Compositional Information Flow Security for Concurrent Programs*

Annalisa Bossi[1] and Carla Piazza[2] and Sabina Rossi[1]†

[1]Dipartimento di Informatica, Università Ca' Foscari di Venezia
via Torino 155, 30172 Venezia, Italy
{bossi,srossi}@dsi.unive.it

[2]Dipartimento di Matematica ed Informatica, Università di Udine
via Le Scienze 206, 33100 Udine, Italy
piazza@dimi.uniud.it

Abstract

We present a general unwinding framework for the definition of information flow security properties of concurrent programs, described in a simple imperative language enriched with parallelism and atomic statement constructors. We study different classes of programs obtained by instantiating the general framework and we prove that they entail the noninterference principle. Accurate proof techniques for the verification of such properties are defined by exploiting the Tarski decidability result for first-order formulae over the reals. Moreover, we illustrate how the unwinding framework can be instantiated in order to deal with intentional information release and we extend our verification techniques to the analysis of security properties of programs admitting downgrading.

Keywords: Security, Noninterference, Concurrency, Bisimulation.

1 Introduction

The protection of confi dential data in computing systems has long been recognized as a diffi cult and daunting problem. In order to guarantee the confi dentiality of sensitive data it is necessary to analyze how information flows so that secrets are not transmitted

*Work partially supported by the MIUR project "Fondamenti Logici dei Sistemi Distribuiti e Codice Mobile" (grant 2005015785), the FIRB project "Interpretazione Astratta e Model Checking per la Verifi ca di Sistemi Embedded" (grant RBAU018RCZ), the MIUR Project PRIN 2005 project 2005015491, and the FVG regional project BioCheck.

†Corresponding author.

to unauthorized parties. A common way to control information flow is to associate a security level with information in the system, and to prevent higher level (more confi dential) information from affecting lower level (less confi dential) information. Recently, there has been much work applying this approach in a language-based setting. We refer the reader to [28] for a clear and wide overview about the different approaches. All of these proposals accomplish the *noninterference* principle [13] which requires that secret input data cannot be inferred through the observation of non confi dential outputs. Among the approaches based on formal methods, noninterference has been formalized in terms of behavioural equivalences, e.g., [12, 26], and ensured through type-systems, e.g., [27, 28, 37, 38], and logical formulations, e.g., [2, 4].

In this paper we face the problem of specifying and verifying noninterference properties for concurrent programs, described in a simple imperative language admitting parallel executions on a shared memory and atomic statement constructors. We consider two confi dentiality levels, a public level and a confi dential one, and associate a level to each location (variable) of the language.

We start from the observation that, in order to be of practical usefulness, the method used to check a property should be compositional with respect to the language operators. In particular, in the context of concurrent programs, it would be desirable to have properties which are compositional with respect to the parallel operator. In our previous works (see [5] for an overview) we showed how compositional information flow security properties for the Security Process Algebra (SPA) language [12] can be naturally characterized in terms of *unwinding conditions* [14]. In this paper we investigate how to instantiate the unwinding based framework for the defi nition of noninterference properties of concurrent programs.

Unwinding conditions have been used to express security properties of processes described through, e.g, event systems or labelled transition systems [14, 18, 24]. They demand properties of individual actions and are easier to handle with respect to global conditions. Intuitively, an unwinding condition requires that each high level (confi dential) transition is simulated in such a way that a low level observer cannot infer whether such high level action has been performed or not. Thus the low level observation of the process is not influenced in any way by its high behaviour.

Following this idea, in Section 3 we defi ne a generalized unwinding condition for our simple programming language. We study different classes of programs obtained by instantiating the unwinding framework through different notions of low level bisimulation. In particular, we study security properties which entail the noninterference principle.

The problem of verifi cation is tracked in Section 4. We focus on one instance of our unwinding condition which is compositional with respect to the language constructors. We defi ne accurate proof methods for the verifi cation of such property which are more precise than previous type-based techniques such as those presented in [1, 7, 27, 34]. Indeed, in our language, insecure flows can be explicit, e.g., when assigning the value of a high variable to a low variable, or implicit, e.g., when testing the value of a high variable and then assigning to a low variable a value depending on the result of the test. In most of previous approaches explicit flows are prevented by asking that only low level expressions, i.e., not containing high level variables, are assigned to low variables, while implicit flows are prevented by requiring that the boolean expressions of while-

loops and conditionals contain high level variables only under strict restrictions (see, e.g., [11, 6, 37, 34, 33, 28]). Thus, for instance, if H is a high level variable while L is a low level one, the commands

$$L := H \tag{1}$$

$$L := H - H \tag{2}$$

$$\texttt{while}(L + H > H)\ \{L := 1\} \tag{3}$$

$$\texttt{if}(H = 0)\ \{L := 1\}\ \texttt{else}\ \{L := 1\} \tag{4}$$

are deemed insecure by the type systems mentioned above. Instead, our proof techniques exploit the Tarski decidability result for first-order formulae over the reals and allow us to establish that, e.g., commands (2), (3) and (4) are secure.

The security properties studied in Sections 3 and 4 entail the standard noninterference principle. However, as already noticed by many authors, e.g., [17, 29, 41], noninterference is too strong for practical applications. Indeed, many realistic systems do release some confidential information as part of their intended function. For example, a password checker leaks a small amount of information when a user attempts to log in by inserting his password, since an external observer can learn whether the password has been guessed or not. Such a program violates the noninterference principle and would be rejected by our verification systems. In Section 5 we extend our approach to programs which intentionally release some information. We first illustrate how the unwinding framework can be instantiated in order to deal with intentional information release. In particular, we model a security property which can be viewed as the compositional version of the *delimited release* property defined in [29]. We also extend our verification techniques to the analysis of security properties of programs admitting downgrading.

The paper is organized as follows. In Section 2 we introduce the language together with its syntax and semantics. In Section 3 we define a general unwinding schema for our imperative language and study different instantiations of it. We also prove a soundness theorem with respect to the standard noninterference property. In Section 4 we show that the noninterference properties of previous section are undecidable and define two different proof methods which exploit the Tarski's decidability result for first-order formulae over the reals to gain precision. In Section 5 we show how our unwinding condition can be instantiated in order to model security properties of programs in which there is an intentional release of information. We also extend our proof techniques to deal with such properties. Finally, in Section 6 we discuss related work and draw some conclusions.

2 The Language: Syntax and Semantics

The language we consider in this paper is inspired by the one for concurrent programming introduced by Andrews in [3]. It is an imperative language with a parallel operator and an atomic block constructor, called `await`. As described in [3], such a language is suitable to deal with standard problems of concurrent programming (e.g., mutual

exclusion) and to implement semaphores and monitors. Here we assume that the locations (variables) are partitioned into two levels: a public level and a confidential one. Intuitively, the values contained in the confidential locations are accessible only to authorized users (high level users), while the values in the public locations are available to all the users. The security properties we are going to study aim at detecting any flow of information from high level to low level locations, i.e., a variation of confidential (high level) inputs should not cause a variation of values in the low level locations.

The operational semantics of our language is expressed in terms of *labelled transition systems*, i.e., graphs with labels on the edges and on the nodes. The labels on the nodes correspond to the states of the locations, while the labels on the edges denote the level (high or low) of the transitions.

Let $\mathbb{Z}$ be the set of integer numbers, $\mathbb{T} = \{\text{true}, \text{false}\}$ be the set of boolean values, $\mathbb{L}$ be a set of low level locations and $\mathbb{H}$ be a set of high level locations, with $\mathbb{L} \cap \mathbb{H} = \emptyset$. The set **Aexp** of arithmetic expressions is defined by the grammar:

$$a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 * a_1$$

where $n \in \mathbb{Z}$ and $X \in \mathbb{L} \cup \mathbb{H}$. We assume that arithmetic expressions are total. The set **Bexp** of boolean expressions is defined by:

$$b ::= \text{true} \mid \text{false} \mid (a_0 = a_1) \mid (a_0 \leq a_1) \mid \neg b \mid b_0 \wedge b_1 \mid b_0 \vee b_1$$

where $a_0, a_1 \in$ **Aexp**.

We say that an arithmetic expression a is *confidential*, denoted by $a \in \texttt{high}$, if there is a high level location which occurs in it. Otherwise we say that a is *public*, denoted by $a \in \texttt{low}$. Similarly, we say that a boolean expression b is *confidential*, denoted by $b \in \texttt{high}$, if there is a confidential arithmetic expression which occurs in it. Otherwise we say that b is *public*, denoted by $b \in \texttt{low}$. This notion of confidentiality, both for arithmetic and boolean expressions, is purely syntactic (as in [11]). Notice that a high level expression can contain low level locations, i.e., its value can depend on the values of low level locations. This reflects the idea that a high level user can read both high and low level data.

The set **Prog** of programs of our language is defined as:

$$\begin{array}{lcl} S & ::= & \texttt{skip} \mid X := a \mid S_0; S_1 \\ P & ::= & S \mid P_0; P_1 \mid \texttt{if}(b)\ \{P_0\}\ \texttt{else}\ \{P_1\} \mid \texttt{while}(b)\ \{P\} \mid \texttt{await}(b)\ \{S\} \mid \\ & & \texttt{co}\ P_1 \| \ldots \| P_n\ \texttt{oc} \end{array}$$

where $a \in$ **Aexp**, $X \in \mathbb{L} \cup \mathbb{H}$, and $b \in$ **Bexp**. Notice that, as in [3], in the body of the `await` operator only sequences of assignments are allowed.

The operational semantics of our language is based on the notion of *state*. A state σ is a function which assigns to each location an integer, i.e., $\sigma : \mathbb{L} \cup \mathbb{H} \longrightarrow \mathbb{Z}$. Given a state σ, we denote by $\sigma[X/n]$ the state σ' such that $\sigma'(X) = n$ and $\sigma'(Y) = \sigma(Y)$ for all $Y \neq X$. Moreover, we denote by σ_L the restriction of σ to the low level locations and we write $\sigma =_l \theta$ for $\sigma_L = \theta_L$.

Given an arithmetic expression $a \in$ **Aexp** and a state σ, the evaluation of a in σ, denoted by $\langle a, \sigma \rangle \rightarrow n$ with $n \in \mathbb{Z}$, is defined in the standard way (see, e.g., [40]).

Similarly, $\langle b,\sigma\rangle \to v$ with $b \in \mathbf{Bexp}$ and $v \in \{\text{true},\text{false}\}$, denotes the evaluation of a boolean expression b in a state σ (see, e.g., [40]). In both cases atomicity of the evaluation operation is assumed.

Our operational semantics is expressed in terms of state transitions. A transition from a program P and a state σ has the form $\langle P,\sigma\rangle \xrightarrow{\varepsilon} \langle P',\sigma'\rangle$ where P' is either a program or the special symbol `end` (denoting termination) and $\varepsilon \in \{\texttt{high},\texttt{low}\}$ stating that the transition is either confi dential or public. The operation $\varepsilon_1 \cup \varepsilon_2$ returns `low` if both ε_1 and ε_2 are `low` otherwise it returns `high`. Let $\mathbb{P} = \mathbf{Prog} \cup \{\texttt{end}\}$ and Σ be the set of all the possible states. The operational semantics of $\langle P,\sigma\rangle \in \mathbb{P}\times\Sigma$ is the *labelled transition system* (LTS) defi ned by structural induction on P according to the rules depicted in Table 1. Intuitively, the semantics of the sequential composition imposes that a program of the form $P_0;P_1$ behaves like P_0 until P_0 terminates and then it behaves like P_1. To describe the semantics of a program of the form $\texttt{while}(b)\ \{P\}$ we have to distinguish two cases: if b is true, then the program is unravelled to $P;\texttt{while}(b)\ \{P\}$; otherwise it terminates. As far as the `await` operator is concerned, if b is true then $\texttt{await}(b)\ \{P\}$ terminates executing P in one indivisible action (as an atomic block), i.e., it is not possible to observe the state changes internal to the execution of P, while if b is false then $\texttt{await}(b)\ \{P\}$ loops waiting for b to become true. Finally, in a parallel composition of the form $\texttt{co}\ P_0 \| \ldots \| P_n \texttt{oc}$ any of the P_i can move, and the termination is reached only when all the P_i's have terminated.

