we have that $\{\langle \sigma_A, E.(\theta,\mathcal{U}[\mathsf{unit}]).E', O, l\rangle, \langle T \mid t[exp \leftarrow\!\shortmid \mathcal{U}[\mathsf{unit}] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ and $L = l$ because none of the rules applied to the two systems changed the memory and the logs.

We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[\mathsf{unit}]$ (because $t.\mathsf{exp} = \mathcal{U}[\mathsf{unit}]$)). We have that the element in O does not change in fact we have that all the tuple (e, f, a, P_o) in O have correspond to a thread in T, the ids of these threads are in S. Furthermore for all the tuple (e, f, a, P_o) in O each element in a and P_o has a correspondence with the elements in A and P.

- if in Abadi.system we have this transition:

  TRANS UNDO

  $$\langle \sigma_A, E, O, l\rangle \xmapsto{(\theta_1,\ldots,\theta_n,\mathsf{undo}(l))} \langle \sigma_A l, origin(O).E, \emptyset, \emptyset\rangle \qquad \text{where } \theta_1, \ldots, \theta_n \text{ are the } \theta_i \text{ in } O$$

  in our system we have:

UNDO

$$\frac{}{T \mid P \xrightarrow{(\theta_1,\ldots,\theta_n,\mathsf{undo}(L))} init(\theta_1, \ldots, \theta_n, T) \mid \emptyset}$$

UNDO

$$\frac{}{\{S; A; L\} \xrightarrow{(S,\mathsf{undo}(L))} \{\emptyset; \emptyset; \emptyset\}}$$

UNDO

$$\frac{}{\sigma \xrightarrow{(S,\mathsf{undo}(L))} \sigma L}$$

$$T \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(S,\mathsf{undo}(L))} init(\theta_1, \ldots, \theta_n, T) \mid \emptyset \otimes_{\mathcal{R}} \{\emptyset; \emptyset; \emptyset\} \otimes_{\mathcal{R}} \sigma L$$

We know that $\{\langle \sigma_A, E, O, l\rangle, \langle T \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

Then after the transitions we have that $\{\langle \sigma_A l, origin(O).E, \emptyset, \emptyset\rangle$, $\langle init(\theta_1, \ldots, \theta_n, T) \mid \emptyset \otimes_{\mathcal{R}} \{\emptyset; \emptyset; \emptyset\} \otimes_{\mathcal{R}} \sigma L\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because the rules change the same element in σ_A and in σ such that $L = l$. $L = l$ because both rules applied put the logs equal to the $\emptyset$.

We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T). $origin(O)$ correspond to the expressions f of all the elements inside O, in our system all the elements inside O correspond to the threads with ids $\theta_1, \ldots, \theta_n$ and the expression of this threads is set to $t.\mathsf{init}$ that correspond

to f (thanks to the predicate $init(\theta_1, \ldots, \theta_n, T)$). A, P and S became equal to $\emptyset$ because O became equal to $\emptyset$ and then there is not element inside O.

- if in Abadi.system we have this transition:

  Trans Unprotected
  $$\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{unprotected}\ e]), f, a, P_o).O', l\rangle \xmapsto{(\theta,\mathsf{unprotect})} \langle \sigma_A, E.(\theta, \mathcal{P}[\mathsf{unprotected}\ e]).P_o, O.O', l - a\rangle$$
  $$\text{if } (\mathcal{P}[\mathsf{unprotected}\ e], f, a, P_o) \text{ does not conflict with } O.O'$$

  in our system we have:

  $$\frac{\dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{P}[\mathsf{unprotected}\ e]}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{unprtect})} T \mid t \mid P(\theta)} \qquad \dfrac{\theta \in S \wedge \text{no conflicts}}{\{S; A; L\} \xrightarrow{(\theta,\mathsf{unprotect})} S \setminus \theta \mid A \setminus A(\theta) \mid L - A(\theta)}}{T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta',\theta,\mathsf{unprtect})} T \mid t \mid P(\theta) \otimes_{\mathcal{R}} \{S \setminus \theta; A \setminus A(\theta); L - A(\theta)\} \otimes_{\mathcal{R}} \sigma}$$

  Unprotect: $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{unprotected}\ e]$ / $T \mid t \mid P \xrightarrow{(\theta,\mathsf{unprtect})} T \mid t \mid P(\theta)$

  Unprotect/Done: $\theta \in S \wedge \text{no conflicts}$ / $\{S; A; L\} \xrightarrow{(\theta,\mathsf{unprotect})} S \setminus \theta \mid A \setminus A(\theta) \mid L - A(\theta)$

  We know that $\{\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{unprotected}\ e]), f, a, P_o).O', l\rangle,$
  $\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{init} = f$ and mi $t.\mathsf{exp} = \mathcal{P}[\mathsf{unprotected}\ e]$.

  Then after the transitions we have that $\{\langle \sigma_A, E.(\theta, \mathcal{P}[\mathsf{unprotected}\ e]).P_o, O.O', l - a\rangle,$
  $\langle T \mid t \mid P(\theta) \otimes_{\mathcal{R}} \{S \setminus \theta; A \setminus A(\theta); L - A(\theta)\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

  This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. $L = l$ because the rules applied to the two systems changed the logs in the same way, $A(\theta)$ correspond to the elements in a.

  We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $\mathcal{P}[\mathsf{unprotected}\ e]$ (because $\theta \in S$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{unprotected}\ e]$). $P(\theta)$ activate the threads that correspond to P_o. $A - A(\theta)$ correspond to the elements in $O.O'$ because the two rules delete the elements that correspond to a.

- if in Abadi.system we have this transition:

  Trans Done
  $$\langle \sigma_A, E, O.((\theta, \mathsf{unit}), f, a, P_o).O', l\rangle \xmapsto{\mathsf{done}} \langle \sigma_A, E.P_o, O.O', l - a\rangle$$
  $$\text{if } (\mathsf{unit}, f, a, P) \text{ does not conflict with } O.O'$$

  in our system we have:

  $$\frac{\dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathsf{unit}}{T \mid t \mid P \xrightarrow{(\theta,\mathsf{done})} T \mid P(\theta)} \qquad \dfrac{\theta \in S \wedge \text{no conflicts}}{\{S; A; L\} \xrightarrow{(\theta,\mathsf{done})} S \setminus \theta \mid A \setminus A(\theta) \mid L - A(\theta)}}{T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta,\mathsf{done})} T \mid P(\theta) \otimes_{\mathcal{R}} \{S \setminus \theta; A \setminus A(\theta); L - A(\theta)\} \otimes_{\mathcal{R}} \sigma}$$

  Done: $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathsf{unit}$ / $T \mid t \mid P \xrightarrow{(\theta,\mathsf{done})} T \mid P(\theta)$

  Unprotect/Done: $\theta \in S \wedge \text{no conflicts}$ / $\{S; A; L\} \xrightarrow{(\theta,\mathsf{done})} S \setminus \theta \mid A \setminus A(\theta) \mid L - A(\theta)$

We know that $\{\langle sigma_A, E, O.((\theta, \mathsf{unit}), f, a, P_o).O', l\rangle$,
$\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{init} = f$ and mi $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$.

Then after the transitions we have that $\{\langle \sigma_A, E.P_o, O.O', l - a\rangle$,
$\langle T \mid P(\theta) \otimes_{\mathcal{R}} \{S \setminus \theta; A \setminus A(\theta); L - A(\theta)\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ because none of the rules applied to the two systems changed the memory. $L = l$ because the rules applied to the two systems changed the logs in the same way, $A(\theta)$ correspond to the elements in a.

We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T). $P(\theta)$ activate the threads that correspond to P_o. $A - A(\theta)$ correspond to the elements in $O.O'$ because the two rules delete the elements that correspond to a.

- if in Abadi.system we have this transition:

  Trans Close
  $$\langle \sigma_A, E.(\theta, \mathcal{E}[\mathsf{unprotected}\ V]).E', O, l\rangle \xrightarrow{(\theta, \mathsf{unp}(V))} \langle \sigma_A, E.(\theta, \mathcal{E}[V]).E', O, l\rangle$$

  in our system we have:

$$\frac{\dfrac{t.\mathsf{id} = \theta \qquad t.\mathsf{exp} = \mathcal{E}[\mathsf{unprotected}\ V]}{T \mid t \mid P \xrightarrow{(\theta, \mathsf{unp}(V))} T \mid t[exp \leftarrowtail \mathcal{E}[V]] \mid P} \text{ Close} \qquad \frac{\theta \notin S}{\{S; A; L\} \xrightarrow{(\theta, \mathsf{unp}(V))} \{S; A; L\}} \text{ U Step/Close}}{T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle (\theta, \mathsf{unp}(V)), \phi, (\theta, \mathsf{unp}(V))\rangle} T \mid t[exp \leftarrowtail \mathcal{E}[V]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma}$$

  We know that $\{\langle \sigma_A, E.(\theta, \mathcal{E}[\mathsf{unprotected}\ V]).E', O, l\rangle$,
  $\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{init} = f$ and mi $t.\mathsf{exp} = \mathcal{E}[\mathsf{unprotected}\ V]$.

  Then after the transitions we have that $\{\langle \sigma_A, E.(\theta, \mathcal{E}[V]).E', O, l\rangle$,
  $\langle T \mid t[exp \leftarrowtail \mathcal{E}[V]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle\} \in Rel$.

  This couple belongs to the relation Rel because we have that $\sigma = \sigma_A$ and $L = l$ because none of the rules applied to the two systems changed the memory and the logs.

  We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T) and the thread t correspond to $\mathcal{E}[V]$ (because $\theta \notin S$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{E}[V]$). A and P do not change in fact the rule does not modify O.

To proof the bi-simulation we can prove that given Rel as a strong simulation relation then if also Rel^{-1} is a strong simulation relation then Rel is a bi-simulation relation. Now we show the relationship Rel^{-1} derived directly from Rel.

Given $\{\langle T \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, O, l\rangle\}$ then this tuple is in Rel^{-1} if:

- $\sigma_A = \sigma$

- $\forall \theta_i \in S$:
  - we have $t_i \in T$ such that $t_i.\mathsf{id} = \theta_i$, $t_i.\mathsf{exp} = e_i$ and $t_i.\mathsf{init} = f_i$,
  - we have a set of $\lfloor t'_i \rfloor_{\theta_i} \in P$ where $t'_i.\mathsf{exp} = e'_i$, the set of this expression is indicate with E_i
  - then exist $o \in O$ such that $o = (e_i, f_i, A(\theta_i), E_i)$
- $\forall \theta_j \notin S$ we have $t_j \in T$ such that $t_j.\mathsf{id} = \theta_j$ and $t_j.\mathsf{exp} = e_j$, then exist $e_j \in E$
- $l = L$

Now we start to analyses all the possible case:

- if in our system we have:

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta,\mathsf{act})} T \mid t \mid P \otimes_{\mathcal{R}} \{S \cup \{\theta\}; A; L\} \otimes_{\mathcal{R}} \sigma$$

  in Abadi.system we have this transition:

  Trans Activate
  $\langle \sigma_A, E.(\theta, e).E', O, l\rangle \xmapsto{(\theta,\mathsf{act})} \langle \sigma_A, E.E', ((\theta, e), e, \emptyset, \emptyset).O, l\rangle$

  We know that $\{\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta, e).E', O, l\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = e$.

  Then after the transitions we have that $\{\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S \cup \{\theta\}; A; L\} \otimes_{\mathcal{R}} \sigma\rangle$, $\langle \sigma_A, E.E', ((\theta, e), e, \emptyset, \emptyset).O, l\rangle\} \in Rel^{-1}$.

  This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ and $l = L$ because none of the rules applied to the two systems changed the memory and the logs.

  We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T) and $(e, e, \emptyset, \emptyset)$ correspond to the thread t (in fact $\theta \in S$, $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = e$ and $t.\mathsf{init} = e$). A and P do not change in fact we add to O the element $(e, E, \emptyset, \emptyset).O$.

- if in our system we have:

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta,\mathsf{pure})} T \mid t[\mathsf{exp} \leftarrow \mathcal{P}[e[V/x]]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma$$

  in Abadi.system we have this transition:

  Trans Appl P
  $\langle \sigma_A, E, O.((\theta, \mathcal{P}[(\lambda\ x.e)\ V]), f, a, P_o).O', l\rangle \xmapsto{(\theta,\mathsf{pure})} \langle \sigma_A, E, O.((\theta, \mathcal{P}[e[V/x]]), f, a, P_o).O', l\rangle$

  We know that $\{\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, O.((\theta, \mathcal{P}[(\lambda\ x.e)\ V]), f, a, P_o).O', l\rangle\} \in Rel$ where $t.\mathsf{id} = \theta$, $t.\mathsf{init} = f$ and $t.\mathsf{exp} = \mathcal{P}[(\lambda\ x.e)\ V]$.

Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \leftarrow\!\!\shortmid \mathcal{P}[e[V/x]]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, O.((\theta, \mathcal{P}[e[V/x]]), f, a, P_o).O', l\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ and $l = L$ because none of the rules applied to the two systems changed the memory and the logs.

We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $(\mathcal{P}[e[V/x]], f, a, P_o)$ (because $\theta \in S$, $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[e[V/x]]$ and $t.\mathsf{init} = f$). A and P do not change in fact the rule does not modify a and P_o.

- if in our system we have:

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{pure})),\phi,(\theta,\mathsf{unp}(\mathsf{pure}))\rangle} T \mid t[exp \leftarrow\!\!\shortmid \mathcal{U}[e[V/x]]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have this transition:

Trans Appl U
$$\langle \sigma_A, E.(\theta,\mathcal{U}[(\lambda\ x.e)\ V]).E', O, l\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{pure}))} \langle \sigma_A, E.(\theta,\mathcal{U}[e[V/x]]).E', O, l\rangle$$

We know that $\{\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta,\mathcal{U}[(\lambda\ x.e)\ V]).E', O, ll\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{U}[(\lambda\ x.e)\ V]$ and $\theta \notin S$.

Then after the transitions we have that $\{\langle T \mid t[exp \leftarrow\!\!\shortmid \mathcal{U}[e[V/x]]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E.(\theta,\mathcal{U}[e[V/x]]).E', O, l\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ and $l = L$ because none of the rules applied to the two systems changed the memory and the logs.

We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[e[V/x]]$ (because $t.\mathsf{exp} = \mathcal{U}[e[V/x]]$)). We have that the element in O does not change in fact we have that all the tuple (e, f, a, P_o) in O have correspond to a thread in T, the ids of these threads are in S. Furthermore for all the tuple (e, f, a, P_o) in O each element in a and P_o has a correspondence with the elements in A and P.

- if in our system we have:

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta,\mathsf{ref}(r,v))} T \mid t[\mathsf{exp} \leftarrow\!\!\shortmid \mathcal{P}[r]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma[r \mapsto v]$$

in Abadi.system we have this transition:

Trans Ref P
$$\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{ref}\ V]), f, a, P_o).O', l\rangle \xmapsto{(\theta,\mathsf{ref}(r,v))} \langle \sigma_A[r \mapsto V], E, O.((\theta, \mathcal{P}[r]), f, a, P_o).O', l\rangle$$
$$\mathsf{if}\ r \in RefLoc\ -\ dom(\sigma)$$

We know that $\{\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle$, $\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{ref}\ V]), f, a, P_o).O', l\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{init} = f$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{ref}\ V]$.

Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \hookleftarrow \mathcal{P}[r]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle$, $\langle \sigma_A[r \mapsto V], E, O.((\theta, \mathcal{P}[r]), f, a, P_o).O', l\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because the rules add the same element to σ and to σ_A. $l = L$ because none of the rules applied to the two systems changed the logs.

We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $(\mathcal{P}[r], f, a, P_o)$ (because $\theta \in S$, $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[r]$ and $t.\mathsf{init} = f$). A and P do not change in fact the rule does not modify a and P_o.

- if in our system we have:

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{ref}(r,v))),(\theta,\mathsf{unp}(\mathsf{ref}(r,v))),(\theta,\mathsf{unp}(\mathsf{ref}(r,v)))\rangle} T \mid t[exp \hookleftarrow \mathcal{U}[r]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma[r \mapsto v]$$

in Abadi.system we have this transition:

$$\textsc{Trans Ref U}$$
$$\langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{ref}\ V]).E', O, l\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{ref}(r,v)))} \langle \sigma_A[r \mapsto V], E.(\theta, \mathcal{U}[r]).E', O, l\rangle \quad \mathsf{if}\ r \in RefLoc\ -\ dom(\sigma)$$

We know that $\{\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle$, $\langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{ref}\ V]).E', O, l\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{U}[\mathsf{ref}\ V]$ and $\theta \notin S$.

Then after the transitions we have that $\{\langle T \mid t[exp \hookleftarrow \mathcal{U}[r]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle$, $\langle \sigma_A[r \mapsto V], E.(\theta, \mathcal{U}[r]).E', O, l\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because the rules add the same element to σ and to σ_A. $l = L$ because none of the rules applied to the two systems changed the logs.

We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[r]$ (because $t.\mathsf{exp} = \mathcal{U}[r]$)). We have that the element in O does not change in fact we have that all the tuple (e, f, a, P_o) in O have correspond to a thread in T, the ids of these threads are in S. Furthermore for all the tuple (e, f, a, P_o) in O each element in a and P_o has a correspondence with the elements in A and P.

- if in our system we have:

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{rd}(r,v)),(\theta,\mathsf{rd}(r,v)),(\theta,\mathsf{rd}(r,v))\rangle} T \mid t[\mathsf{exp} \hookleftarrow \mathcal{P}[v]] \mid P \otimes_{\mathcal{R}} \{S; A \cup \{\theta, r\}; L\} \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have this transition:

$$\textsc{Trans Deref P}$$
$$\langle \sigma_A, E, O.((\theta, \mathcal{P}[!r]), f, a, P_o).O', l\rangle \xmapsto{(\theta,\mathsf{rd}(r,v))} \langle \sigma_A, E, O.((\theta, \mathcal{P}[V]), f, r.a, P_o).O', l\rangle \quad \text{if } \sigma(r) = V$$

We know that $\{\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle,$
$\langle \sigma_A, E, O.((\theta, \mathcal{P}[!r]), f, a, P_o).O', l\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{init} = f$ and $t.\mathsf{exp} = \mathcal{P}[!r]$.

Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \leftarrow\!\!\shortmid \mathcal{P}[v]] \mid P \otimes_{\mathcal{R}} \{S; A \cup \{\theta, r\}; L\} \otimes_{\mathcal{R}} \sigma\rangle,$
$\langle \sigma_A, E, O.((\theta, \mathcal{P}[V]), f, r.a, P_o).O', l\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ and $l = L$ because none of the rules applied to the two systems changed the memory and the logs.

We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $(\mathcal{P}[V], f, r.a, P_o)$ (because $\theta \in S$, $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[V]$ and $t.\mathsf{init} = f$). All the other element in O and O' does not change, $(\mathcal{P}[V], f, r.a, P_o)$ add r to a in fact in our system the element $\{\theta, r\}$ is added to A. P does not change in fact the rule does not modify P_o.

- if in our system we have:

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{rd}(r,v))),(\theta,\mathsf{unp}(\mathsf{rd}(r,v))),(\theta,\mathsf{unp}(\mathsf{rd}(r,v)))\rangle} T \mid t[\mathsf{exp} \leftarrow\!\!\shortmid \mathcal{U}[v]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have this transition:

$$\textsc{Trans Deref U}$$
$$\langle \sigma_A, E.(\theta,\mathcal{U}[!r]).E', O, l\rangle \xmapsto{(\theta,\mathsf{unp}(\mathsf{rd}(r,v)))} \langle \sigma_A, E.(\theta,\mathcal{U}[V]).E', O, l\rangle \qquad \text{if } \sigma(r) = V$$

We know that $\{\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle,\ \langle \sigma_A, E.(\theta,\mathcal{U}[!r]).E', O, l\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{U}[!r]$ and $\theta \notin S$.

Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \leftarrow\!\!\shortmid \mathcal{U}[v]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle,$
$\langle \sigma_A, E.(\theta,\mathcal{U}[V]).E', O, l\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ and $l = L$ because none of the rules applied to the two systems changed the memory and the logs.

We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[v]$ (because $t.\mathsf{exp} = \mathcal{U}[v]$)). We have that the element in O does not change in fact we have that all the tuple (e, f, a, P_o) in O have correspond to a thread in T, the ids of these threads are in S. Furthermore for all the tuple (e, f, a, P_o) in O each element in a and P_o has a correspondence with the elements in A and P.