We use the following notations. We write $\langle P,\sigma\rangle \to \langle P',\sigma'\rangle$ to denote $\langle P,\sigma\rangle \xrightarrow{\varepsilon} \langle P',\sigma'\rangle$ with $\varepsilon \in \{\texttt{low},\texttt{high}\}$ and $\langle P_0,\sigma_0\rangle \to^n \langle P_n,\sigma_n\rangle$ with $n \geq 0$ for $\langle P_0,\sigma_0\rangle \to \langle P_1,\sigma_1\rangle \to \cdots \to \langle P_{n-1},\sigma_{n-1}\rangle \to \langle P_n,\sigma_n\rangle$. The notation $\langle P_0,\sigma_0\rangle \overset{\texttt{low}}{\rightsquigarrow} \langle P_n,\sigma_n\rangle$ stands for $\langle P_0,\sigma_0\rangle \to^n \langle P_n,\sigma_n\rangle$ for some $n \geq 0$ with all the n transitions labelled with `low`; similarly $\langle P_0,\sigma_0\rangle \overset{\texttt{high}}{\rightsquigarrow} \langle P_n,\sigma_n\rangle$ stands for $\langle P_0,\sigma_0\rangle \to^n \langle P_n,\sigma_n\rangle$ for some $n \geq 0$ with at least one of the n transitions labelled with `high`. Finally, we write $\langle P,\sigma\rangle \rightsquigarrow \langle P',\sigma'\rangle$ to denote $\langle P,\sigma\rangle \overset{\varepsilon}{\rightsquigarrow} \langle P',\sigma'\rangle$ with $\varepsilon \in \{\texttt{low},\texttt{high}\}$.

Let $\mathcal{R}$ be a binary relation over a set S and $s \in S$. We denote by $\mathcal{R}(s)$ the set $\{s' \in S \mid (s,s') \in \mathcal{R}\}$. We defi ne the relation *Reach* as the binary relation over $\mathbb{P}$ such that $(P,Q) \in \mathit{Reach}$ if and only if there exist σ and σ' in Σ such that $\langle P,\sigma\rangle \rightsquigarrow \langle P',\sigma'\rangle$. Moreover, the relation Reach^* denotes the transitive closure of the relation *Reach*. Observe that $Q \in \mathit{Reach}^*(P)$ if and only if there exist $n \geq 0, \sigma_0,\ldots,\sigma_{n-1},\theta_0,\ldots,\theta_{n-1}$ such that

$$\begin{array}{rcl} \langle P,\sigma_0\rangle & \to & \langle P_1,\theta_0\rangle \\ \langle P_1,\sigma_1\rangle & \to & \langle P_2,\theta_1\rangle \\ & \cdots & \\ \langle P_{n-1},\sigma_{n-1}\rangle & \to & \langle Q,\theta_{n-1}\rangle \end{array}$$

Intuitively, a program Q is in $\mathit{Reach}^*(P)$ if it can be obtained by reducing P starting from an arbitrary initial state and allowing arbitrary changes in the memory throughout the derivation. This is motivated by the fact that in a concurrent setting other threads could change the memory state.

By abuse of notation, for any $\langle P,\sigma\rangle \in \mathbb{P}\times\Sigma$, we denote by $\mathit{Reach}^*(\langle P,\sigma\rangle)$ the set of pairs $\langle F,\psi\rangle$ such that $\psi \in \Sigma$ and $F \in \mathit{Reach}^*(P)$. Notice that both relations $\rightsquigarrow$ and

Reach* over $\mathbb{P}\times\Sigma$ are transitive, i.e., for $\mathcal{R}\in\{\rightsquigarrow, Reach^*\}$, if $\langle F_2,\psi_2\rangle\in\mathcal{R}(\langle F_1,\psi_1\rangle)$ and $\langle F_3,\psi_3\rangle\in\mathcal{R}(\langle F_2,\psi_2\rangle)$, then $\langle F_3,\psi_3\rangle\in\mathcal{R}(\langle F_1,\psi_1\rangle)$.

EXAMPLE 2.1. Consider the following program

$$P\equiv \texttt{if}(L=1)\ \{P'\}\ \texttt{else}\ \{\texttt{skip}\}$$

where P' is the program

$$P'\equiv \texttt{if}(L\neq 1)\ \{L:=2\}\ \texttt{else}\ \{\texttt{skip}\}.$$

In this case $Reach(P)=\{P,P',\texttt{skip}\}$ while $Reach^*(P)=Reach(P)\cup\{L:=2\}$. In fact, if σ is such that $\sigma(L)=1$ we get that $\langle P,\sigma\rangle\rightarrow\langle P',\sigma\rangle$. If now we take σ' such that $\sigma'(L)\neq 1$ we get that $\langle P',\sigma'\rangle\rightarrow\langle L:=2,\sigma'\rangle$, i.e., $L:=2\in Reach^*(P)$. In particular, $\langle L:=2,\psi\rangle\in Reach^*(\langle P,\sigma\rangle)$ for all $\psi,\sigma\in\Sigma$. □

From now on, in the examples we denote by L a low level location and by H a high level one.

EXAMPLE 2.2. Consider the following program

$$P\equiv\ \ \texttt{if}(H\leq 3)\ \{L:=L+1\}\ \ \texttt{else}\ \ \{L:=L+2\}.$$

Let σ_1, σ_2 be two states such that $\sigma_1(H)\leq 3$ and $\sigma_2(H)>3$. The LTS's associated to the pairs $\langle P,\sigma_1\rangle$ and $\langle P,\sigma_2\rangle$ are

$$\begin{array}{c}\langle P,\sigma_1\rangle\\ \downarrow{\scriptstyle \texttt{high}}\\ \langle L:=L+1,\sigma_1\rangle\\ \downarrow{\scriptstyle \texttt{low}}\\ \langle \texttt{end},\sigma_1[L/\sigma_1(L)+1]\rangle\end{array}\qquad\qquad\begin{array}{c}\langle P,\sigma_2\rangle\\ \downarrow{\scriptstyle \texttt{high}}\\ \langle L:=L+2,\sigma_2\rangle\\ \downarrow{\scriptstyle \texttt{low}}\\ \langle \texttt{end},\sigma_2[L/\sigma_2(L)+2]\rangle\end{array}$$

In this case the final value of the low level location depends on the initial value of the high level one. Hence a low level user can infer whether H is less or equal than 3 or not just by observing the initial and final values of L.

To illustrate the semantics of the `await` statement, consider now the program

$$Q\equiv\ \ \texttt{await}(L\leq 3)\ \{L_1:=H;L_2:=5;L_1:=L_2\};H:=L_1.$$

Let σ be a state such that $\sigma(L)\leq 3$. The LTS associated to the pair $\langle Q,\sigma\rangle$ is

$$\begin{array}{c}\langle Q,\sigma\rangle\\ \downarrow{\scriptstyle \texttt{high}}\\ \langle H:=L_1,\sigma[L_1/5,L_2/5]\rangle\\ \downarrow{\scriptstyle \texttt{low}}\\ \langle \texttt{end},\sigma[L_1/5,L_2/5,H/5]\rangle\end{array}$$

□

The following property holds.

Lemma 2.3. *For each ψ and π such that $\psi =_l \pi$, if $\langle F,\psi\rangle \xrightarrow{\texttt{low}} \langle F',\psi'\rangle$, then $\langle F,\pi\rangle \xrightarrow{\texttt{low}} \langle F',\pi'\rangle$ with $\pi' =_l \psi'$.* □

Proof. By structural induction on programs since low transitions do not read high level variables. □

We are interested in a notion of behavioural equivalence which equates two programs if they are indistinguishable for a low level observer. In particular, we assume that a low level observer may have access to the low level locations at any point of the execution and not only at program termination. In fact, by exploiting the parallel composition operator any user may keep trace of the values of the variables during the execution.

EXAMPLE 2.4. Consider the programs P and Q below:

$$P \equiv L := H; L := 0$$

$$Q \equiv L := H - H; L := 0.$$

Let σ be a state such that $\sigma(H) \neq 0$. Since $\sigma(H) - \sigma(H)$ is always 0, the LTS's associated to the two programs are respectively

$$\begin{array}{ccc} \langle L := H; L := 0, \sigma\rangle & & \langle L := H - H; L := 0, \sigma\rangle \\ \downarrow{\scriptstyle \texttt{high}} & & \downarrow{\scriptstyle \texttt{high}} \\ \langle L := 0, \sigma[L/\sigma(H)]\rangle & & \langle L := 0, \sigma[L/0]\rangle \\ \downarrow{\scriptstyle \texttt{low}} & & \downarrow{\scriptstyle \texttt{low}} \\ \langle \texttt{end}, \sigma[L/0]\rangle & & \langle \texttt{end}, \sigma[L/0]\rangle \end{array}$$

These two program executions cannot be considered equivalent. In fact, let $R \equiv L_1 := L$. If we consider the program co $P \| R$ oc, it is possible that (i.e., there exists an execution such that) at the end of the execution L_1 contains the value stored in H, while this is never possible with the program co $Q \| R$ oc.

Consider now the programs

$$P \equiv H := 1; L := 1 \qquad \text{and} \qquad Q \equiv H := 2; L := H - 1.$$

Given a state σ the LTS's associated to the two programs are respectively

$$\begin{array}{ccc} \langle H := 1; L := 1, \sigma\rangle & & \langle H := 2; L := H - 1, \sigma\rangle \\ \downarrow{\scriptstyle \texttt{low}} & & \downarrow{\scriptstyle \texttt{low}} \\ \langle L := 1, \sigma[H/1]\rangle & & \langle L := H - 1, \sigma[H/2]\rangle \\ \downarrow{\scriptstyle \texttt{low}} & & \downarrow{\scriptstyle \texttt{high}} \\ \langle \texttt{end}, \sigma[H/1, L/1]\rangle & & \langle \texttt{end}, \sigma[H/2, L/1]\rangle \end{array}$$

The two program executions described above could be considered equivalent for a low level observer which can only read the values in the low level locations. This is captured by the following notion of *low level bisimulation* [7]: □

Definition 2.5. (Low Level Bisimulation) A binary symmetric relation $\mathcal{B}$ over $\mathbb{P} \times \Sigma$ is a *low level bisimulation* if for each $(\langle P,\sigma\rangle, \langle Q,\theta\rangle) \in \mathcal{B}$ it holds that:

- $\sigma =_l \theta$, i.e., the states coincide on low level locations;
- if $\langle P,\sigma\rangle \rightarrow \langle P',\sigma'\rangle$, then there exists $\langle Q',\theta'\rangle$ such that $\langle Q,\theta\rangle \rightarrow \langle Q',\theta'\rangle$ and $(\langle P',\sigma'\rangle,\langle Q',\theta'\rangle) \in \mathcal{B}$.

Two pairs $\langle P,\sigma\rangle$ and $\langle Q,\theta\rangle \in \mathbb{P}\times\Sigma$ are *low level bisimilar*, denoted by $\langle P,\sigma\rangle \approx_l \langle Q,\theta\rangle$ if there exists a low level bisimulation $\mathcal{B}$ such that $(\langle P,\sigma\rangle,\langle Q,\theta\rangle) \in \mathcal{B}$.
Two programs P and Q are said to be *low level bisimilar*, denoted by $P \simeq_l Q$, if for each $\sigma,\theta \in \Sigma$ it holds that if $\sigma =_l \theta$, then $\langle P,\sigma\rangle \approx_l \langle Q,\theta\rangle$. □

Notice that a pair $\langle P,\sigma\rangle$ is bisimilar to a pair of the form $\langle \texttt{end},\theta\rangle$ if and only if $P \equiv \texttt{end}$ and $\sigma =_l \theta$. In fact, end is the only program which cannot make any transition.

A *partial equivalence relation* (*per*, for short) [31] is a symmetric and transitive relation.

Lemma 2.6. *The relation $\approx_l \subseteq (\mathbb{P}\times\Sigma)^2$ is the largest low level bisimulation and it is an equivalence relation. The relation $\simeq_l \subseteq \mathbb{P}^2$ is a partial equivalence relation.* □

Proof. If $\langle P,\sigma\rangle \approx_l \langle Q,\theta\rangle$, then there exists a low level bisimulation $\mathcal{B}$ such that it holds $(\langle P,\sigma\rangle,\langle Q,\theta\rangle) \in \mathcal{B}$. Hence if $\langle P,\sigma\rangle \rightarrow \langle P',\sigma'\rangle$ we have that $\langle Q,\theta\rangle \rightarrow \langle Q',\theta'\rangle$ with $(\langle P',\sigma'\rangle,\langle Q',\theta'\rangle) \in \mathcal{B}$, i.e., $\langle P',\sigma'\rangle \approx_l \langle Q',\theta'\rangle$. So we have that $\approx_l$ is a low level bisimulation. It is the largest since by definition all the other low level bisimulations are included in it.

It is easy to prove that $\approx_l$ is reflexive and symmetric. The fact that $\approx_l$ is transitive follows from the fact that if $\mathcal{B}_1,\mathcal{B}_2$ are low level bisimulations, then the relation $\mathcal{B}_1 \circ \mathcal{B}_2$, where $\circ$ is the composition of relations, is still a low level bisimulation.

The relation $\simeq_l \subseteq \mathbb{P}^2$ is symmetric and transitive since $\approx_l$ is symmetric and transitive. □

In [30] a stronger low level bisimulation is introduced to reason on concurrent and multi-threaded programs.

Definition 2.7. (Strong Low Level Bisimulation) A binary symmetric relation $\mathcal{B}$ over $\mathbb{P}$ is a *strong low level bisimulation* if for each $(P,Q) \in \mathcal{B}$ it holds that

- for all $\sigma,\theta \in \Sigma$ such that $\sigma =_l \theta$, if $\langle P,\sigma\rangle \rightarrow \langle P',\sigma'\rangle$, then there exists Q' and θ' such that $\langle Q,\theta\rangle \rightarrow \langle Q',\theta'\rangle$, $\sigma' =_l \theta'$ and $(P',Q') \in \mathcal{B}$.

Two programs $P,Q \in \mathbb{P}$ are *strongly low level bisimilar*, denoted by $P \sim_l Q$ if there exists a low level bisimulation $\mathcal{B}$ such that $(P,Q) \in \mathcal{B}$. □

The difference between Definitions 2.5 and 2.7 is that, in the second one, P' and Q' have to be equivalent under any low-equivalent memories.

Lemma 2.8. *[30] The relation $\sim_l \subseteq \mathbb{P}^2$ is a partial equivalence relation.* □

Both the relations $\simeq_l$ and $\sim_l$ are not reflexive. For example, the program $L := H$ is neither low level bisimilar nor strongly low level bisimilar to itself. Consider for instance the states σ and θ such that $\sigma(H) = 1$, $\theta(H) = 2$, $\sigma(L) = \theta(L)$. In this case $\sigma =_l \theta$. However, $\langle L := H,\sigma\rangle \rightarrow \langle \texttt{end},\sigma[L/1]\rangle$ and $\langle L := H,\theta\rangle \rightarrow \langle \texttt{end},\theta[L/2]\rangle$ where $\sigma[L/1] \neq_l \theta[L/2]$. Therefore neither $L := H \simeq_l L := H$ nor $L := H \sim_l L := H$.

EXAMPLE 2.9. Consider the programs of Example 2.4:

$$P \equiv H := 1; L := 1 \qquad \text{and} \qquad Q \equiv H := 2; L := H - 1.$$

It is easy to prove that $P \simeq_l Q$. In fact, a low level user which can only observe the low level location L cannot distinguish the two programs. However, P and Q are not strongly low level bisimilar, i.e., $P \not\sim_l Q$. Indeed, changing the state σ to $\sigma_{[H/5]}$ after the first transition, we obtain two different final values for L (1 and 4, respectively). This reflects the fact that if one considers, for instance, the program $R \equiv H := 5$ then the programs $\mathtt{co}\ P \| R\ \mathtt{oc}$ and $\mathtt{co}\ Q \| R\ \mathtt{oc}$ do not exhibit the same behaviour from the low level point of view. In fact, starting from a state σ, any execution of $\mathtt{co}\ P \| R\ \mathtt{oc}$ always terminates in a state σ_1 such that $\sigma_1(L) = 1$. On the other hand, there exists one execution of $\mathtt{co}\ Q \| R\ \mathtt{oc}$ which terminate in a state σ_2 such that $\sigma_2(L) = 4$, i.e., $\sigma_1 \not=_l \sigma_2$. □

The following lemma trivially follows from the definition of partial equivalence relation.

Lemma 2.10. *Let P be a program. Either $P \simeq_l P$ ($P \sim_l P$) or for each $P' \in \mathbb{P}$ it holds $P \not\sim_l P'$ ($P \not\sim_l P'$).* □

The following useful property of $\sim_l$ holds.

Lemma 2.11. *If $P \sim_l P$, then for all $F \in Reach^*(P)$ it holds that $F \sim_l F$.* □

Proof. If $F \in Reach^*(P)$, then there are $n \geq 0$ and $P_0, \ldots, P_n$, $\sigma_0, \ldots, \sigma_{n-1}$, $\theta_0, \ldots, \theta_{n-1}$ such that $P_0 \equiv P$, $P_n \equiv F$ and for each $0 \leq i \leq n-1$ it holds $\langle P_i, \sigma_i \rangle \to \langle P_{i+1}, \theta_i \rangle$. We prove that $P_n \sim_l P_n$ by induction on n.

Base. If $n = 0$, then $P_n \equiv P$, hence we immediately get the thesis.

Inductive step. Let $n > 0$ and $\langle P_{n-1}, \sigma_{n-1} \rangle \to \langle P_n, \theta_{n-1} \rangle$. By inductive hypothesis, $P_{n-1} \sim_l P_{n-1}$. Thus by definition of $\sim_l$ there exist Q such that $\langle P_{n-1}, \sigma_{n-1} \rangle \to \langle Q, \mu \rangle$ with $P_n \sim_l Q$. By Lemma 2.10 from $P_n \sim_l Q$ we get $P_n \sim_l P_n$. □

It is immediate to prove that $\sim_l \subseteq \simeq_l$.

The following lemma states that the relations $\simeq_l$ and $\sim_l$ equate programs which exhibit the same behavior. Specifically, each step performed by one program is simulated by exactly one step performed by the other program.

Lemma 2.12. *Let P and Q be two programs and $\sigma, \theta \in \Sigma$.*

(1) *Let $\langle P, \sigma \rangle \approx_l \langle Q, \theta \rangle$. If $\langle P, \sigma \rangle \to^n \langle P', \sigma' \rangle$, then there exists Q' and θ' such that $\langle Q, \theta \rangle \to^n \langle Q', \theta' \rangle$ and $\langle P', \sigma' \rangle \approx_l \langle Q', \theta' \rangle$, and viceversa.*

(2) *Let $P \sim_l Q$ and $\sigma =_l \theta$. If $\langle P, \sigma \rangle \to^n \langle P', \sigma' \rangle$, then there exists Q' and θ' such that $\langle Q, \theta \rangle \to^n \langle Q', \theta' \rangle$, $\sigma' =_l \theta'$ and $P' \sim_l Q'$, and viceversa.*

□

Proof. (1). By induction on n.

- Base: $n = 1$. We immediately have the thesis by definition of $\approx_l$.

- Inductive step: $n = m+1$ and we proved the thesis for m. We have that $\langle P,\sigma\rangle \rightarrow^m \langle P'',\sigma''\rangle \rightarrow \langle P',\sigma'\rangle$. By inductive hypothesis we get $\langle Q,\theta\rangle \rightarrow^m \langle Q'',\theta''\rangle$ with $\langle P'',\sigma''\rangle \approx_l \langle Q'',\theta''\rangle$. By definition of bisimulation we get the thesis.

The proof of (2) is analogous. □

3 Unwinding Conditions for Security

In [5] we introduced a general framework to define classes of secure processes written in the Security Process Algebra (SPA) language, an extension of Milner's CCS [20]. The framework is based on a generalized unwinding condition which is a local persistent property parametric with respect to a notion of low level behavioral observation and a reachability relation. We proved that many noninterference properties can be seen as instances of this framework. Following a similar approach, in this paper we introduce a generalized unwinding condition for defining classes of programs that is parametric with respect to:

- a binary relation $\doteq$ which equates two states if they are indistinguishable for a low level observer;
- a binary relation $\dot{\div}$ which equates two pairs $\langle P,\sigma\rangle$ and $\langle Q,\theta\rangle$ if they are indistinguishable for a low level observer;
- a binary reachability relation $\mathcal{R}$ on $\mathbb{P}\times\Sigma$ which associates to each pair $\langle P,\sigma\rangle$ all the pairs $\langle F,\psi\rangle$ which, in some sense, are reachable from $\langle P,\sigma\rangle$.

A pair $\langle P,\sigma\rangle$ satisfies (an instance of) our unwinding framework if any high level step $\langle F,\psi\rangle \stackrel{\mathtt{high}}{\rightarrow} \langle G,\varphi\rangle$ performed by a pair $\langle F,\psi\rangle$ reachable from $\langle P,\sigma\rangle$ has no effect on the observation of a low level user. This is achieved by requiring that all the elements in the set $\{\langle F,\pi\rangle \mid \pi \doteq \psi\}$ (whose states are low level equivalent) may perform a transition reaching an element of the set $\{\langle R,\rho\rangle \mid \langle R,\rho\rangle \dot{\div} \langle G,\varphi\rangle\}$ (whose elements are all indistinguishable for a low level observer).

Definition 3.1. (Generalized Unwinding) Let $\doteq$ be a binary relation over Σ, $\dot{\div}$ and $\mathcal{R}$ be two binary relations over $\mathbb{P}\times\Sigma$. We define the *unwinding class* $\mathcal{W}(\doteq,\dot{\div},\mathcal{R})$ by:

$$\begin{aligned}\mathcal{W}(\doteq,\dot{\div},\mathcal{R}) \stackrel{\text{def}}{=} \{\langle P,\sigma\rangle \in \mathbf{Prog}\times\Sigma \mid \forall\, \langle F,\psi\rangle \in \mathcal{R}(\langle P,\sigma\rangle)\\ \text{if } \langle F,\psi\rangle \stackrel{\mathtt{high}}{\rightarrow} \langle G,\varphi\rangle \text{ then}\\ \forall\, \pi\in\Sigma \text{ such that } \pi\doteq\psi,\ \exists\, \langle R,\rho\rangle :\\ \langle F,\pi\rangle \rightarrow \langle R,\rho\rangle \text{ and } \langle G,\varphi\rangle \dot{\div} \langle R,\rho\rangle\}\end{aligned}$$

□

The intuition behind the unwinding condition is that any high level transition should be simulated by a high independent transition guaranteeing that the low level observer cannot infer which kind of transition occurred. In other words, the high level transitions have no influence on the low level observation. This intuition is depicted by the following diagram:

$$\begin{array}{ccc} \langle F,\psi\rangle & \stackrel{\mathtt{high}}{\longrightarrow} & \langle G,\varphi\rangle \\ \downarrow & & \uparrow \\ \pi \doteq \psi & & \doteqdot \\ \downarrow & & \downarrow \\ \langle F,\pi\rangle & \longrightarrow & \langle R,\rho\rangle \end{array}$$

In the above definition we do not assume any condition on the reachability relation $\mathcal{R}$. We can show that whenever $\mathcal{R}$ is transitive, the generalized unwinding condition allows one to specify properties which are closed under $\mathcal{R}$. In this sense we say that our properties are *persistent*.