- if in our system we have:

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta,\mathsf{wr}(r,v))} T \mid t[\mathsf{exp} \leftarrow \mathcal{P}[\mathsf{unit}]] \mid P \otimes_{\mathcal{R}} \{S; A \cup \{\theta, r\}; L.[r \mapsto \sigma(r)]\} \otimes_{\mathcal{R}} \sigma[r \mapsto v]$$

in Abadi.system we have this transition:

Trans Set P

$$\langle \sigma_A, E, O.((\theta, \mathcal{P}[r := V]), f, a, P_o).O', l\rangle \xrightarrow{(\theta,\mathsf{wr}(r,v))} \langle \sigma_A[r \mapsto V], E, O.((\theta, \mathcal{P}[\mathsf{unit}]), f, r.a, P_o).O', l'\rangle$$
$$\text{where } l' = \mathsf{if}\ r \in dom(l)\ \mathsf{then}\ l\ \mathsf{else}\ l.[r \mapsto V]$$

We know that $\{\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle$,
$\langle \sigma_A, E, O.((\theta, \mathcal{P}[r := V]), f, a, P_o).O', l\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{init} = f$ and $t.\mathsf{exp} = \mathcal{P}[r := V]$.

Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \leftarrow \mathcal{P}[\mathsf{unit}]] \mid P \otimes_{\mathcal{R}} \{S; A \cup \{\theta, r\}; L.[r \mapsto \sigma(r)]\} \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle$,
$\langle \sigma_A[r \mapsto V], E, O.((\theta, \mathcal{P}[\mathsf{unit}]), f, r.a, P_o).O', l'\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ and $l = L$ because the rules applied to the two systems changed the memory and the logs in the same way.

We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $(\mathcal{P}[\mathsf{unit}], f, a, P_o)$ (because $\theta \in S$, $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$ and $t.\mathsf{init} = f$). All the other element in O and O' does not change, $(\mathcal{P}[\mathsf{unit}], f, r.a, P_o)$ add r to a in fact in our system the element $\{\theta, r\}$ is added to A. P does not change in fact the rule does not modify P_o.

- if in our system we have:

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(\mathsf{wr}(r,v))),(\theta,\mathsf{unp}(\mathsf{wr}(r,v))),(\theta,\mathsf{unp}(\mathsf{wr}(r,v)))\rangle} T \mid t[\mathsf{exp} \leftarrow \mathcal{U}[\mathsf{unit}]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma[r \mapsto v]$$

in Abadi.system we have this transition:

Trans Set U

$$\langle \sigma_A, E.(\theta, \mathcal{U}[r := V]).E', O, l\rangle \xrightarrow{(\theta,\mathsf{unp}(\mathsf{wr}(r,v)))} \langle \sigma_A[r \mapsto V], E.(\theta, \mathcal{U}[\mathsf{unit}]).E', O, l\rangle$$

We know that $\{\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle$, $\langle \sigma_A, E.(\theta, \mathcal{U}[r := V]).E', O, l\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{U}[r := V]$ and $\theta \notin S$.

Then after the transitions we have that $\{\langle T \mid t[\mathsf{exp} \leftarrow \mathcal{U}[\mathsf{unit}]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma[r \mapsto v]\rangle$, $\langle \sigma_A[r \mapsto V], E.(\theta, \mathcal{U}[\mathsf{unit}]).E', O, l\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because the rules add the same element to σ and to σ_A. $l = L$ because none of the rules applied to the two systems changed the logs.

We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond

to $\mathcal{U}[]$ (because $t.\mathsf{exp} = \mathcal{U}[\mathsf{unit}]$)). We have that the element in O does not change in fact we have that all the tuple (e, f, a, P_o) in O have correspond to a thread in T, the ids of these threads are in S. Furthermore for all the tuple (e, f, a, P_o) in O each element in a and P_o has a correspondence with the elements in A and P.

- if in our system we have:

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta, \mathsf{async}(e'))} T \mid t[exp \hookleftarrow \mathcal{P}[\mathsf{unit}]] \mid P \mid \lfloor t'[id \hookleftarrow \theta', exp \hookleftarrow e'] \rfloor_\theta \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have this transition:

Trans Async P
$$\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{async}\ e']), f, a, P_o).O', l\rangle \xmapsto{(\theta, \mathsf{async}(e'))} \langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{unit}]), f, a, (\theta', e').P_o).O', l\rangle$$

We know that $\{\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle$,
$\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{async}\ e']), f, a, P_o).O', l\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{init} = f$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{async}\ e']$.

Then after the transitions we have that $\{\langle T \mid t[exp \hookleftarrow \mathcal{P}[\mathsf{unit}]] \mid P \mid \lfloor t'[id \hookleftarrow \theta', exp \hookleftarrow e'] \rfloor_\theta \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle$,
$\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{unit}]), f, a, (\theta', e').P_o).O', l\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ and $l = L$ because none of the rules applied to the two systems changed the memory and the logs.

We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $(\mathcal{P}[\mathsf{unit}], f, a, e'.P_o)$ (because $\theta \in S$, $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$ and $t.\mathsf{init} = f$). A does not change in fact the rule does not modify a. $\lfloor t' \rfloor_\theta$ correspond to e' in fact $t'.\mathsf{exp} = e'$, instead the other elements in P_o have a thread in P that correspond to them.

- if in our system we have:

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta, \mathsf{unp}(\mathsf{async}(e')))} T \mid t[exp \hookleftarrow \mathcal{U}[\mathsf{unit}]] \mid t'[id \hookleftarrow \theta', exp \hookleftarrow e'] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have this transition:

Trans Async U
$$\langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{async}\ e']).E', O, l\rangle \xmapsto{(\theta, \mathsf{unp}(\mathsf{async}(e')))} \langle \sigma_A, E.(\theta', e').(\theta, \mathcal{U}[\mathsf{unit}]).E', O, l\rangle$$

We know that $\{\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle$, $\langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{async}\ e']).E', O, l\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{U}[\mathsf{async}\ e']$ and $\theta \notin S$.

Then after the transitions we have that $\{\langle T \mid t[exp \hookleftarrow \mathcal{U}[\mathsf{unit}]] \mid t'[id \hookleftarrow \theta', exp \hookleftarrow e'] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle$, $\langle \sigma_A, E.(\theta', e').(\theta, \mathcal{U}[\mathsf{unit}]).E', O, l\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ and $l = L$ because none of the rules applied to the two systems changed the memory and the logs.

We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[\mathsf{unit}]$ (because $t.\mathsf{exp} = \mathcal{U}[\mathsf{unit}]$)). e' correspond to t' in fact $t'.\mathsf{exp} = e'$. We have that the element in O does not change in fact we have that all the tuple (e, f, a, P_o) in O have correspond to a thread in T, the ids of these threads are in S. Furthermore for all the tuple (e, f, a, P_o) in O each element in a and P_o has a correspondence with the elements in A and P.

- if in our system we have:

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta,\mathsf{blockUntil(true)})} T \mid t[exp \leftarrow \mathcal{P}[\mathsf{unit}]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have this transition:

Trans Block P

$$\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{blockUntil\ true}]), f, a, P_o).O', l\rangle \xrightarrow{(\theta,\mathsf{blockUntil(true)})} \langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{unit}]), f, a, P_o).O', l\rangle$$

We know that $\{\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle$, $\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{blockUntil\ true}]), f, a, P_o).O', l\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{init} = f$ and mi $t.\mathsf{exp} = \mathcal{P}[\mathsf{blockUntil\ true}]$.

Then after the transitions we have that $\{\langle T \mid t[exp \leftarrow \mathcal{P}[\mathsf{unit}]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle$, $\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{unit}]), f, a, P_o).O', l\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ and $l = L$ because none of the rules applied to the two systems changed the memory and the logs.

We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $(\mathcal{P}[\mathsf{unit}], f, a, P_o)$ (because $\theta \in S$, $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$ and $t.\mathsf{init} = f$). A and P do not change in fact the rule does not modify a and P_o.

- if in our system we have:

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp(blockUntil\ true)}),\phi,(\theta,\mathsf{unp(blockUntil\ true)})\rangle} T \mid t[exp \leftarrow \mathcal{U}[\mathsf{unit}] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have this transition:

Trans Block U

$$\langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{blockUntil\ true}]).E', O, l\rangle \xrightarrow{(\theta,\mathsf{unp(blockUntil(true))})} \langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{unit}]).E', O, l\rangle$$

We know that $\{\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle$, $\langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{blockUntil\ true}]).E', O, l\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{exp} = \mathcal{U}[\mathcal{U}[\mathsf{blockUntil\ true}]]$ and $\theta \notin S$.

Then after the transitions we have that $\{\langle T \mid t[exp \leftarrow \mathcal{U}[\mathsf{unit}]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle$, $\langle \sigma_A, E.(\theta, \mathcal{U}[\mathsf{unit}]).E', O, l\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ and $l = L$ because none of the rules applied to the two systems changed the memory and the logs.

We also have that all the expressions in E and E' have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T), the thread t correspond to $\mathcal{U}[\mathsf{unit}]$ (because $t.\mathsf{exp} = \mathcal{U}[\mathsf{unit}]$)). We have that the element in O does not change in fact we have that all the tuple (e, f, a, P_o) in O have correspond to a thread in T, the ids of these threads are in S. Furthermore for all the tuple (e, f, a, P_o) in O each element in a and P_o has a correspondence with the elements in A and P.

- if in our system we have:

$$T \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(S, \mathsf{undo}(L))} init(\theta_1, \ldots, \theta_n, T) \mid \emptyset \otimes_{\mathcal{R}} \{\emptyset; \emptyset; \emptyset\} \otimes_{\mathcal{R}} \sigma L$$

in Abadi.system we have this transition:

Trans Undo
$\langle \sigma_A, E, O, l\rangle \xmapsto{(\theta_1, \ldots, \theta_n, \mathsf{undo}(l))} \langle \sigma_A l, origin(O).E, \emptyset, \emptyset\rangle$ where $\theta_1, \ldots, \theta_n$ are the θ_i in O

We know that $\{\langle T \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle, \langle \sigma_A, E, O, l\rangle\} \in Rel^{-1}$.

Then after the transitions we have that $\{\langle init(\theta_1, \ldots, \theta_n, T) \mid \emptyset \otimes_{\mathcal{R}} \{\emptyset; \emptyset; \emptyset\} \otimes_{\mathcal{R}} \sigma L\rangle$, $\langle \sigma_A l, origin(O).E, \emptyset, \emptyset\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because the rules change the same element in σ_A and in σ such that $l = L$. $l = L$ because both rules applied put the logs equal to the $\emptyset$.