Lemma 3.2. *Let $\mathcal{R}$ be a transitive reachability relation on pairs of $\mathbb{P}\times\Sigma$ and let $\langle P,\sigma\rangle \in \mathbf{Prog}\times\Sigma$. If $\langle P,\sigma\rangle \in \mathcal{W}(\doteq,\doteqdot,\mathcal{R})$, then $\langle F,\psi\rangle \in \mathcal{W}(\doteq,\doteqdot,\mathcal{R})$ for all $\langle F,\psi\rangle \in \mathcal{R}(\langle P,\sigma\rangle)$.* □

Proof. Let $\mathcal{R}$ be transitive, $\langle P,\sigma\rangle \in \mathcal{W}(\doteq,\doteqdot,\mathcal{R})$, and $\langle F,\psi\rangle \in \mathcal{R}(\langle P,\sigma\rangle)$. If $\langle F',\psi'\rangle \in \mathcal{R}(\langle F,\psi\rangle)$, then by transitivity we have that $\langle F',\psi'\rangle \in \mathcal{R}(\langle P,\sigma\rangle)$. Hence we get that if $\langle F',\psi'\rangle \stackrel{\mathtt{high}}{\rightarrow} \langle G',\varphi'\rangle$, then for each π such that $\psi' =_l \pi'$ there exists $\langle R',\rho'\rangle$ such that $\langle F',\pi'\rangle \rightarrow \langle R',\rho'\rangle$ with $\langle G',\varphi'\rangle \doteqdot \langle R',\rho'\rangle$, i.e., the thesis. □

Hereafter we discuss some instantiations of our generalized unwinding condition.

The class of secure imperative programs $\mathbf{SIMP}_{\approx_l}$ is obtained by instantiating Definition 3.1 with the low level bisimilarities $=_l$ for $\doteq$ and $\approx_l$ for $\doteqdot$, and the relation $\rightsquigarrow$ for $\mathcal{R}$.

Definition 3.3. ($\mathbf{SIMP}_{\approx_l}$) A program P belongs to the class $\mathbf{SIMP}_{\approx_l}$ if for each state σ, $\langle P,\sigma\rangle \in \mathcal{W}(=_l,\approx_l,\rightsquigarrow)$. □

EXAMPLE 3.4. Consider the program

$$P \equiv L := H; L := 1.$$

We can prove that it does not hold $\langle P,\sigma\rangle \in \mathcal{W}(=_l,\approx_l,\rightsquigarrow)$ for any $\sigma\in\Sigma$. In fact, let for instance $\sigma(H)=1$, $\sigma(L)=0$, $\theta(H)=2$, $\theta(L)=0$. It holds that $\sigma =_l \theta$, but after the execution of the first high transition we reach the states σ' and θ' with $\sigma'(L)=1 \neq \theta'(L)=2$. In this case, a low level user which can observe the intermediate values of the low level locations, may infer the initial value of H just by observing the state of the memory after one execution step.

Let now

$$P \equiv H := 4; L := 1; \mathtt{if}(L=1)\ \{\mathtt{skip}\}\ \mathtt{else}\ \{L := H\}.$$

P is in the class $\mathbf{SIMP}_{\approx_l}$. In fact, the first branch of the conditional is always executed independently of the value in the high level location. □

Since $\rightsquigarrow$ is transitive, by Lemma 3.2 we get that $\mathcal{W}(=_l,\approx_l,\rightsquigarrow)$ is persistent, i.e., if a program P starting in a state σ belongs to the class $\mathcal{W}(=_l,\approx_l,\rightsquigarrow)$, then also each

pair $\langle P', \sigma' \rangle$ reachable from $\langle P, \sigma \rangle$ does. However, in general it does not hold that if P is in $\mathbf{SIMP}_{\approx_l}$, then also each program $P' \in \mathit{Reach}(P)$ is in $\mathbf{SIMP}_{\approx_l}$. This is illustrated in the following example.

EXAMPLE 3.5. Consider the program $P \equiv L := 0; P'$ where

$$P' \equiv \ \texttt{if}(L = 1)\ \{L := H\}\ \texttt{else}\ \{\texttt{skip}\}.$$

It holds that $P \in \mathbf{SIMP}_{\approx_l}$ since, for each state σ, $\langle P, \sigma \rangle$ will never perform any high transition. However, the program P' is in $\mathit{Reach}(P)$ but $P' \notin \mathbf{SIMP}_{\approx_l}$.

Consider now the programs

$$Q_1 \equiv \ \texttt{await}(\texttt{true})\ \{L_1 := H; L_2 := 5; L_1 := L_2\}; H := L_1$$

$$Q_2 \equiv \ \texttt{await}(\texttt{true})\ \{L_1 := H; L_2 := L_1; L_1 := L_2\}; H := L_1.$$

It is immediate to prove that $Q_1 \in \mathbf{SIMP}_{\approx_l}$ since, for all states σ_1 and θ_1 such that $\sigma_1 =_l \theta_1$, $\langle Q_1, \sigma_1 \rangle \overset{\texttt{high}}{\rightarrow} \langle H := L_1, \sigma_1' \rangle$ and $\langle Q_1, \theta_1 \rangle \overset{\texttt{high}}{\rightarrow} \langle H := L_1, \theta_1' \rangle$ and $\sigma_1' =_l \theta_1'$. This does not hold for Q_2. Consider for instance σ_2 and θ_2 such that $\sigma_2 =_l \theta_2$, $\sigma_2(H) = 1$ and $\theta_2(H) = 0$. In this case, $\langle Q_2, \sigma_2 \rangle \overset{\texttt{high}}{\rightarrow} \langle H := L_1, \sigma_2' \rangle$ and $\langle Q_2, \theta_2 \rangle \overset{\texttt{high}}{\rightarrow} \langle H := L_1, \theta_2' \rangle$ but $\sigma_2'(L_1) = \sigma_2'(L_2) = 1$ while $\theta_2'(L_1) = \theta_2'(L_2) = 0$. □

A more restrictive class of secure imperative programs can be introduced by instantiating our generalized unwinding with the reachability relation Reach^*.

Definition 3.6. ($\mathbf{SIMP}^*_{\approx_l}$) A program P belongs to the class $\mathbf{SIMP}^*_{\approx_l}$ if for each state σ, $\langle P, \sigma \rangle \in \mathcal{W}(=_l, \approx_l, \mathit{Reach}^*)$. □

Lemma 3.7. $\mathbf{SIMP}^*_{\approx_l} \subseteq \mathbf{SIMP}_{\approx_l}$. □

Proof. This is a consequence of the fact that for each P, Q and σ, θ it holds that $\langle P, \sigma \rangle \rightsquigarrow \langle Q, \theta \rangle$ implies $\langle Q, \theta \rangle \in \mathit{Reach}^*(\langle P, \sigma \rangle)$. □

The two classes do not coincide as shown below.

EXAMPLE 3.8. Consider the program

$$P \equiv H := 4; L := 1; \texttt{if}(L = 1)\ \{\texttt{skip}\}\ \texttt{else}\ \ \{L := H\}.$$

It belongs to the class $\mathbf{SIMP}_{\approx_l}$ but it does not belong to the class $\mathbf{SIMP}^*_{\approx_l}$. In fact given an initial state σ there exists a state ψ such that the pair $\langle L := H, \psi \rangle$ belongs to $\mathit{Reach}^*(\langle P, \sigma \rangle)$. Moreover $\langle L := H, \psi \rangle \overset{\texttt{high}}{\rightarrow} \langle \texttt{end}, \varphi \rangle$ but it clearly does not hold that for each π such that $\pi =_l \psi$ there exists $\langle R, \rho \rangle$ such that $\langle L := H, \pi \rangle \rightarrow \langle R, \rho \rangle$ and $\langle R, \rho \rangle \approx_l \langle \texttt{end}, \varphi \rangle$. □

The relation Reach^* on $\mathbb{P} \times \Sigma$ is transitive and then, by Lemma 3.2, the class $\mathcal{W}(=_l, \approx_l, \mathit{Reach}^*)$ is persistent, i.e., if $\langle P, \sigma \rangle$ is in $\mathcal{W}(=_l, \approx_l, \mathit{Reach}^*)$, then also each pair $\langle P', \sigma' \rangle \in \mathit{Reach}^*(\langle P, \sigma \rangle)$ is in $\mathcal{W}(=_l, \approx_l, \mathit{Reach}^*)$. Moreover, differently from $\mathbf{SIMP}_{\approx_l}$, if a program P is in $\mathbf{SIMP}^*_{\approx_l}$, then also each $P' \in \mathit{Reach}^*(P)$ is in $\mathbf{SIMP}^*_{\approx_l}$.

Lemma 3.9. *Let P be a program. If $P \in \mathbf{SIMP}^*_{\approx_l}$, then for all $P' \in \mathit{Reach}^*(P)$, $P' \in \mathbf{SIMP}^*_{\approx_l}$.* □

Proof. Let $P' \in Reach^*(P)$, hence, $\langle P', \sigma' \rangle \in Reach^*(\langle P, \sigma \rangle)$ for some σ and σ'. By definition of $Reach^*$, $\langle P', \theta \rangle \in Reach^*(\langle P, \sigma \rangle)$ for each state θ. Hence, by persistence of $\mathcal{W}(=_l, \approx_l, Reach^*)$, $\langle P', \theta \rangle \in \mathcal{W}(=_l, \approx_l, Reach^*)$, i.e., $P' \in \mathbf{SIMP}^*_{\approx_l}$. □

Both $\mathcal{W}(=_l, \approx_l, \rightsquigarrow)$ and $\mathcal{W}(=_l, \approx_l, Reach^*)$ allow us to express notions of security. This is a consequence of the fact that $\approx_l$ equates programs which exhibit the same behavior (see Lemma 2.12).

In the previous section we observed that the relation $\simeq_l$ is not reflexive. However, $\simeq_l$ is reflexive on the class $\mathbf{SIMP}_{\approx_l}$ (and then, by Lemma 3.7, on $\mathbf{SIMP}^*_{\approx_l}$).

Lemma 3.10. *Let P be a program. If $P \in \mathbf{SIMP}_{\approx_l}$, then $P \simeq_l P$.* □

Proof. Assume that $P \in \mathbf{SIMP}_{\approx_l}$. Then for all states σ and θ, $\langle P, \sigma \rangle, \langle P, \theta \rangle \in \mathcal{W}(=_l, \approx_l, \rightsquigarrow)$. Hence, in order to prove that $P \simeq_l P$, it is sufficient to show that for all σ and θ such that $\langle P, \sigma \rangle, \langle P, \theta \rangle \in \mathcal{W}(=_l, \approx_l, \rightsquigarrow)$ and $\sigma =_l \theta$, it holds $\langle P, \sigma \rangle \approx_l \langle P, \theta \rangle$. Consider the binary relation

$$\begin{array}{rcl} \mathcal{S} & = & \{(\langle P, \sigma \rangle, \langle P, \theta \rangle) \mid \langle P, \sigma \rangle, \langle P, \theta \rangle \in \mathcal{W}(=_l, \approx_l, \rightsquigarrow), \sigma =_l \theta\} \\ & \cup & \{(\langle P, \sigma \rangle, \langle Q, \theta \rangle) \mid \langle P, \sigma \rangle \sim_l \langle Q, \theta \rangle\} \end{array}$$

We show that $\mathcal{S}$ is a low level bisimulation $\approx_l$.

If $\langle P, \sigma \rangle \stackrel{\mathtt{high}}{\rightarrow} \langle P', \sigma' \rangle$, then from the fact that $\langle P, \sigma \rangle \in \mathcal{W}(=_l, \approx_l, \rightsquigarrow)$ we have that $\langle P, \theta \rangle \rightarrow \langle P'', \theta' \rangle$ with $\langle P', \sigma' \rangle \approx_l \langle P'', \theta' \rangle$. By definition of $\mathcal{S}$, $(\langle P', \sigma' \rangle, \langle P'', \theta' \rangle) \in \mathcal{S}$.

If $\langle P, \sigma \rangle \stackrel{\mathtt{low}}{\rightarrow} \langle P', \sigma' \rangle$, then by Lemma 2.3 we have that $\langle P, \theta \rangle \stackrel{\mathtt{low}}{\rightarrow} \langle P', \theta' \rangle$ with $\sigma' =_l \theta'$. By Lemma 3.2, since $\rightsquigarrow$ is transitive, we have that, both $\langle P', \sigma' \rangle \in \mathcal{W}(=_l, \approx_l, \rightsquigarrow)$ and $\langle P', \theta' \rangle \in \mathcal{W}(=_l, \approx_l, \rightsquigarrow)$. Hence we have that $(\langle P', \sigma' \rangle, \langle P', \theta' \rangle) \in \mathcal{S}$, i.e., the thesis. □

The converse of Lemma 3.10 does not hold in general as illustrated in the following example.