We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T). $origin(O)$ correspond to the expressions f of all the elements inside O, in our system all the elements inside O correspond to the threads with ids $\theta_1, \ldots, \theta_n$ and the expression of this threads is set to $t.\mathsf{init}$ that correspond to f (thanks to the predicate $init(\theta_1, \ldots, \theta_n, T)$). A, P and S became equal to $\emptyset$ because O became equal to $\emptyset$ and then there is not element inside O.

- if in our system we have:

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta, \mathsf{unprtect})} T \mid t \mid P(\theta) \otimes_{\mathcal{R}} \{S \setminus \theta; A \setminus A(\theta); L - A(\theta)\} \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have this transition:

Trans Unprotected
$$\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{unprotected}\ e]), f, a, P_o).O', l\rangle \xmapsto{(\theta,\mathsf{unprotect})} \langle \sigma_A, E.(\theta, \mathcal{P}[\mathsf{unprotected}\ e]).P_o, O.O', l - a\rangle$$
$$\text{if } (\mathcal{P}[\mathsf{unprotected}\ e], f, a, P_o) \text{ does not conflict with } O.O'$$

We know that $\{\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle$,
$\langle \sigma_A, E, O.((\theta, \mathcal{P}[\mathsf{unprotected}\ e]), f, a, P_o).O', l\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{init} = f$ and mi $t.\mathsf{exp} = \mathcal{P}[\mathsf{unprotected}\ e]$.

Then after the transitions we have that $\{\langle T \mid t \mid P(\theta) \otimes_{\mathcal{R}} \{S \setminus \theta; A \setminus A(\theta); L - A(\theta)\} \otimes_{\mathcal{R}} \sigma\rangle$,
$\langle \sigma_A, E.(\theta, \mathcal{P}[\mathsf{unprotected}\ e]).P_o, O.O', l - a\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. $l = L$ because the rules applied to the two systems changed the logs in the same way, $A(\theta)$ correspond to the elements in a.

We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T) and the thread t correspond to $\mathcal{P}[\mathsf{unprotected}\ e]$ (because $\theta \in S$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{P}[\mathsf{unprotected}\ e]$). $P(\theta)$ activate the threads that correspond to P_o. $A - A(\theta)$ correspond to the elements in $O.O'$ because the two rules delete the elements that correspond to a.

- if in our system we have:

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{(\theta,\mathsf{done})} T \mid P(\theta) \otimes_{\mathcal{R}} \{S \setminus \theta; A \setminus A(\theta); L - A(\theta)\} \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have this transition:

Trans Done
$$\langle \sigma_A, E, O.((\theta, \mathsf{unit}), f, a, P_o).O', l\rangle \xmapsto{(\theta,\mathsf{done})} \langle \sigma_A, E.P_o, O.O', l - a\rangle$$
$$\text{if } (\mathsf{unit}, f, a, P) \text{ does not conflict with } O.O'$$

We know that $\{\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle$,
$\langle \sigma_A, E, O.((\theta, \mathsf{unit}), f, a, P_o).O', l\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{init} = f$ and mi $t.\mathsf{exp} = \mathcal{P}[\mathsf{unit}]$.

Then after the transitions we have that $\{\langle T \mid P(\theta) \otimes_{\mathcal{R}} \{S \setminus \theta; A \setminus A(\theta); L - A(\theta)\} \otimes_{\mathcal{R}} \sigma\rangle$,
$\langle \sigma_A, E.P_o, O.O', l - a\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ because none of the rules applied to the two systems changed the memory. $l = L$ because the rules applied to the two systems changed the logs in the same way, $A(\theta)$ correspond to the elements in a.

We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E or T). $P(\theta)$ activate the threads that

correspond to P_o. $A - A(\theta)$ correspond to the elements in $O.O'$ because the two rules delete the elements that correspond to a.

- if in our system we have:

$$T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma \xrightarrow{\langle(\theta,\mathsf{unp}(V)),\phi,(\theta,\mathsf{unp}(V))\rangle} T \mid t[exp \leftarrow\!\shortmid \mathcal{E}[V]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma$$

in Abadi.system we have this transition:

TRANS CLOSE
$$\langle \sigma_A, E.(\theta, \mathcal{E}[\mathsf{unprotected}\ V]).E', O, l\rangle \xrightarrow{(\theta,\mathsf{unp}(V))} \langle \sigma_A, E.(\theta, \mathcal{E}[V]).E', O, l\rangle$$

We know that $\{\langle T \mid t \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle$, $\langle \sigma_A, E.(\theta, \mathcal{E}[\mathsf{unprotected}\ V]).E', O, l\rangle\} \in Rel^{-1}$ where $t.\mathsf{id} = \theta$, $t.\mathsf{init} = f$ and mi $t.\mathsf{exp} = \mathcal{E}[\mathsf{unprotected}\ V]$.

Then after the transitions we have that $\{\langle T \mid t[exp \leftarrow\!\shortmid \mathcal{E}[V]] \mid P \otimes_{\mathcal{R}} \{S; A; L\} \otimes_{\mathcal{R}} \sigma\rangle$, $\langle \sigma_A, E.(\theta, \mathcal{E}[V]).E', O, l\rangle\} \in Rel^{-1}$.

This couple belongs to the relation Rel^{-1} because we have that $\sigma_A = \sigma$ and $l = L$ because none of the rules applied to the two systems changed the memory and the logs.

We also have that all the expressions in E have a correspond thread in T (none of the rules applied to the two systems changed either E and E' or T) and the thread t correspond to $\mathcal{E}[V]$ (because $\theta \notin S$, $t.\mathsf{id} = \theta$ and $t.\mathsf{exp} = \mathcal{E}[V]$). A and P do not change in fact the rule does not modify O.

□

Appendix C

Monolithic and Compositional LTSs

C.1 Bi-similarity between Monolithic and Compositional LTSs

In order to show the bi-simulation between the monolithic LTS $(\mathcal{T} \mid \mathcal{M} \mid \mathcal{S})$ and the compositional LTS $(\mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S})$ we recall the notions of simulation (Definition 30) and bi-simulation (Definition 31) that we use.

Definition 30 (F-Simulation)**.** *Given two LTSs $L_1 = (S_1, \Lambda_1, \rightarrow_1)$ and $L_2 = (S_2, \Lambda_2, \rightarrow_2)$, a function $f : \Lambda_1 \rightarrow \Lambda_2$ and a relation $R \subseteq S_1 \times S_2$ is a f-simulation (indicated with $L_1 \overset{f}{\prec}_R L_2$) if and only if for every pair of states $(p, q) \in R$ and all labels $\lambda \in \Lambda$:*

- *if $p \overset{\lambda}{\rightarrow} p'$, then there is $q \overset{f(\lambda)}{\rightarrow} q'$ such that $(p', q') \in R$.*

Definition 31 (F-Bisimulation)**.** *Given two LTSs $L_1 = (S_1, \Lambda_1, \rightarrow_1)$ and $L_2 = (S_2, \Lambda_2, \rightarrow_2)$, a function $f : \Lambda_1 \rightarrow \Lambda_2$ and a relation $R \subseteq S_1 \times S_2$ is a f-bisimulation (indicated with $L_1 \overset{f}{\sim}_R L_2$) if $L_1 \overset{f}{\prec}_R L_2$ and $L_2 \overset{f^{-1}}{\prec}_{R^{-1}} L_1$*

We recall also the function that we use for the relation between the label (Definition 44).

Definition 44. *The function $\varphi(\cdot)$ is defined over labels:*

$$\varphi(((\theta, \alpha), \phi, \phi)) = (\theta, \alpha),$$
$$\varphi(((\theta, \mathsf{ref}(r, v)), (\theta, \mathsf{ref}(r, v)), \phi)) = (\theta, \mathsf{ref}(r, v)),$$
$$\varphi(((\theta, \mathsf{rd}(r, v)), (\theta, \mathsf{rd}(r, v)), \phi)) = (\theta, \mathsf{rd}(r, v)),$$
$$\varphi(((\theta, \mathsf{wr}(r, v)), (\theta, \mathsf{wr}(r, v)), \phi)) = (\theta, \mathsf{wr}(r, v)),$$
$$\varphi(((\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l)))) = (\theta, \mathsf{acq}(l)),$$
$$\varphi(((\theta, \mathsf{rel}(l)), \phi, (\theta, \mathsf{rel}(l)))) = (\theta, \mathsf{rel}(l))$$

where $\alpha = \mathsf{sp}(\theta'), \mathsf{pure}$.

We also give the definition of $\varphi^{-1}(\cdot)$, the function that we use in the second part of the bi simulation.

Definition 45. *The function $\varphi^{-1}(\cdot)$ is defined over labels:*

$$\varphi^{-1}((\theta,\alpha)) = ((\theta,\alpha),\phi,\phi),$$
$$\varphi^{-1}((\theta,\mathsf{ref}(r,v))) = ((\theta,\mathsf{ref}(r,v)),(\theta,\mathsf{ref}(r,v)),\phi),$$
$$\varphi^{-1}((\theta,\mathsf{rd}(r,v))) = ((\theta,\mathsf{rd}(r,v)),(\theta,\mathsf{rd}(r,v)),\phi),$$
$$\varphi^{-1}((\theta,\mathsf{wr}(r,v))) = ((\theta,\mathsf{wr}(r,v)),(\theta,\mathsf{wr}(r,v)),\phi),$$
$$\varphi^{-1}((\theta,\mathsf{acq}(l))) = ((\theta,\mathsf{acq}(l)),\phi,(\theta,\mathsf{acq}(l))),$$
$$\varphi^{-1}((\theta,\mathsf{rel}(l))) = ((\theta,\mathsf{rel}(l)),\phi,(\theta,\mathsf{rel}(l)))$$

Let us now recall the Theorem 20 and show its complete proof.

Theorem 20 (Bi-Simulation)**.** $(\mathcal{T}\otimes_{\mathcal{R}}\mathcal{M}\otimes_{\mathcal{R}}\mathcal{S}) \overset{\mathbf{lab}}{\sim} (\mathcal{T}_1 \mid \mathcal{M}_1 \mid \mathcal{S}_1)$.

Proof. Firstly, we need to define the relation between the states of the compositional LTS and of the monolithic LTS, we define the relation as a set of couple $\mathcal{RL} = \{\langle(\mathcal{T}\otimes_{\mathcal{R}}\mathcal{M}\otimes_{\mathcal{R}}\mathcal{S}),(\mathcal{T}_1 \mid \mathcal{M}_1 \mid \mathcal{S}_1)\rangle$ such that $\mathcal{T}=\mathcal{T}_1,\mathcal{M}=\mathcal{M}_1,\mathcal{S}=\mathcal{S}_1\}$. Now we show by case that for each transitions done by the compositional LTS (with the transformed label) then the monolithic LTS can do the same transitions. Now we proceed by cases for on the labels:

- We consider the case where $\varphi(((\theta,\mathsf{ref}(r,v)),(\theta,\mathsf{ref}(r,v)),\phi)) = (\theta,\mathsf{ref}(r,v))$, then in the compositional LTS we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{ref}(r,v))} e'}{\langle\theta,e\rangle \xrightarrow{(\theta,\mathsf{ref}(r,v))} \langle\theta,e'\rangle}}{\langle\theta,e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{ref}(r,v))} \langle\theta,e'\rangle \mid \mathcal{T}} \quad \dfrac{\dfrac{}{(r\mapsto\circ) \xrightarrow{(\theta,\mathsf{ref}(r,v))} (r\mapsto v)}}{(r\mapsto\circ)\mid\mathcal{M} \xrightarrow{(\theta,\mathsf{ref}(r,v))} (r\mapsto v)\mid\mathcal{M}} \quad \mathcal{S}\xrightarrow{\phi}\mathcal{S} \quad \langle((\theta,\mathsf{ref}(r,v)),(\theta,\mathsf{ref}(r,v)),\phi)\rangle\in\mathcal{R}}{\langle\theta,e\rangle \mid \mathcal{T}\otimes_{\mathcal{R}}(r\mapsto\circ)\mid\mathcal{M}\otimes_{\mathcal{R}}\mathcal{S} \xrightarrow{((\theta,\mathsf{ref}(r,v)),(\theta,\mathsf{ref}(r,v)),\phi)} \langle\theta,e'\rangle \mid \mathcal{T}\otimes_{\mathcal{R}}(r\mapsto v)\mid\mathcal{M}\otimes_{\mathcal{R}}\mathcal{S}}$$

  in the monolithic LTS we have the following configuration $\langle\theta,e\rangle \mid \mathcal{T} \mid (r\mapsto\circ) \mid \mathcal{M} \mid \mathcal{S}$ obtained by $\mathcal{RL}$, and we consider the label $(\theta,\mathsf{ref}(r,v))$. Then, we obtain:

$$\textsc{Par}\ \frac{\textsc{Alloc}\ \dfrac{e \xhookrightarrow{(\theta,\mathsf{ref}(r,v))} e'}{\langle\theta,e\rangle \mid (r\mapsto\circ) \xrightarrow{(\theta,\mathsf{ref}(r,v))} \langle\theta,e'\rangle \mid (r\mapsto v)}}{\langle\theta,e\rangle \mid \mathcal{T} \mid (r\mapsto\circ) \mid \mathcal{M} \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{ref}(r,v))} \langle\theta,e'\rangle \mid \mathcal{T} \mid (r\mapsto v) \mid \mathcal{M} \mid \mathcal{S}}$$

  we can easily see that the final states are still related.

- We consider the case where $\varphi((\theta,\mathsf{rd}(r,v)),(\theta,\mathsf{rd}(r,v)),\phi) = (\theta,\mathsf{rd}(r,v))$, then in the

compositional LTS we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xrightarrow{(\theta,\mathsf{rd}(r,v))} e'}{\langle\theta,e\rangle \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle\theta,e'\rangle}}{\langle\theta,e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle\theta,e'\rangle \mid \mathcal{T}} \quad \dfrac{\dfrac{}{(r\mapsto v) \xrightarrow{(\theta,\mathsf{rd}(r,v))} (r\mapsto v)}}{(r\mapsto v) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{rd}(r,v))} (r\mapsto v) \mid \mathcal{M}} \quad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \quad \langle((\theta,\mathsf{rd}(r,v)),(\theta,\mathsf{rd}(r,v)),\phi)\rangle \in \mathcal{R}}{\langle\theta,e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r\mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\mathsf{rd}(r,v)),(\theta,\mathsf{rd}(r,v)),\phi)} \langle\theta,e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r\mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

in the monolithic LTS we have the following configuration $\langle\theta,e\rangle \mid \mathcal{T} \mid (r\mapsto v) \mid \mathcal{M} \mid \mathcal{S}$ obtained by $\mathcal{RL}$, and we consider the label $(\theta,\mathsf{rd}(r,v))$. Then, we obtain:

$$\textsc{Par}\ \frac{\textsc{Read}\ \dfrac{e \xrightarrow{(\theta,\mathsf{rd}(r,v))} e'}{\langle\theta,e\rangle \mid (r\mapsto v) \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle\theta,e'\rangle \mid (r\mapsto v)}}{\langle\theta,e\rangle \mid \mathcal{T} \mid (r\mapsto v) \mid \mathcal{M} \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle\theta,e'\rangle \mid \mathcal{T} \mid (r\mapsto v) \mid \mathcal{M} \mid \mathcal{S}}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi((\theta,\mathsf{wr}(r,v)),(\theta,\mathsf{wr}(r,v)),\phi) = (\theta,\mathsf{wr}(r,v))$, then in the compositional LTS we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xrightarrow{(\theta,\mathsf{wr}(r,v))} e'}{\langle\theta,e\rangle \xrightarrow{(\theta,\mathsf{wr}(r,v))} \langle\theta,e'\rangle}}{\langle\theta,e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{wr}(r,v))} \langle\theta,e'\rangle \mid \mathcal{T}} \quad \dfrac{\dfrac{}{(r\mapsto v_{old}) \xrightarrow{(\theta,\mathsf{wr}(r,v))} (r\mapsto v)}}{(r\mapsto v_{old}) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{wr}(r,v))} (r\mapsto v) \mid \mathcal{M}} \quad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \quad \langle((\theta,\mathsf{wr}(r,v)),(\theta,\mathsf{wr}(r,v)),\phi)\rangle \in \mathcal{R}}{\langle\theta,e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r\mapsto v_{old}) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\mathsf{wr}(r,v)),(\theta,\mathsf{wr}(r,v)),\phi)} \langle\theta,e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r\mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

in the monolithic LTS we have the following configuration $\langle\theta,e\rangle \mid \mathcal{T} \mid (r\mapsto v_{old}) \mid \mathcal{M} \mid \mathcal{S}$ obtained by $\mathcal{RL}$, and we consider the label $(\theta,\mathsf{wr}(r,v))$. Then, we obtain:

$$\textsc{Par}\ \frac{\textsc{Write}\ \dfrac{e \xrightarrow{(\theta,\mathsf{wr}(r,v))} e'}{\langle\theta,e\rangle \mid (r\mapsto v_{old}) \xrightarrow{(\theta,\mathsf{wr}(r,v))} \langle\theta,e'\rangle \mid (r\mapsto v)}}{\langle\theta,e\rangle \mid \mathcal{T} \mid (r\mapsto v_{old}) \mid \mathcal{M} \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{wr}(r,v))} \langle\theta,e'\rangle \mid \mathcal{T} \mid (r\mapsto v) \mid \mathcal{M} \mid \mathcal{S}}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi((\theta,\mathsf{acq}(l)),\phi,(\theta,\mathsf{acq}(l))) = (\theta,\mathsf{acq}(l))$, then in the

compositional LTS we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{acq}(l))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \frac{}{(l \mapsto \circ) \xrightarrow{(\theta,\mathsf{acq}(l))} (l \mapsto \theta)}}{(l \mapsto \circ) \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{acq}(l))} (l \mapsto \theta) \mid \mathcal{S}} \quad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad \langle((\theta, \mathsf{acq}(l)), \phi, (\theta, \mathsf{acq}(l)))\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \circ) \mid \mathcal{S} \xrightarrow{((\theta,\mathsf{acq}(l)),\phi,(\theta,\mathsf{acq}(l)))} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \theta) \mid \mathcal{S}}$$

in the monolithic LTS we have the following configuration $\langle\theta, e\rangle \mid \mathcal{T} \mid \mathcal{M} \mid (l \mapsto \circ) \mid \mathcal{S}$ obtained by $\mathcal{RL}$, and we consider the label $(\theta, \mathsf{acq}(l))$. Then, we obtain:

$$\textsc{Par}\ \frac{\textsc{Acquire}\ \dfrac{e \xhookrightarrow{(\theta,\mathsf{acq}(l))} e'}{\langle\theta, e\rangle \mid (l \mapsto \circ) \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta, e'\rangle \mid (l \mapsto \theta)}}{\langle\theta, e\rangle \mid \mathcal{T} \mid \mathcal{M} \mid (l \mapsto \circ) \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta, e'\rangle \mid \mathcal{T} \mid \mathcal{M} \mid (l \mapsto \theta) \mid \mathcal{S}}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi((\theta, \mathsf{rel}(l)), \phi, (\theta, \mathsf{rel}(l))) = (\theta, \mathsf{rel}(l))$, then in the compositional LTS we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{rel}(l))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{rel}(l))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{rel}(l))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \frac{}{(l \mapsto \theta) \xrightarrow{(\theta,\mathsf{rel}(l))} (l \mapsto \circ)}}{(l \mapsto \theta) \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{rel}(l))} (l \mapsto \circ) \mid \mathcal{S}} \quad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad \langle((\theta, \mathsf{rel}(l)), \phi, (\theta, \mathsf{rel}(l)))\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \theta) \mid \mathcal{S} \xrightarrow{((\theta,\mathsf{rel}(l)),\phi,(\theta,\mathsf{rel}(l)))} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \circ) \mid \mathcal{S}}$$

in the monolithic LTS we have the following configuration $\langle\theta, e\rangle \mid \mathcal{T} \mid \mathcal{M} \mid (l \mapsto \theta) \mid \mathcal{S}$ obtained by $\mathcal{RL}$, and we consider the label $(\theta, \mathsf{rel}(l))$. Then, we obtain:

$$\textsc{Par}\ \frac{\textsc{Release}\ \dfrac{e \xhookrightarrow{(\theta,\mathsf{rel}(l))} e'}{\langle\theta, e\rangle \mid (l \mapsto \theta) \xrightarrow{(\theta,\mathsf{rel}(l))} \langle\theta, e'\rangle \mid (l \mapsto \circ)}}{\langle\theta, e\rangle \mid \mathcal{T} \mid \mathcal{M} \mid (l \mapsto \theta) \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{rel}(l))} \langle\theta, e'\rangle \mid \mathcal{T} \mid \mathcal{M} \mid (l \mapsto \circ) \mid \mathcal{S}}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi((\theta, \tau), \phi, \phi) = (\theta, \tau)$, then in the compositional LTS we have