EXAMPLE 3.11. Consider the program

$$P \equiv \mathtt{if}(H = 1)\ \{P_0\}\ \mathtt{else}\ \{P_1\}$$

where

$$P_0 \equiv \mathtt{while}(H > 1)\ \{\mathtt{skip}\} \qquad \text{and} \qquad P_1 \equiv \mathtt{skip}.$$

One can prove that $P \simeq_l P$, i.e., for all states σ and θ such that $\sigma =_l \theta$, $\langle P, \sigma \rangle \approx_l \langle P, \theta \rangle$. However, the program $P \notin \mathbf{SIMP}_{\approx_l}$. In fact, for any σ such that $\langle P, \sigma \rangle \rightsquigarrow \langle P_0, \sigma \rangle$ we have that $\langle P_0, \sigma \rangle \stackrel{\mathtt{high}}{\rightarrow} \langle \mathtt{end}, \sigma \rangle$. But it does not hold that for all ρ such that $\sigma =_l \rho$ there exist R and ρ' such that $\langle P_0, \rho \rangle \rightarrow \langle R, \rho' \rangle$ and $\langle \mathtt{end}, \sigma \rangle \sim_l \langle R, \rho' \rangle$. Indeed, if $\rho(H) > 1$, $\langle P_0, \rho \rangle \rightarrow \langle \mathtt{skip}; P_0, \rho \rangle$ and $\langle \mathtt{end}, \sigma \rangle \not\sim_l \langle \mathtt{skip}; P_0, \rho \rangle$. This is due to the fact that the subprogram P_0 of P is not in $\mathbf{SIMP}_{\approx_l}$. □

However, it holds that if $P' \simeq_l P'$ for all $P' \in Reach^*(P)$, then $P \in \mathbf{SIMP}^*_{\approx_l}$.

Lemma 3.12. *Let P be a program such that $P' \simeq_l P'$ for all $P' \in Reach^*(P)$. It holds $P \in \mathbf{SIMP}^*_{\approx_l}$.* □

Proof. We prove that for all $\sigma \in \Sigma$, $\langle P,\sigma\rangle \in \mathcal{W}(=_l, \approx_l, Reach^*)$. Let us consider $\langle F,\psi\rangle \in Reach^*(\langle P,\sigma\rangle)$. Then, by definition of $Reach^*$, $F \in Reach^*(P)$. Hence, by hypothesis, $F \simeq_l F$.

If $\langle F,\psi\rangle \stackrel{\mathtt{high}}{\rightarrow} \langle G,\varphi\rangle$, then, since $F \simeq_l F$, for each $\pi \in \Sigma$ such that $\psi =_l \pi$ it holds $\langle F,\pi\rangle \rightarrow \langle R,\rho\rangle$ with $\langle G,\varphi\rangle \approx_l \langle R,\rho\rangle$, i.e., the thesis. □

The next theorem provides a characterization of $\mathbf{SIMP}^*_{\approx_l}$ in terms of the low level bisimulation $\simeq_l$.

Theorem 3.13. *$P \in \mathbf{SIMP}^*_{\approx_l}$ if and only if $P' \simeq_l P'$ for all $P' \in Reach^*(P)$.* □

Proof. $\Leftarrow$) This has been proved in Lemma 3.12.

$\Rightarrow$) If $P \in \mathbf{SIMP}^*_{\approx_l}$, then by Lemma 3.9 we get that for each $P' \in Reach^*(P)$ it holds $P' \in \mathbf{SIMP}^*_{\approx_l}$. Hence, by Lemma 3.7 we have that for each $P' \in Reach^*$ it holds $P' \in \mathbf{SIMP}_{\approx_l}$. By Lemma 3.10 we can conclude that for each $P' \in Reach^*$ it holds $P' \simeq_l P'$. □

The classes $\mathbf{SIMP}_{\approx_l}$ and $\mathbf{SIMP}^*_{\approx_l}$ introduced above are not compositional with respect to the parallel composition constructor as illustrated by the following example.

EXAMPLE 3.14. Consider the program

$$P \equiv \mathtt{if}(H = 1)\ \{P_0\}\ \mathtt{else}\ \{P_1\}$$

where

$$P_0 \equiv L := 1;\ \ \mathtt{if}(L = 1)\ \{L := 2\}\ \ \mathtt{else}\ \{L := 3\}$$

and

$$P_1 \equiv L := 1; \mathtt{skip}; L := 2.$$

We have that for each σ and θ such that $\sigma =_l \theta$ it holds $\langle P_0,\sigma\rangle \approx_l \langle P_1,\theta\rangle$, i.e., $P_0 \simeq_l P_1$. From this we get that $P \simeq_l P$. Moreover, it is easy to see that for each $P' \in Reach^*(P_0) \cup Reach^*(P_1)$ it holds $P' \simeq_l P'$. Hence, by Theorem 3.13, we can say that $P \in \mathbf{SIMP}^*_{\approx_l}$. Consider also the program $Q \equiv L := 0$. It is immediate to see that $Q \in \mathbf{SIMP}^*_{\approx_l}$. However, if we consider $\mathtt{co}\ P \| Q\ \mathtt{oc}$ this does not belong to $\mathbf{SIMP}^*_{\approx_l}$. In fact, if σ and θ are such that $\sigma =_l \theta$, $\sigma(H) = 1$ and $\theta(H) = 0$ we get that $\langle \mathtt{co}\ P \| Q\ \mathtt{oc},\sigma\rangle \rightarrow \langle \mathtt{co}\ P_0 \| Q\ \mathtt{oc},\sigma\rangle$, while $\langle \mathtt{co}\ P \| Q\ \mathtt{oc},\theta\rangle \rightarrow \langle \mathtt{co}\ P_1 \| Q\ \mathtt{oc},\theta\rangle$. We can see that it does not hold $\langle \mathtt{co}\ P_0 \| Q\ \mathtt{oc},\sigma\rangle \approx_l \langle \mathtt{co}\ P_1 \| Q\ \mathtt{oc},\theta\rangle$, since the first can assign 3 to L, while the second does not. □

Compositionality is useful both for verification and synthesis: if a property is preserved when programs are composed, then the analysis may be performed on subprograms and, in case of success, the program as a whole will satisfy the desired property by construction. This motivates the study of stronger properties such as the one defined below.

We introduce the class of secure imperative programs $\mathbf{SIMP}^*$ which is obtained by instantiating our generalized unwinding condition with the relation $Reach^*$ for $\mathcal{R}$ and the low level bisimilarity $\ddot{\sim}_l$ defined below for $\div$.

Definition 3.15. The relation $\ddot{\sim}_l$ over $\mathbb{P} \times \Sigma$ is defined as follows: $\langle P,\sigma\rangle \ddot{\sim}_l \langle Q,\theta\rangle$ if $\sigma =_l \theta$ and $P \sim_l Q$. □

The relation $\ddot{\sim}_l$ is a partial equivalence relation. This follows from the fact that both $=_l$ and $\sim_l$ are symmetric and transitive. The following inclusion holds: $\ddot{\sim}_l \subseteq \approx_l$.

Definition 3.16. (SIMP*) A program P belongs to the class **SIMP*** if for each state σ, $\langle P,\sigma\rangle \in \mathcal{W}(=_l, \ddot{\sim}_l, Reach^*)$. □

As for the above classes, $\mathcal{W}(=_l, \ddot{\sim}_l, Reach^*)$ is persistent in the sense that if $\langle P,\sigma\rangle$ is in $\mathcal{W}(=_l, \ddot{\sim}_l, Reach^*)$, then also each pair $\langle P',\sigma'\rangle \in Reach^*(\langle P,\sigma\rangle)$ is in $\mathcal{W}(=_l, \ddot{\sim}_l, Reach^*)$. Moreover, it holds that if a program P is in **SIMP***, then also each program $P' \in Reach^*(P)$ is in **SIMP***.

Lemma 3.17. *Let P be a program. If $P \in$ **SIMP***, then for all $P' \in Reach^*(P)$, $P' \in$ **SIMP***.* □

Proof. Let $P' \in Reach^*(P)$, hence, $\langle P',\sigma'\rangle \in Reach^*(\langle P,\sigma\rangle)$ for some σ and σ'. By definition of $Reach^*$, $\langle P',\theta\rangle \in Reach^*(\langle P,\sigma\rangle)$ for each state θ. Hence, by persistence of $\mathcal{W}(=_l, \ddot{\sim}_l, Reach^*)$, $\langle P',\theta\rangle \in \mathcal{W}(=_l, \ddot{\sim}_l, Reach^*)$, i.e., $P' \in$ **SIMP***. □

Interestingly, it turns out that the reflexive closure of the relation $\sim_l$ exactly coincides with the set of programs in **SIMP***.

Theorem 3.18. *Let P be a program. $P \in$ **SIMP*** if and only if $P \sim_l P$.* □

Proof. $\Rightarrow$) Consider the binary relation

$$\mathcal{S} = \{(P,P) \mid P \in \textbf{SIMP}^*\} \cup \{(P,Q) \mid P \sim_l Q\}$$

We show that $\mathcal{S}$ is a strong low level bisimulation $\sim_l$. This follows from the following cases. Let σ and θ be two states such that $\sigma =_l \theta$.

If $\langle P,\sigma\rangle \overset{\texttt{high}}{\rightarrow} \langle P',\sigma'\rangle$, then since $P \in$ **SIMP***, $\langle P,\sigma\rangle \in \mathcal{W}(=_l, \ddot{\sim}_l, Reach^*)$ and then $\langle P,\theta\rangle \rightarrow \langle P'',\theta'\rangle$ with $\langle P',\sigma'\rangle \ddot{\sim}_l \langle P'',\theta'\rangle$, i.e., $\sigma' =_l \theta'$ and $P' \sim_l P''$. Hence, by definition of $\mathcal{S}$, $(P',P'') \in \mathcal{S}$.

If $\langle P,\sigma\rangle \overset{\texttt{low}}{\rightarrow} \langle P',\sigma'\rangle$, then by Lemma 2.3 we have that $\langle P,\theta\rangle \overset{\texttt{low}}{\rightarrow} \langle P',\theta'\rangle$ with $\sigma' =_l \theta'$. By Lemma 3.17, we have that $P' \in$ **SIMP*** and then, by definition of $\mathcal{S}$, $(P',P') \in \mathcal{S}$, i.e., the thesis.

$\Leftarrow$) Let P be a program such that $P \sim_l P$. Let $\sigma,\psi \in \Sigma$ and $\langle F,\psi\rangle \in Reach^*(\langle P,\sigma\rangle)$. Then $F \in Reach^*(P)$ and, by Lemma 2.11, $F \sim_l F$. Let π be such that $\psi =_l \pi$.

If $\langle F,\psi\rangle \overset{\texttt{high}}{\rightarrow} \langle G,\varphi\rangle$, then, since $F \sim_l F$, $\langle F,\pi\rangle \rightarrow \langle R,\rho\rangle$ with $\varphi =_l \rho$ and $G \sim_l R$, i.e., the thesis. □

The class **SIMP*** is more restrictive than $\textbf{SIMP}^*_{\approx_l}$ and $\textbf{SIMP}_{\approx_l}$. This is a consequence of the fact that $\ddot{\sim}_l \subseteq \approx_l$.

Lemma 3.19. $\textbf{SIMP}^* \subseteq \textbf{SIMP}^*_{\approx_l} \subseteq \textbf{SIMP}_{\approx_l}$. □

Proof. We only have to prove that $\textbf{SIMP}^* \subseteq \textbf{SIMP}^*_{\approx_l}$. If $P \in$ **SIMP*** by Lemma 3.17 we get that for each $P' \in Reach^*(P)$ it holds $P' \in$ **SIMP***. Hence by Theorem 3.18 we have that for each $P' \in Reach^*(P)$ it holds $P' \sim_l P'$. Since $\sim_l \subseteq \simeq_l$, we get that for each $P' \in Reach^*(P)$ it holds $P' \simeq_l P'$, i.e., by Theorem 3.13, the thesis. □

Moreover, **SIMP*** does not coincide with $\mathbf{SIMP}^*_{\approx_l}$ as shown by the following example.