that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \overset{\tau}{\hookrightarrow} e'}{\langle \theta, e\rangle \xrightarrow{(\theta,\tau)} \langle \theta, e'\rangle}}{\langle \theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\tau)} \langle \theta, e'\rangle \mid \mathcal{T}} \qquad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \qquad \langle((\theta,\tau),\phi,\phi)\rangle \in \mathcal{R}}{\langle \theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\tau),\phi,\phi)} \langle \theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

in the monolithic LTS we have the following configuration $\langle \theta, e\rangle \mid \mathcal{T} \mid \mathcal{M} \mid \mathcal{S}$ obtained by $\mathcal{RL}$, and we consider the label (θ, τ). Then, we obtain:

$$\textsc{Par}\ \frac{\textsc{Thread Step}\ \dfrac{e \overset{\tau}{\hookrightarrow} e'}{\langle \theta, e\rangle \xrightarrow{(\theta,\tau)} \langle \theta, e'\rangle}}{\langle \theta, e\rangle \mid \mathcal{T} \mid \mathcal{M} \mid \mathcal{S} \xrightarrow{(\theta,\tau)} \langle \theta, e'\rangle \mid \mathcal{T} \mid \mathcal{M} \mid \mathcal{S}}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi((\theta, \mathsf{sp}(\theta')), \phi, \phi) = (\theta, \mathsf{sp}(\theta'))$, then in the compositional LTS we have that:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \overset{\mathsf{sp}\ e''}{\hookrightarrow} e'; e''}{\langle \theta, e\rangle \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \langle \theta, e'\rangle \mid \langle \theta', e''\rangle}}{\langle \theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \langle \theta, e'\rangle \mid \langle \theta', e''\rangle \mid \mathcal{T}} \qquad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \qquad \langle((\theta,\mathsf{sp}(\theta')),\phi,\phi)\rangle \in \mathcal{R}}{\langle \theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\mathsf{sp}(\theta')),\phi,\phi)} \langle \theta, e'\rangle \mid \langle \theta', e''\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

in the monolithic LTS we have the following configuration $\langle \theta, e\rangle \mid \mathcal{T} \mid \mathcal{M} \mid \mathcal{S}$ obtained by $\mathcal{RL}$, and we consider the label $(\theta, \mathsf{sp}(\theta'))$. Then, we obtain:

$$\textsc{Par}\ \frac{\textsc{Thread Spawn}\ \dfrac{e \overset{\mathsf{sp}\ e''}{\hookrightarrow} e'; e''}{\langle \theta, e\rangle \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \langle \theta, e'\rangle \mid \langle \theta', e''\rangle}}{\langle \theta, e\rangle \mid \mathcal{T} \mid \mathcal{M} \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \langle \theta, e'\rangle \mid \langle \theta', e''\rangle \mid \mathcal{T} \mid \mathcal{M} \mid \mathcal{S}}$$

we can easily see that the final states are still related.

Now we consider the case starting from the monolithic LTS and we consider then $\mathcal{RL}^{-1}$ and φ^{-1}. We proceed by the case for each possible labels:

- We consider the case where $\varphi^{-1}() = ((\theta, \mathsf{ref}(r, v)), (\theta, \mathsf{ref}(r, v)), \phi)$, then in monolithic LTS

we have that:

$$\textsc{Par}\ \frac{\textsc{Alloc}\ \dfrac{e \xhookrightarrow{(\theta,\mathsf{ref}(r,v))} e'}{\langle\theta, e\rangle \mid (r \mapsto \circ) \xrightarrow{(\theta,\mathsf{ref}(r,v))} \langle\theta, e'\rangle \mid (r \mapsto v)}}{\langle\theta, e\rangle \mid \mathcal{T} \mid (r \mapsto \circ) \mid \mathcal{M} \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{ref}(r,v))} \langle\theta, e'\rangle \mid \mathcal{T} \mid (r \mapsto v) \mid \mathcal{M} \mid \mathcal{S}}$$

in the compositional LTS we have the following configuration $\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto \circ) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}$ obtained by $\mathcal{R}\mathcal{L}^{-1}$, and we consider the label $((\theta, \mathsf{ref}(r, v)), (\theta, \mathsf{ref}(r, v)), \phi)$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{ref}(r,v))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{ref}(r,v))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{ref}(r,v))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \dfrac{\dfrac{}{(r \mapsto \circ) \xrightarrow{(\theta,\mathsf{ref}(r,v))} (r \mapsto v)}}{(r \mapsto \circ) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{ref}(r,v))} (r \mapsto v) \mid \mathcal{M}} \quad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \quad \langle((\theta, \mathsf{ref}(r, v)), (\theta, \mathsf{ref}(r, v)), \phi)\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto \circ) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\mathsf{ref}(r,v)),(\theta,\mathsf{ref}(r,v)),\phi)} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi^{-1}(\theta, \mathsf{rd}(r, v)) = ((\theta, \mathsf{rd}(r, v)), (\theta, \mathsf{rd}(r, v)), \phi)$, then in monolithic LTS we have that:

$$\textsc{Par}\ \frac{\textsc{Read}\ \dfrac{e \xhookrightarrow{(\theta,\mathsf{rd}(r,v))} e'}{\langle\theta, e\rangle \mid (r \mapsto v) \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle\theta, e'\rangle \mid (r \mapsto v)}}{\langle\theta, e\rangle \mid \mathcal{T} \mid (r \mapsto v) \mid \mathcal{M} \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle\theta, e'\rangle \mid \mathcal{T} \mid (r \mapsto v) \mid \mathcal{M} \mid \mathcal{S}}$$

in the compositional LTS we have the following configuration $\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}$ obtained by $\mathcal{R}\mathcal{L}^{-1}$, and we consider the label $((\theta, \mathsf{rd}(r, v)), (\theta, \mathsf{rd}(r, v)), \phi)$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{rd}(r,v))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{rd}(r,v))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \dfrac{\dfrac{}{(r \mapsto v) \xrightarrow{(\theta,\mathsf{rd}(r,v))} (r \mapsto v)}}{(r \mapsto v) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{rd}(r,v))} (r \mapsto v) \mid \mathcal{M}} \quad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \quad \langle((\theta, \mathsf{rd}(r, v)), (\theta, \mathsf{rd}(r, v)), \phi)\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\mathsf{rd}(r,v)),(\theta,\mathsf{rd}(r,v)),\phi)} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi^{-1}(\theta, \mathsf{wr}(r, v)) = ((\theta, \mathsf{wr}(r, v)), (\theta, \mathsf{wr}(r, v)), \phi)$, then in

monolithic LTS we have that:

$$\textsc{Par}\ \frac{\textsc{Write}\ \dfrac{e \xhookrightarrow{(\theta,\mathsf{wr}(r,v))} e'}{\langle\theta,e\rangle \mid (r \mapsto v_{old}) \xrightarrow{(\theta,\mathsf{wr}(r,v))} \langle\theta,e'\rangle \mid (r \mapsto v)}}{\langle\theta,e\rangle \mid \mathcal{T} \mid (r \mapsto v_{old}) \mid \mathcal{M} \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{wr}(r,v))} \langle\theta,e'\rangle \mid \mathcal{T} \mid (r \mapsto v) \mid \mathcal{M} \mid \mathcal{S}}$$

in the compositional LTS we have the following configuration $\langle\theta,e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v_{old}) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}$ obtained by $\mathcal{R}\mathcal{L}^{-1}$, and we consider the label $((\theta,\mathsf{wr}(r,v)),(\theta,\mathsf{wr}(r,v)),\phi)$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{wr}(r,v))} e'}{\langle\theta,e\rangle \xrightarrow{(\theta,\mathsf{wr}(r,v))} \langle\theta,e'\rangle} \quad \dfrac{}{(r \mapsto v_{old}) \xrightarrow{(\theta,\mathsf{wr}(r,v))} (r \mapsto v)}}{\begin{array}{c}\langle\theta,e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{wr}(r,v))} \langle\theta,e'\rangle \mid \mathcal{T} \quad (r \mapsto v_{old}) \mid \mathcal{M} \xrightarrow{(\theta,\mathsf{wr}(r,v))} (r \mapsto v) \mid \mathcal{M} \\ \mathcal{S} \xrightarrow{\phi} \mathcal{S} \qquad \langle((\theta,\mathsf{wr}(r,v)),(\theta,\mathsf{wr}(r,v)),\phi)\rangle \in \mathcal{R}\end{array}}{\langle\theta,e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v_{old}) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\mathsf{wr}(r,v)),(\theta,\mathsf{wr}(r,v)),\phi)} \langle\theta,e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} (r \mapsto v) \mid \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi^{-1}(\theta,\mathsf{acq}(l)) = ((\theta,\mathsf{acq}(l)),\phi,(\theta,\mathsf{acq}(l)))$, then in monolithic LTS we have that:

$$\textsc{Par}\ \frac{\textsc{Acquire}\ \dfrac{e \xhookrightarrow{(\theta,\mathsf{acq}(l))} e'}{\langle\theta,e\rangle \mid (l \mapsto \circ) \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta,e'\rangle \mid (l \mapsto \theta)}}{\langle\theta,e\rangle \mid \mathcal{T} \mid \mathcal{M} \mid (l \mapsto \circ) \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta,e'\rangle \mid \mathcal{T} \mid \mathcal{M} \mid (l \mapsto \theta) \mid \mathcal{S}}$$

in the compositional LTS we have the following configuration $\langle\theta,e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \circ) \mid \mathcal{S}$ obtained by $\mathcal{R}\mathcal{L}^{-1}$, and we consider the label $((\theta,\mathsf{acq}(l)),\phi,(\theta,\mathsf{acq}(l)))$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{acq}(l))} e'}{\langle\theta,e\rangle \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta,e'\rangle} \quad \dfrac{}{(l \mapsto \circ) \xrightarrow{(\theta,\mathsf{acq}(l))} (l \mapsto \theta)}}{\begin{array}{c}\langle\theta,e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{acq}(l))} \langle\theta,e'\rangle \mid \mathcal{T} \quad (l \mapsto \circ) \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{acq}(l))} (l \mapsto \theta) \mid \mathcal{S} \\ \mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad \langle((\theta,\mathsf{acq}(l)),\phi,(\theta,\mathsf{acq}(l)))\rangle \in \mathcal{R}\end{array}}{\langle\theta,e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \circ) \mid \mathcal{S} \xrightarrow{((\theta,\mathsf{acq}(l)),\phi,(\theta,\mathsf{acq}(l)))} \langle\theta,e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \theta) \mid \mathcal{S}}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi^{-1}(\theta,\mathsf{rel}(l)) = ((\theta,\mathsf{rel}(l)),\phi,(\theta,\mathsf{rel}(l)))$, then in monolithic