EXAMPLE 3.20. Consider the program P of Example 3.14. It holds that $P \in \mathbf{SIMP}^*_{\approx_l}$. However, it does not hold $P \in \mathbf{SIMP}^*$. To show this it is sufficient to prove that $P \not\sim_l P$. Indeed, if σ and θ are such that $\sigma =_l \theta$, $\sigma(H) = 1$, and $\theta(H) = 0$, we get that $\langle P, \sigma\rangle \to \langle P_0, \sigma\rangle$ and $\langle P, \theta\rangle \to \langle P_1, \theta\rangle$. It does not hold $P_0 \sim_l P_1$ since $\langle P_0, \sigma\rangle \to \langle P_0', \sigma[L/1]\rangle$, where $P_0' \equiv \texttt{if}\ (L = 1)\{L := 2\}\ \texttt{else}\ \{L := 3\}$ and $\langle P_1, \theta\rangle \to \langle P_1', \theta[L/1]\rangle$, where $P_1' \equiv \texttt{skip}; L := 2$ and it is plain that $P_0' \not\sim_l P_1'$. □

The class **SIMP*** enjoys several compositional properties which are useful for the development of automatic verification techniques. The following theorem states that the class **SIMP*** is compositional with respect to most of the language constructors.

Theorem 3.21. (Compositionality) *Let H be a high level location, L be a low level location, a_h and b_h be high level expressions, and a_l and b_l be low level expressions. If P_0, P_1, and S are in* **SIMP****, then also the following programs are in* **SIMP****:*

(1) `skip`*;*

(2) $L := a_l$*,* $H := a_h$*, and* $H := a_l$*;*

(3) $P_0; P_1$*;*

(4) $\texttt{if}(b_l)\ \{P_0\}\ \texttt{else}\ \{P_1\}$*;*

(5) $\texttt{if}(b_h)\ \{P_0\}\ \texttt{else}\ \{P_1\}$*, whenever* $P_0 \sim_l P_1$*;*

(6) $\texttt{while}(b_l)\ \{P_0\}$*;*

(7) $\texttt{await}(b_l)\ \{S\}$*;*

(8) $\texttt{co}\ P_0 \| P_1\ \texttt{oc}$.

□

Proof. We consider case (3), all the other cases are similar. We exploit the fact that $P \in \mathbf{SIMP}^*$ if and only if $P \sim_l P$. The thesis follows from the fact that if P, P', Q, Q' are such that $P \sim_l P'$ and $Q \sim_l Q'$, then $P; Q \sim_l P'; Q'$. To prove this it is sufficient to show that

$$\mathcal{S} = \{(P; Q, P'; Q') \mid P \sim_l P' \text{ and } Q \sim_l Q'\} \cup \{(P, Q) \mid P \sim_l Q\}$$

is a strong low level bisimulation. In fact, let $\sigma =_l \theta$.

If $\langle P; Q, \sigma\rangle \to \langle P_1; Q, \sigma_1\rangle$, then $\langle P, \sigma\rangle \to \langle P_1, \sigma_1\rangle$. Hence, since $P \sim_l P'$, we get that $\langle P', \theta\rangle \to \langle P_1', \theta_1\rangle$ with $P_1 \sim_l P_1'$ and $\sigma_1 =_l \theta_1$. From this last we get $\langle P'; Q', \theta\rangle \to \langle P_1'; Q', \theta_1\rangle$ with $(P_1; Q, P_1'; Q') \in \mathcal{S}$.

If $\langle P; Q, \sigma\rangle \to \langle Q, \sigma_1\rangle$, then $\langle P, \sigma\rangle \to \langle \texttt{end}, \sigma_1\rangle$. Hence, since $P \sim_l P'$, we get that $\langle P', \theta\rangle \to \langle P_1', \theta_1\rangle$ with $\texttt{end} \sim_l P_1'$ and $\sigma_1 =_l \theta_1$. Therefore, $P_1' \equiv \texttt{end}$. From this last we get $\langle P'; Q', \theta\rangle \to \langle Q', \theta_1\rangle$ with $(Q, Q') \in \mathcal{S}$. □

Notice that Theorem 3.21 does not provide a procedure to decide whether $P \in \mathbf{SIMP}^*$. This is due to the request $P_0 \sim_l P_1$ in item (5) which is undecidable. Moreover, a program P could be in $\mathbf{SIMP}^*$ even if it does not satisfy the conditions of Theorem 3.21.

EXAMPLE 3.22. Consider

$$P \equiv \texttt{if}(L = 1)\ \{L := H - H\}\ \texttt{else}\ \{L := 2\}.$$

The program $L := H - H$ is in $\mathbf{SIMP}^*$, since for each state σ it holds $\langle L := H - H, \sigma\rangle \to \langle \texttt{end}, \sigma[L/0]\rangle$. However it does not satisfy any of the conditions of Theorem 3.21. As a consequence, by applying Theorem 3.21 we cannot prove that $P \in \mathbf{SIMP}^*$.

Let now $P \equiv \texttt{await}(0 = 0)\ \{L := H; L := 1\}$. For each state σ it holds $\langle P, \sigma\rangle \to \langle \texttt{end}, \sigma[L/1]\rangle$. Hence, P is in $\mathbf{SIMP}^*$. However, we cannot use Theorem 3.21 to conclude this fact since $S \equiv L := H; L := 1$ does not satisfy Theorem 3.21. □

In the next section we exploit the decidability of first-order formulae over the reals to get sound and accurate (but not complete) proof systems both for $\sim_l$ and for $\mathbf{SIMP}^*$.

The following theorem shows that all the security properties introduced above imply a lockstep noninterference principle.

Theorem 3.23. (Soundness) *Let P be a program such that $P \in \mathbf{SIMP}_{\approx_l}$ (respectively, $\mathbf{SIMP}^*_{\approx_l}$, $\mathbf{SIMP}^*$). For each state σ and θ such that $\sigma =_l \theta$,*

$$\langle P, \sigma\rangle \to^n \langle \texttt{end}, \sigma'\rangle \text{ if and only if } \langle P, \theta\rangle \to^n \langle \texttt{end}, \theta'\rangle \text{ with } \sigma' =_l \theta'.$$

□

Proof. It is sufficient to prove the theorem for $P \in \mathbf{SIMP}_{\approx_l}$. Then the thesis follows from the fact that $\mathbf{SIMP}^* \subseteq \mathbf{SIMP}^*_{\approx_l} \subseteq \mathbf{SIMP}_{\approx_l}$.

By Lemma 3.10, since $\sigma =_l \theta$, we have that $\langle P, \sigma\rangle \approx_l \langle P, \theta\rangle$. Then, by Lemma 2.12, we get that $\langle P, \theta\rangle \to^n \langle P', \theta'\rangle$ with $\langle P', \theta'\rangle \approx_l \langle \texttt{end}, \sigma'\rangle$. Hence we immediately have $\sigma' =_l \theta'$. Moreover, since end is not bisimilar to any program different from end, it must be $P' \equiv \texttt{end}$. □

We conclude this section with a mutual exclusion example.

EXAMPLE 3.24. Consider the programs

$P_0 \equiv$ `while`(true) {
  `await`($M = 0$) {`skip`};
  $L := H_1 + H_2$;
  $H_1 := H_1 - H_2$;
  $H_2 := L$;
  $L := 0$;
  $M := 1$
}

$P_1 \equiv$ `while`(true) {
  `await`($M = 1$) {`skip`};
  $L := H_1 * H_2$;
  $H_1 := H_1 + H_2$;
  $H_2 := L$;
  $L := 0$;
  $M := 0$
}

where M is a low level variable. Let $P \equiv \texttt{co}\ P_0 \| P_1\ \texttt{oc}$. The two programs P_0 and P_1 alternatively exploit the low level variable L to temporary store high level information. They write on L, H_1, H_2 in mutual exclusion. In fact, the `await` condition on

the variable M ensures that the subsequent writing operations cannot be interleaved. However, the program P does not belong to the class **SIMP***. In fact, the program $Q \equiv \texttt{co}\ L := H_1 + H_2; H_1 := H_1 - H_2; H_2 := L; L := 0; M := 1; P_0 \| P_1\ \texttt{oc}$ is in $Reach^*(P)$ and it is immediate to show that $Q \not\sim_l Q$. Intuitively, the point in which L is used to store high level information is in mutual exclusion between P_0 and P_1, but it is not included into an atomic sequence such as the body of the `await` operator, hence the high level information could be read by a low level user running a program in parallel with P. In order to ensure security we could modify P_0 and P_1 as follows

$$
\begin{array}{ll}
P_0' \equiv \texttt{while}(\text{true})\ \{ & P_1' \equiv \texttt{while}(\text{true})\ \{ \\
\quad \texttt{await}(M = 0)\ \{ & \quad \texttt{await}(M = 1)\ \{ \\
\quad\quad L := H_1 + H_2; & \quad\quad L := H_1 * H_2; \\
\quad\quad H_1 := H_1 - H_2; & \quad\quad H_1 := H_1 + H_2; \\
\quad\quad H_2 := L; & \quad\quad H_2 := L; \\
\quad\quad L := 0; & \quad\quad L := 0; \\
\quad \} & \quad \} \\
\quad M := 1; & \quad M := 0; \\
\} & \}
\end{array}
$$

Now if we consider the program $P' \equiv \texttt{co}\ P_0' \| P_1'\ \texttt{oc}$ then we can prove that $P' \in$ **SIMP*** (see next section). In fact, the low level user cannot observe L inside the `await` statements and outside them L does not depend on the high level values. □

4 Verification Techniques

In general, it is difficult to decide whether a program belongs to an unwinding class. First of all, given a program P and a state σ, the LTS associated to $\langle P, \sigma \rangle$ could be infinite.

EXAMPLE 4.1. Consider the program

$$P \equiv\ \texttt{while}(1 = 1)\ \{L := L + 1\}$$

and a state σ such that $\sigma(L) = 1$. The LTS associated to $\langle P, \sigma \rangle$ consists of an infinite chain of transitions to different pairs:

$$\langle P, \sigma \rangle \to \langle L := L + 1; P, \sigma \rangle \to \langle P, \sigma[L/2] \rangle \to \ldots \to \langle P, \sigma[L/n] \rangle \to \ldots$$

□

Another difficulty arises from the fact that even if we restrict ourselves to terminating programs, the relation $\sim_l \subseteq (\mathbb{P})^2$ is not decidable.

Lemma 4.2. *The relation $\sim_l \subseteq (\mathbb{P})^2$ is undecidable.* □

Proof. A diophantine equation is an equation *deq* of the form $p(X_1, \ldots, X_n) = 0$, where p is a polynomial with integer coefficients. The 10th Hilbert Problem over a diophantine equation *deq*, which consists in deciding whether *deq* has integer solutions, has

been proved to be undecidable [19]. We prove that given an arbitrary diophantine equation deq we can reduce the 10th Hilbert Problem over deq to the problem $P_{deq} \sim_l P_{deq}$ for an opportune program P_{deq}. This is suffi cient to prove that $\sim_l$ is undecidable.

Let $deq \equiv p(X_1,\ldots,X_n) = 0$ be a diophantine equation. Consider the program defi ned as

$$P_{deq} \equiv \texttt{if}(p(X_1,\ldots,X_n) = 0)\ \{L := H\}\ \texttt{else}\ \{\texttt{skip}\}$$

where $X_1,\ldots,X_n,L$ are low level variables and H is a high level variable. P_{deq} is a program, since $p(X_1,\ldots,X_n)$ is an arithmetic expression of our language. If $P_{deq} \sim_l P_{deq}$, then $L := H$ is not in $Reach^*(P_{deq})$ which implies that there do not exist $x_1,\ldots,x_n \in \mathbb{Z}$ such that $p(X_1/x_1,\ldots,X_n/x_n) = 0$ is true, i.e., deq does not admit integer solutions. On the other hand if deq does not admit integer solutions, then there does not exist a state σ such that $\langle p(X_1,\ldots,X_n) = 0, \sigma\rangle \rightarrow$ true, hence $L := H$ is not in $Reach^*(P_{deq})$ and $P_{deq} \sim_l P_{deq}$ is true. □

The above result is a direct consequence of the undecidability of the equational theory over integer with multiplication. In order to cope with this problem, we abstract integers with reals and exploit the Tarski's decidability result for fi rst-order formulae over the reals [35] to defi ne a decidable binary relation $\backsimeq_l$ over programs which entails $\sim_l$. Let $\mathbb{R}$ be the set of real numbers. A *real state* is a function $\sigma^r : \mathbb{L} \cup \mathbb{H} \longrightarrow \mathbb{R}$, i.e., a state in which the variables range over the reals. Two real states σ^r and θ^r are low level equivalent, $\sigma^r =_l \theta^r$, if they assign the same values to the low level variables. We start by defi ning a decidable binary relation $\backsimeq_l$ over expressions and sequences of assignments.