LTS we have that:

$$\textsc{Par}\ \frac{\textsc{Release}\ \dfrac{e \xhookrightarrow{(\theta,\mathsf{rel}(l))} e'}{\langle\theta, e\rangle \mid (l \mapsto \theta) \xrightarrow{(\theta,\mathsf{rel}(l))} \langle\theta, e'\rangle \mid (l \mapsto \circ)}}{\langle\theta, e\rangle \mid \mathcal{T} \mid \mathcal{M} \mid (l \mapsto \theta) \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{rel}(l))} \langle\theta, e'\rangle \mid \mathcal{T} \mid \mathcal{M} \mid (l \mapsto \circ) \mid \mathcal{S}}$$

in the compositional LTS we have the following configuration $\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \theta) \mid \mathcal{S}$ obtained by $\mathcal{R}\mathcal{L}^{-1}$, and we consider the label $((\theta, \mathsf{rel}(l)), \phi, (\theta, \mathsf{rel}(l)))$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{(\theta,\mathsf{rel}(l))} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{rel}(l))} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{rel}(l))} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \dfrac{}{(l \mapsto \theta) \xrightarrow{(\theta,\mathsf{rel}(l))} (l \mapsto \circ)}}{(l \mapsto \theta) \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{rel}(l))} (l \mapsto \circ) \mid \mathcal{S}} \quad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \quad \langle((\theta, \mathsf{rel}(l)), \phi, (\theta, \mathsf{rel}(l)))\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \theta) \mid \mathcal{S} \xrightarrow{((\theta,\mathsf{rel}(l)),\phi,(\theta,\mathsf{rel}(l)))} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} (l \mapsto \circ) \mid \mathcal{S}}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi^{-1}(\theta, \tau) = ((\theta, \tau), \phi, \phi)$, then in monolithic LTS we have that:

$$\textsc{Par}\ \frac{\textsc{Thread Step}\ \dfrac{e \xhookrightarrow{\tau} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\tau)} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \mid \mathcal{M} \mid \mathcal{S} \xrightarrow{(\theta,\tau)} \langle\theta, e'\rangle \mid \mathcal{T} \mid \mathcal{M} \mid \mathcal{S}}$$

in the compositional LTS we have the following configuration $\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}$ obtained by $\mathcal{R}\mathcal{L}^{-1}$, and we consider the label $((\theta, \tau), \phi, \phi)$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xhookrightarrow{\tau} e'}{\langle\theta, e\rangle \xrightarrow{(\theta,\tau)} \langle\theta, e'\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\tau)} \langle\theta, e'\rangle \mid \mathcal{T}} \quad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \quad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \quad \langle((\theta, \tau), \phi, \phi)\rangle \in \mathcal{R}}{\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\tau),\phi,\phi)} \langle\theta, e'\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

we can easily see that the final states are still related.

- We consider the case where $\varphi^{-1}(\theta, \mathsf{sp}(\theta')) = ((\theta, \mathsf{sp}(\theta')), \phi, \phi)$, then in monolithic LTS we have that:

$$\textsc{Par}\ \frac{\textsc{Thread Spawn}\ \dfrac{e \xhookrightarrow{\mathsf{sp}\ e''} e'; e''}{\langle\theta, e\rangle \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \langle\theta, e'\rangle \mid \langle\theta', e''\rangle}}{\langle\theta, e\rangle \mid \mathcal{T} \mid \mathcal{M} \mid \mathcal{S} \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \langle\theta, e'\rangle \mid \langle\theta', e''\rangle \mid \mathcal{T} \mid \mathcal{M} \mid \mathcal{S}}$$

in the compositional LTS we have the following configuration $\langle\theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}$ obtained

by $\mathcal{RL}^{-1}$, and we consider the label $((\theta, \mathsf{sp}(\theta')), \phi, \phi)$. Then, we obtain:

$$\textsc{Sync}\ \frac{\dfrac{\dfrac{e \xrightarrow{\mathsf{sp}\ e''} e'; e''}{\langle \theta, e\rangle \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \langle \theta, e'\rangle \mid \langle \theta', e''\rangle}}{\langle \theta, e\rangle \mid \mathcal{T} \xrightarrow{(\theta,\mathsf{sp}(\theta'))} \langle \theta, e'\rangle \mid \langle \theta', e''\rangle \mid \mathcal{T}} \qquad \mathcal{M} \xrightarrow{\phi} \mathcal{M} \qquad \mathcal{S} \xrightarrow{\phi} \mathcal{S} \qquad \langle((\theta, \mathsf{sp}(\theta')), \phi, \phi)\rangle \in \mathcal{R}}{\langle \theta, e\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S} \xrightarrow{((\theta,\mathsf{sp}(\theta')),\phi,\phi)} \langle \theta, e'\rangle \mid \langle \theta', e''\rangle \mid \mathcal{T} \otimes_{\mathcal{R}} \mathcal{M} \otimes_{\mathcal{R}} \mathcal{S}}$$

we can easily see that the final states are still related.

□

C.2 Reversible Monolithic LTS

Here we show the rules of the reversible monolithic LTS obtained with the approach described in [33, 44]. In Fig. C.1 we show the forward rules, while Fig. C.2 depicts the backward rules.

$$\textsc{Alloc}\ \frac{e \xrightarrow{(\theta,\mathsf{ref}(r,v))} e' \qquad k_1', k_2'\ \ fresh}{k_1 : \langle \theta, e\rangle \mid k_2 : (r \mapsto \circ) \xrightarrow{(\theta,\mathsf{ref}(r,v))} k_1' : \langle \theta, e'\rangle \mid k_2' : (r \mapsto v) \mid [k_1 : \langle \theta, e\rangle \mid k_2 : (r \mapsto \circ); k_1' : \bullet_1 \mid k_2' : \bullet_2]}$$

$$\textsc{Read}\ \frac{e \xrightarrow{(\theta,\mathsf{rd}(r,v))} e' \qquad k_1', k_2'\ \ fresh}{k_1 : \langle \theta, e\rangle \mid k_2 : (r \mapsto v) \xrightarrow{(\theta,\mathsf{rd}(r,v))} k_1' : \langle \theta, e'\rangle \mid k_2' : (r \mapsto v) \mid [k_1 : \langle \theta, e\rangle \mid k_2 : (r \mapsto v); k_1' : \bullet_1 \mid k_2' : \bullet_2]}$$

$$\textsc{Write}\ \frac{e \xrightarrow{(\theta,\mathsf{wr}(r,v))} e' \qquad k_1', k_2'\ \ fresh}{k_1 : \langle \theta, e\rangle \mid k_2 : (r \mapsto v_{old}) \xrightarrow{(\theta,\mathsf{wr}(r,v))} k_1' : \langle \theta, e'\rangle \mid k_2' : (r \mapsto v) \mid [k_1 : \langle \theta, e\rangle \mid k_2 : (r \mapsto v_{old}); k_1' : \bullet_1 \mid k_2' : \bullet_2]}$$

$$\textsc{Acquire}\ \frac{e \xrightarrow{(\theta,\mathsf{acq}(l))} e' \qquad k_1', k_2'\ \ fresh}{k_1 : \langle \theta, e\rangle \mid k_2 : (l \mapsto \circ) \xrightarrow{(\theta,\mathsf{acq}(l))} k_1' : \langle \theta, e'\rangle \mid k_2' : (l \mapsto \theta) \mid [k_1 : \langle \theta, e\rangle \mid k_2 : (l \mapsto \circ); k_1' : \bullet_1 \mid k_2' : \bullet_2]}$$

$$\textsc{Release}\ \frac{e \xrightarrow{(\theta,\mathsf{rel}(l))} e' \qquad k_1', k_2'\ \ fresh}{k_1 : \langle \theta, e\rangle \mid k_2 : (l \mapsto \theta) \xrightarrow{(\theta,\mathsf{rel}(l))} k_1' : \langle \theta, e'\rangle \mid k_2' : (l \mapsto \circ) \mid [k_1 : \langle \theta, e\rangle \mid k_2 : (l \mapsto \theta); k_1' : \bullet_1 \mid k_2' : \bullet_2]}$$

$$\textsc{Thread Step}\ \frac{e \xrightarrow{(\theta,\alpha)} e' \qquad k_1'\ \ fresh}{k_1 : \langle \theta, e\rangle \xrightarrow{(\theta,\alpha)} k_1' : \langle \theta, e'\rangle \mid [k_1 : \langle \theta, e\rangle; k_1' : \bullet_1]}\ \alpha = \mathsf{pure}$$

$$\textsc{Thread Spawn}\ \frac{e \xrightarrow{(\theta,\mathsf{sp}\ e'')} e'; e'' \qquad \theta'\ \ \mathsf{fresh} \qquad k_1', k_2'\ \ fresh}{k_1 : \langle \theta, e\rangle \xrightarrow{(\theta,\mathsf{sp}(\theta'))} k_1' : \langle \theta, e'\rangle \mid k_2' : \langle \theta', e''\rangle \mid [k_1 : \langle \theta, e\rangle; k_1' : \bullet_1 \mid k_2' : \bullet_2]}$$