Definition 4.3. **($\backsimeq_l$ over Aexp $\cup$ Bexp $\cup\ \mathbb{P}$)** Let $a_1, a_2 \in$ **Aexp**. We say that a_1 and a_2 are *low level equivalent*, denoted by $a_1 \backsimeq_l a_2$, if for all real states σ^r, θ^r such that $\sigma^r =_l \theta^r$ it holds

$$\langle a_1, \sigma^r\rangle \rightarrow r \quad \text{if and only if} \quad \langle a_2, \theta^r\rangle \rightarrow r.$$

Let $b_1, b_2 \in$ **Bexp**. We say that b_1 and b_2 are *low level equivalent*, denoted by $b_1 \backsimeq_l b_2$, if for all real states σ^r, θ^r such that $\sigma^r =_l \theta^r$ it holds

$$\langle b_1, \sigma^r\rangle \rightarrow v \quad \text{if and only if} \quad \langle b_2, \theta^r\rangle \rightarrow v.$$

Let $\texttt{await}(\text{true})\{S_1\}$ and $\texttt{await}(\text{true})\{S_2\}$ be two programs. We say that S_1 and S_2 are *low level equivalent*, denoted by $S_1 \backsimeq_l S_2$, if for all real states σ^r, θ^r such that $\sigma^r =_l \theta^r$ it holds

$$\langle S_1, \sigma^r\rangle \rightsquigarrow \langle \texttt{end}, \sigma_1^r\rangle \text{ if and only if } \langle S_2, \theta^r\rangle \rightsquigarrow \langle \texttt{end}, \theta_1^r\rangle, \text{ where } \sigma_1^r =_l \theta_1^r.$$

□

Notice that the relation $\backsimeq_l$ is symmetric but not reflexive. Moreover, we could avoid to use the `await` statement before S_1 and S_2 by simply specifi ng that they are two sequential programs.

EXAMPLE 4.4. Consider the expression $a \equiv H+1$. It does not hold that $a \simeq_l a$. In fact, if we consider the states σ^r and θ^r such that $\sigma^r =_l \theta^r$ but $\sigma(H) = 1$ while $\theta(H) = 2$, then we obtain $\langle a, \sigma^r \rangle \to 2$ and $\langle a, \theta^r \rangle \to 3$.

Consider the program $\texttt{await}(\text{true})\{S\}$, where

$$S \equiv L := H_1 + H_2; H_1 := H_1 - H_2; H_2 := L; L := 0.$$

We have that $S \simeq_l S$. In fact, for each σ^r and θ^r such that $\sigma^r =_l \theta^r$, we get that $\langle S, \sigma^r \rangle \rightsquigarrow \langle \texttt{end}, \sigma_1^r \rangle$ and $\langle S, \theta^r \rangle \rightsquigarrow \langle \texttt{end}, \theta_1^r \rangle$ with $\sigma_1^r(L) = 0 = \theta_1^r(L)$, i.e., $\sigma_1^r =_l \theta_1^r$. □

As a consequence of the fact that states are a subset of real states and low level equivalence over states is coherent with low level equivalence over real states we get the following result.

Lemma 4.5. *Let $a_1, a_2 \in$ **Aexp**, $b_1, b_2 \in$ **Bexp**, and $S_1, S_2 \in \mathbb{P}$ such that $a_1 \simeq_l a_2$, $b_1 \simeq_l b_2$, and $S_1 \simeq_l S_2$. If σ and θ are two states (over the integers) such that $\sigma =_l \theta$, then:*

- *$\langle a_1, \sigma \rangle \to n$ if and only if $\langle a_2, \theta \rangle \to n$;*
- *$\langle b_1, \sigma \rangle \to v$ if and only if $\langle b_2, \theta \rangle \to v$;*
- *$\langle S_1, \sigma \rangle \rightsquigarrow \langle \texttt{end}, \sigma_1 \rangle$ if and only if $\langle S_2, \theta \rangle \rightsquigarrow \langle \texttt{end}, \theta_1 \rangle$, where $\sigma_1 =_l \theta_1$.*

□

The converse of Lemma 4.5 is not true, i.e., it can be the case that two expressions are not $\simeq_l$-equivalent, even if they are always mapped to the same value by two integer states which are low level equivalent.

EXAMPLE 4.6. Let b be $H^2 \neq 2$. It does not hold $b \simeq_l b$. In fact, over the reals this expression is equivalent to $H \neq \pm\sqrt{2}$ which can be either true or false depending on the value of H. However, over the integers this expression is always true, i.e. for each state σ it holds $\langle b, \sigma \rangle \to \text{true}$. □

Lemma 4.7. *The relation $\simeq_l$ is decidable.* □

Proof. Let $X_1, \ldots, X_n$ ($X'_1, \ldots, X_n$) be variables, we use the notation $\overline{X}$ ($\overline{X'}$) as a shorthand for $X_1, \ldots, X_n$ ($X'_1, \ldots, X_n$, respectively).

Let $a_1, a_2 \in$ **Aexp** be two arithmetic expressions. Let $\overline{L}$ and $\overline{H}$ be the low and high level variables, respectively, occurring in $a_1 + a_2$. We use $a[\overline{X}]$ to denote the fact that a could contain the variables $\overline{X}$. We use the notation $a[\overline{Z}, \overline{X}/\overline{Y}]$ to refer to the expression obtained by replacing in a the variables $\overline{X}$ with $\overline{Y}$. We have that the validity of $a_1 \simeq_l a_2$ is equivalent to the validity of the following first-order formula over the reals

$$\forall \overline{L}, \overline{H}, \overline{K}(a_1[\overline{L}, \overline{H}] = a_2[\overline{L}, \overline{H}/\overline{K}])$$

As a consequence of the decidability of the first-order theory of real numbers (see [35]) we get the thesis in the case of arithmetic expressions.

Similarly given two boolean expressions $b_1, b_2 \in \mathbf{Bexp}$ we have that the validity of $b_1 \backsimeq_l b_2$ is equivalent to the validity of the first-order formula over the reals

$$\forall \overline{L}, \overline{H}, \overline{K}(b_1[\overline{L}, \overline{H}] \leftrightarrow b_2[\overline{L}, \overline{H}/\overline{K}])$$

Hence, exploiting again the result in [35] we get the thesis also in the case of boolean expressions.

Let $\mathtt{await}(\text{true})\ \{S\}$ be a program and T be a subprogram of S. Let $X_1, \ldots, X_n$ be the variables occurring in S and let $X'_1, \ldots, X'_n$ be n new variables. We define the formula $\psi_T[\overline{X}, \overline{X'}]$ by structural induction on T. Intuitively, $\overline{X}$ denotes the input values while $\overline{X'}$ are the outputs after the execution of T. The other variables are used for temporary values. If $T \equiv \mathtt{skip}$, then $\psi_T[\overline{X}, \overline{X'}]$ is the formula $\wedge_{i=1}^{n}(X'_i = X_i)$. If $T \equiv X_j = a_j[X_1, \ldots, X_n]$, then $\psi_T[\overline{X}, \overline{X'}]$ is the formula $X'_j = a_j[X_1, \ldots, X_n] \wedge_{i=1, i \neq j}^{n}(X'_i = X_i)$. If $T \equiv T_1; T_2$, then $\psi_T[\overline{X}, \overline{X'}]$ is the formula $\exists X''_1, \ldots, X''_n(\psi_{T_1}[\overline{X}, \overline{X'}/\overline{X''}] \wedge \psi_{T_2}[\overline{X}/\overline{X''}, \overline{X'}])$. Consider now two programs $\mathtt{await}(\text{true})\ \{S_1\}$ and $\mathtt{await}(\text{true})\ \{S_2\}$. Let $\overline{L}$ ($\overline{H}$) be the low level variables (high level variables, respectively) occurring in S_1 and S_2. The validity of $S_1 \backsimeq_l S_2$ is equivalent to the validity of the first-order formula over the reals

$$\forall \overline{L}, \overline{L'}, \overline{M'}, \overline{H}, \overline{H'}, \overline{K}, \overline{K'} \\ ((\psi_{S_1}[\overline{L}, \overline{H}, \overline{L'}, \overline{H'}] \wedge \psi_{S_2}[\overline{L}, \overline{H}/\overline{K}, \overline{L'}/\overline{M'}, \overline{H'}/\overline{K'}]) \rightarrow \overline{L'} = \overline{M'})$$

Hence we get the thesis. □

Based on the definition of $\backsimeq_l$ over expressions and sequences of assignments, we define the binary relation $\Rrightarrow_l$ over $\mathbb{P}$ which provides a decidable approximation of $\sim_l$. Given a program P we denote by $l(P)$ the number of operators occurring in P, i.e., $l(\mathtt{end}) = l(\mathtt{skip}) = l(\mathtt{X:=a}) = 1$, $l(P_0; P_1) = l(\mathtt{co}\ P_0 \| P_1\ \mathtt{oc}) = l(P_0) + l(P_1) + 1$, $l(\mathtt{if}(b)\ \{P_0\}\ \mathtt{else}\ \{P_1\}) = l(P_0) + l(P_1) + 2$, $l(\mathtt{while}(b)\ \{P\}) = l(\mathtt{await}(b)\ \{P\}) = l(P) + 1$.

Definition 4.8. **($\Rrightarrow_l$ over $\mathbb{P}$)** The binary relation $\Rrightarrow_l$ over $\mathbb{P}$ is defined by the rules given in Table 2 and Table 3. □

Note that the last rule introduces a controlled form of symmetry. Note also that the given rules do not imply the transitivity of $\Rrightarrow_l$. Consider, for instance $P \equiv \mathtt{if}(1 = 1)\ \{P_0\}\ \mathtt{else}\ \{P_1\}$ and $Q \equiv \mathtt{while}(1 \neq 1)\ \{\mathtt{skip}\}; P_0$. We have that $P \Rrightarrow_1 \mathtt{skip}; P_0 \Rrightarrow_l Q$, but $P \not\backsimeq_l Q$. In order to guarantee the decidability of $\Rrightarrow_l$ we cannot simply add a rule for transitivity. In fact a rule of the form

$$\frac{P \Rrightarrow_l R \text{ and } R \Rrightarrow_l Q}{P \Rrightarrow_l Q}$$

without any condition on R, requires to look for R in the set of all programs which is infinite. We could instead enlarge the relation $\Rrightarrow_l$ as follows: if we have to test $P \Rrightarrow_l Q$ we first find all the subprograms of P and Q which are $\Rrightarrow$-equivalent with $\mathtt{skip}$; we replace them by $\mathtt{skip}$ in P and Q; we test $\Rrightarrow_l$ on the programs obtained in this way.

Moreover, the side conditions on the boolean expressions in Table 3 could be relaxed when the bodies of the $\mathtt{while}$ and $\mathtt{await}$ statements meet specific requirements (as in [7, 33]).

Lemma 4.9. *The relation $\precsim_l \subseteq \mathbb{P} \times \mathbb{P}$ is decidable and it entails the relation $\sim_l$.* □

Proof. The fact that the relation $\precsim_l \subseteq \mathbb{P} \times \mathbb{P}$ is decidable is an immediate consequence of Lemma 4.7 and of the fact that $\precsim_l$ is defined by structural induction on the syntax of the programs.

In order to prove that $\precsim_l$ entails the relation $\sim_l$ consider the binary relation $\mathcal{S}$ defined as

$$\mathcal{S} = \{(P,Q) \mid P \precsim_l Q\} \cup \{(P,Q) \mid P \sim_l Q\}$$

We prove that $\mathcal{S}$ is a strong low level bisimulation. Let $(P,Q) \in \mathcal{S}$. Clearly we need to consider only the case $P \precsim_l Q$. We proceed by structural induction on P.