Figure C.1: Forward reversible rules of the monolithic LTS

$$\frac{}{k_1' : \langle \theta, e' \rangle \mid k_2' : (r \mapsto v) \mid [k_1 : \langle \theta, e \rangle \mid k_2 : (r \mapsto \circ); k_1' : \bullet_1 \mid k_2' : \bullet_2] \xrightarrow{(\theta, \mathsf{ref}(r,v))} k_1 : \langle \theta, e \rangle \mid k_2 : (r \mapsto \circ)} \textsc{Alloc}$$

$$\frac{}{k_1' : \langle \theta, e' \rangle \mid k_2' : (r \mapsto v) \mid [k_1 : \langle \theta, e \rangle \mid k_2 : (r \mapsto v); k_1' : \bullet_1 \mid k_2' : \bullet_2] \xrightarrow{(\theta, \mathsf{rd}(r,v))} k_1 : \langle \theta, e \rangle \mid k_2 : (r \mapsto v)} \textsc{Read}$$

$$\frac{}{k_1' : \langle \theta, e' \rangle \mid k_2' : (r \mapsto v) \mid [k_1 : \langle \theta, e \rangle \mid k_2 : (r \mapsto v_{old}); k_1' : \bullet_1 \mid k_2' : \bullet_2] \xrightarrow{(\theta, \mathsf{wr}(r,v))} k_1 : \langle \theta, e \rangle \mid k_2 : (r \mapsto v_{old})} \textsc{Write}$$

$$\frac{}{k_1' : \langle \theta, e' \rangle \mid k_2' : (l \mapsto \theta) \mid [k_1 : \langle \theta, e \rangle \mid k_2 : (l \mapsto \circ); k_1' : \bullet_1 \mid k_2' : \bullet_2] \xrightarrow{(\theta, \mathsf{acq}(l))} k_1 : \langle \theta, e \rangle \mid k_2 : (l \mapsto \circ)} \textsc{Acquire}$$

$$\frac{}{k_1' : \langle \theta, e' \rangle \mid k_2' : (l \mapsto \circ) \mid [k_1 : \langle \theta, e \rangle \mid k_2 : (l \mapsto \theta); k_1' : \bullet_1 \mid k_2' : \bullet_2] \xrightarrow{(\theta, \mathsf{rel}(l))} k_1 : \langle \theta, e \rangle \mid k_2 : (l \mapsto \theta)} \textsc{Release}$$

$$\frac{}{k_1' : \langle \theta, e' \rangle \mid [k_1 : \langle \theta, e \rangle; k_1' : \bullet_1] \xrightarrow{(\theta, \alpha)} k_1 : \langle \theta, e \rangle \ \alpha = \mathsf{pure}} \textsc{Thread Step}$$

$$\frac{}{k_1' : \langle \theta, e' \rangle \mid k_2' : \langle \theta', e'' \rangle \mid [k_1 : \langle \theta, e \rangle; k_1' : \bullet_1 \mid k_2' : \bullet_2] \xrightarrow{(\theta, \mathsf{sp}(\theta'))} k_1 : \langle \theta, e \rangle} \textsc{Thread Spawn}$$

Figure C.2: Backward reversible rules of the monolithic LTS

ALLOC

$$\frac{e \xrightarrow{(\theta,\mathsf{ref}(r,v))} e' \qquad k_1', k_2' \;\; fresh}{k_1 : \langle \theta, e\rangle \mid k_2 : (r \mapsto \circ) \xrightarrow{[k_1:\langle\theta,e\rangle \mid k_2:(r\mapsto\circ); k_1':\bullet_1 \mid k_2':\bullet_2]} k_1' : \langle \theta, e'\rangle \mid k_2' : (r \mapsto v) \mid [k_1 : \langle \theta, e\rangle \mid k_2 : (r \mapsto \circ); k_1' : \bullet_1 \mid k_2' : \bullet_2]}$$

READ

$$\frac{e \xrightarrow{(\theta,\mathsf{rd}(r,v))} e' \qquad k_1', k_2' \;\; fresh}{k_1 : \langle \theta, e\rangle \mid k_2 : (r \mapsto v) \xrightarrow{[k_1:\langle\theta,e\rangle \mid k_2:(r\mapsto v); k_1':\bullet_1 \mid k_2':\bullet_2]} k_1' : \langle \theta, e'\rangle \mid k_2' : (r \mapsto v) \mid [k_1 : \langle \theta, e\rangle \mid k_2 : (r \mapsto v); k_1' : \bullet_1 \mid k_2' : \bullet_2]}$$

WRITE

$$\frac{e \xrightarrow{(\theta,\mathsf{wr}(r,v))} e' \qquad k_1', k_2' \;\; fresh}{k_1 : \langle \theta, e\rangle \mid k_2 : (r \mapsto v_{old}) \xrightarrow{[k_1:\langle\theta,e\rangle \mid k_2:(r\mapsto v_{old}); k_1':\bullet_1 \mid k_2':\bullet_2]} k_1' : \langle \theta, e'\rangle \mid k_2' : (r \mapsto v) \mid [k_1 : \langle \theta, e\rangle \mid k_2 : (r \mapsto v_{old}); k_1' : \bullet_1 \mid k_2' : \bullet_2]}$$

ACQUIRE

$$\frac{e \xrightarrow{(\theta,\mathsf{acq}(l))} e' \qquad k_1', k_2' \;\; fresh}{k_1 : \langle \theta, e\rangle \mid k_2 : (l \mapsto \circ) \xrightarrow{[k_1:\langle\theta,e\rangle \mid k_2:(l\mapsto\circ); k_1':\bullet_1 \mid k_2':\bullet_2]} k_1' : \langle \theta, e'\rangle \mid k_2' : (l \mapsto \theta) \mid [k_1 : \langle \theta, e\rangle \mid k_2 : (l \mapsto \circ); k_1' : \bullet_1 \mid k_2' : \bullet_2]}$$

RELEASE

$$\frac{e \xrightarrow{(\theta,\mathsf{rel}(l))} e' \qquad k_1', k_2' \;\; fresh}{k_1 : \langle \theta, e\rangle \mid k_2 : (l \mapsto \theta) \xrightarrow{[k_1:\langle\theta,e\rangle \mid k_2:(l\mapsto\theta); k_1':\bullet_1 \mid k_2':\bullet_2]} k_1' : \langle \theta, e'\rangle \mid k_2' : (l \mapsto \circ) \mid [k_1 : \langle \theta, e\rangle \mid k_2 : (l \mapsto \theta); k_1' : \bullet_1 \mid k_2' : \bullet_2]}$$

THREAD STEP

$$\frac{e \xrightarrow{(\theta,\alpha)} e' \qquad k_1' \;\; fresh}{k_1 : \langle \theta, e\rangle \xrightarrow{[k_1:\langle\theta,e\rangle; k_1':\bullet_1]} k_1' : \langle \theta, e'\rangle \mid [k_1 : \langle \theta, e\rangle; k_1' : \bullet_1]} \; \alpha = \mathsf{pure}$$

THREAD SPAWN

$$\frac{e \xrightarrow{(\theta,\mathsf{sp}\ e'')} e'; e'' \qquad \theta' \;\; \mathsf{fresh} \qquad k_1', k_2' \;\; fresh}{k_1 : \langle \theta, e\rangle \xrightarrow{[k_1:\langle\theta,e\rangle; k_1':\bullet_1 \mid k_2':\bullet_2]} k_1' : \langle \theta, e'\rangle \mid k_2' : \langle \theta', e''\rangle \mid [k_1 : \langle \theta, e\rangle; k_1' : \bullet_1 \mid k_2' : \bullet_2]}$$

Figure C.3: Forward reversible rules of the monolithic LTS with memories

$$\frac{}{k_1' : \langle \theta, e' \rangle \mid k_2' : (r \mapsto v) \mid [k_1 : \langle \theta, e \rangle \mid k_2 : (r \mapsto \circ); k_1' : \bullet_1 \mid k_2' : \bullet_2] \xrightarrow{[k_1:\langle\theta,e\rangle \mid k_2:(r\mapsto\circ); k_1':\bullet_1 \mid k_2':\bullet_2]} k_1 : \langle \theta, e \rangle \mid k_2 : (r \mapsto \circ)} \text{ALLOC}$$

$$\frac{}{k_1' : \langle \theta, e' \rangle \mid k_2' : (r \mapsto v) \mid [k_1 : \langle \theta, e \rangle \mid k_2 : (r \mapsto v); k_1' : \bullet_1 \mid k_2' : \bullet_2] \xrightarrow{[k_1:\langle\theta,e\rangle \mid k_2:(r\mapsto v); k_1':\bullet_1 \mid k_2':\bullet_2]} k_1 : \langle \theta, e \rangle \mid k_2 : (r \mapsto v)} \text{READ}$$

$$\frac{}{k_1' : \langle \theta, e' \rangle \mid k_2' : (r \mapsto v) \mid [k_1 : \langle \theta, e \rangle \mid k_2 : (r \mapsto v_{old}); k_1' : \bullet_1 \mid k_2' : \bullet_2] \xrightarrow{[k_1:\langle\theta,e\rangle \mid k_2:(r\mapsto v_{old}); k_1':\bullet_1 \mid k_2':\bullet_2]} k_1 : \langle \theta, e \rangle \mid k_2 : (r \mapsto v_{old})} \text{WRITE}$$

$$\frac{}{k_1' : \langle \theta, e' \rangle \mid k_2' : (l \mapsto \theta) \mid [k_1 : \langle \theta, e \rangle \mid k_2 : (l \mapsto \circ); k_1' : \bullet_1 \mid k_2' : \bullet_2] \xrightarrow{[k_1:\langle\theta,e\rangle \mid k_2:(l\mapsto\circ); k_1':\bullet_1 \mid k_2':\bullet_2]} k_1 : \langle \theta, e \rangle \mid k_2 : (l \mapsto \circ)} \text{ACQUIRE}$$

$$\frac{}{k_1' : \langle \theta, e' \rangle \mid k_2' : (l \mapsto \circ) \mid [k_1 : \langle \theta, e \rangle \mid k_2 : (l \mapsto \theta); k_1' : \bullet_1 \mid k_2' : \bullet_2] \xrightarrow{[k_1:\langle\theta,e\rangle \mid k_2:(l\mapsto\theta); k_1':\bullet_1 \mid k_2':\bullet_2]} k_1 : \langle \theta, e \rangle \mid k_2 : (l \mapsto \theta)} \text{RELEASE}$$

$$\frac{}{k_1' : \langle \theta, e' \rangle \mid [k_1 : \langle \theta, e \rangle; k_1' : \bullet_1] \xrightarrow{[k_1:\langle\theta,e\rangle; k_1':\bullet_1]} k_1 : \langle \theta, e \rangle \ \alpha = \mathsf{pure}} \text{THREAD STEP}$$

$$\frac{}{k_1' : \langle \theta, e' \rangle \mid k_2' : \langle \theta', e'' \rangle \mid [k_1 : \langle \theta, e \rangle; k_1' : \bullet_1 \mid k_2' : \bullet_2] \xrightarrow{[k_1:\langle\theta,e\rangle; k_1':\bullet_1 \mid k_2':\bullet_2]} k_1 : \langle \theta, e \rangle} \text{THREAD SPAWN}$$

Figure C.4: Backward reversible rules of the monolithic LTS with memories