Let σ, θ be two states such that $\sigma =_l \theta$.
Let $P \equiv \texttt{end}$ ($\texttt{skip}$). In this case $Q \equiv \texttt{end}$ ($\texttt{skip}$) hence we immediately get the thesis.
Let $P \equiv L := a_0$ with $L \in \mathbb{L}$. In this case we have $Q \equiv L := a_1$ with $a_0 \eqsim_l a_1$. Since $a_0 \eqsim_l a_1$, by Lemma 4.5, we have that $\langle a_0, \sigma\rangle \to n$ and $\langle a_1, \theta\rangle \to n$. Hence, it holds that $\langle P, \sigma\rangle \to \langle \texttt{end}, \sigma[L/n]\rangle$ and $\langle Q, \theta\rangle \to \langle \texttt{end}, \theta[L/n]\rangle$ with $\sigma[L/n] =_l \theta[L/n]$, i.e., the thesis.
Let $P \equiv H := a_0$ with $H \in \mathbb{H}$. In this case $Q \equiv K := a_1$ with $K \in \mathbb{H}$. Let $\langle a_0, \sigma\rangle \to n$ and $\langle a_1, \theta\rangle \to m$. It holds that $\langle P, \sigma\rangle \to \langle \texttt{end}, \sigma[H/n]\rangle$ and $\langle Q, \theta\rangle \to \langle \texttt{end}, \theta[K/m]\rangle$ with $\sigma[H/n] =_l \theta[K/m]$, i.e., the thesis.
Let $P \equiv P_0; P_1$. In this case we have $Q \equiv Q_0; Q_1$ with $P_0 \precsim_l Q_0$ and $P_1 \precsim_l Q_1$. If $\langle P_0, \sigma\rangle \to \langle \texttt{end}, \sigma'\rangle$, then $\langle P, \sigma\rangle \to \langle P_1, \sigma'\rangle$. By inductive hypothesis on P_0 and Q_0 we have that $\langle Q_0, \sigma\rangle \to \langle \texttt{end}, \theta'\rangle$ with $\sigma' =_l \theta'$. Hence $\langle Q, \theta\rangle \to \langle Q_1, \theta'\rangle$ and $(P_1, Q_1) \in \mathcal{S}$, i.e., the thesis. If $\langle P_0, \sigma\rangle \to \langle P_0', \sigma'\rangle$ with $P_0' \not\equiv \texttt{end}$, then $\langle P, \sigma\rangle \to \langle P_0'; P_1, \sigma'\rangle$. By inductive hypothesis on P_0 and Q_0 we have that $\langle Q_0, \sigma\rangle \to \langle Q_0', \theta'\rangle$ with $(P_0', Q_0') \in \mathcal{S}$ and $\sigma' =_l \theta'$. Hence $\langle Q, \theta\rangle \to \langle Q_0'; Q_1, \theta'\rangle$ and $(P_0'; P_1, Q_0'; Q_1) \in \mathcal{S}$, i.e., the thesis.
Let $P \equiv \texttt{co}\ P_0 \| P_1\ \texttt{oc}$. This case is similar to the previous one.
Let $P \equiv \texttt{if}(b_0)\ \{P_0\}\ \texttt{else}\ \{P_1\}$. Let $Q \equiv \texttt{if}(b_1)\ \{Q_0\}\ \texttt{else}\ \{Q_1\}$ with $P_0 \precsim_l Q_0$, $P_1 \precsim_l Q_1$, and $b_0 \eqsim_l b_1$. Since $b_0 \eqsim_l b_1$, by Lemma 4.5, we have that $\langle b_0, \sigma\rangle \to v$ iff $\langle b_1, \theta\rangle \to v$. Let us assume that $\langle b_0, \sigma\rangle \to \text{true}$. We have $\langle P, \sigma\rangle \to \langle P_0, \sigma\rangle$ and $\langle Q, \theta\rangle \to \langle Q_0, \theta\rangle$ with $(P_0, Q_0) \in \mathcal{S}$, i.e., the thesis. The case $\langle b_0, \sigma\rangle \to \text{false}$ is similar. Let $Q \equiv \texttt{skip}; P_0$ with $b_0 \eqsim_l \text{true}$ and $P_0 \precsim_l P_0$. We have $\langle P, \sigma\rangle \to \langle P_0, \sigma\rangle$ and $\langle Q, \theta\rangle \to \langle P_0, \theta\rangle$. By inductive hypothesis we immediately get the thesis. The remaining cases are similar.
Let $P \equiv \texttt{while}(b_0)\ \{P_0\}$. Let $Q \equiv \texttt{while}(b_1)\ \{Q_0\}$ with $b_0 \eqsim_l b_1$ and $P_0 \precsim_l Q_0$. If $\langle P, \sigma\rangle \to \langle \texttt{end}, \sigma\rangle$, then $\langle b_0, \sigma\rangle \to \text{false}$. Hence, by Lemma 4.5, $\langle b_1, \theta\rangle \to \text{false}$ and $\langle Q, \theta\rangle \to \langle \texttt{end}, \theta\rangle$, i.e., we have the thesis. If $\langle P, \sigma\rangle \to \langle P_0; P, \sigma\rangle$, then $\langle b_0, \sigma\rangle \to \text{true}$. Hence $\langle b_1, \theta\rangle \to \text{true}$ and $\langle Q, \theta\rangle \to \langle Q_0; Q, \theta\rangle$, with $P_0; P \precsim_l Q_0; Q$, i.e., with $(P_0; P, Q_0; Q) \in \mathcal{S}$. The remaining cases are similar.

Let $P \equiv \texttt{await}(b_0)\{S_0\}$. Let $Q \equiv \texttt{await}(b_1)\{S_1\}$ with $b_0 \eqsim_l b_1$ and $S_0 \eqsim_l S_1$. If $\langle b_0, \sigma\rangle \to \text{false}$, then $\langle b_1, \theta\rangle \to \text{false}$. In this case both $\langle P, \sigma\rangle \to \langle P, \sigma\rangle$ and $\langle Q, \theta\rangle \to \langle Q, \theta\rangle$ with $(P, Q) \in \mathcal{S}$. If $\langle b_0, \sigma\rangle \to \text{true}$, then $\langle b_1, \theta\rangle \to \text{true}$. Hence $\langle P, \sigma\rangle \to \langle \texttt{end}, \sigma'\rangle$ with $\langle S_0, \sigma\rangle \rightsquigarrow \langle \texttt{end}, \sigma'\rangle$. Since $S_0 \eqsim_l S_1$, by Lemma 4.5, we have that $\langle S_1, \theta\rangle \rightsquigarrow \langle \texttt{end}, \theta'\rangle$ with $\sigma' =_l \theta'$. Hence $\langle Q, \theta\rangle \to \langle \texttt{end}, \theta'\rangle$ and we get the thesis. □

As a consequence of Theorem 3.18, we can exploit the proof system for $\precsim_l$ to check if a program is in **SIMP***.

Theorem 4.10. *If $P \precsim_l P$ then $P \in \mathbf{SIMP}^*$.* □

EXAMPLE 4.11. Let the program

$$P \equiv \ \texttt{if}(H = 0)\ \{L := 1; H := 1\}\ \texttt{else}\ \{L := 1; H := 2\}.$$

By applying the proof system for $\precsim_l$ defined above, one can easily check that $P \precsim_l P$, and then $P \in \mathbf{SIMP}^*$.

Consider the program

$$P \equiv \ \texttt{while}(L^2 + H \geq H)\ \{L := 1\}.$$

By applying the Tarski's decidability result for first-order formulae over the reals, we can prove that $(L^2 + H \geq H) \precsim_l (L^2 + H \geq H)$. Hence, since $L := 1 \precsim_l L := 1$, by using our proof system one can derive that $P \precsim_l P$, i.e., $P \in \mathbf{SIMP}^*$. □

The decidability of $\eqsim_l$, and then of $\precsim_l$, depends on the Tarski's result on quantifier elimination. However, the result in [35] is mainly of theoretical interest. More efficient techniques to deal with formulae over the reals have been later developed and integrated in systems for automatic computations. Hong developed the first practical quantifier elimination software, named `Qepcad` [9]. A quantifier elimination procedure based on Collins' algorithm [8] and called `Cylindrical_Decomposition` has been integrated in `Mathematica` starting from Version 5.0. Since the formulae used to prove $\eqsim_l$ have no quantifier alternations (see proof of Lemma 4.7), the symbolic computation module of `Maple` (see `http://www.maplesoft.com/products/maple/index.aspx`) integrated in `Matlab` Version 5 is sufficient for our purposes.

In order to analyze the time complexity of $\precsim_l$, we denote by $c(P)$ the time complexity of evaluating the $\eqsim_l$ equivalence on the expressions and sequences of assignments occurring in P. In general $c(P)$ is a function of the maximum number v of variables which occurs in each expression of P, the degree d of the polynomials, and the quantifier alternations q, and it strongly depends on the algorithm used to check the first-order formulae over the reals. Since the formulae we use to prove $\eqsim_l$ are closed and have only universal quantifications, the number q of quantifier alternations in our analysis is always zero. So, the methods proposed by Grigoriev [15] and Renegar [25] in our case are polynomial in d and exponential in v.

Theorem 4.12. *Let P be a program. The complexity of deciding $P \precsim_l P$ is $O(c(P) * l(P))$.* □

Proof. In the worst case at each step two rules which require a checking of the form $b_0 \eqsim_l b_1$ are applicable. Moreover, at each step we apply only the first matching rule and $l(P)$ decreases by at least 1. □

The proof system for $\precsim_l$ defined above is quite involved since it has been designed to decide if $P_0 \eqsim_l P_1$ for any pair of programs P_0 and P_1. However, to check whether a program P belongs to the class $\mathbf{SIMP}^*$, it is sufficient to verify if $P \precsim_l P$. Below, we exploit the unwinding characterization of $\mathbf{SIMP}^*$ to specialize some of the rules and to reduce the number of checks. In particular, we introduce a decidable class $\mathcal{W}(\precsim_l)$ of secure programs such that $P \in \mathcal{W}(\precsim_l)$ if and only if $P \precsim_l P$. The class $\mathcal{W}(\precsim_l)$ is

defined through a proof system which can be used to incrementally build programs which are secure by construction.

Definition 4.13. **($\mathcal{W}(\approx_l)$)** The class $\mathcal{W}(\approx_l)$ is defined by the rules given in Table 4. □

Lemma 4.14. *The class $\mathcal{W}(\approx_l) \subseteq \mathbb{P}$ is decidable. Moreover, $P \in \mathcal{W}(\approx_l)$ if and only if $P \approx_l P$.* □

Proof. The fact that $\mathcal{W}(\approx_l) \subseteq \mathbb{P}$ is decidable is an immediate consequence of Lemma 4.7, Lemma 4.9, and of the fact that $\mathcal{W}(\approx_l)$ is defined by structural induction on the syntax of the programs.

It is easy to prove that if $a, b \in \texttt{low}$, then $a \backsimeq_l a$ and $b \backsimeq_l b$. Hence, if $P \in \mathcal{W}(\approx_l)$, then it holds $P \approx_l P$, since in Table 4 we only instantiate some of the rules of $\approx_l$. On the other hand, if $P \approx_l P$, then the only rules which can have been applied are that which occur in Table 4, hence $P \in \mathcal{W}(\approx_l)$. □

As a consequence, we get the following result.

Theorem 4.15. *If $P \in \mathcal{W}(\approx_l)$, then $P \in \mathbf{SIMP}^*$.* □

EXAMPLE 4.16. Let P' be the program of Example 3.24. We can use our proof system to show that P' is in $\mathbf{SIMP}^*$. First, we have to prove that both P'_0 and P'_1 are in $\mathcal{W}(\approx_l)$. Let us consider P'_0. It has the form $\texttt{while}(\text{true})\{\texttt{await}(M=0)\{S\}; M := 1\}$. By using the second rule for the `while` statement we get that we have to prove that $\texttt{await}(M = 0)\{S\}; M := 1$ is in $\mathcal{W}(\approx_l)$. By applying the rule for sequential composition we need to prove that both $\texttt{await}(M=0)\{S\}$ and $M := 1$ are in $\mathcal{W}(\approx_l)$. The second proof is immediate. As far as $\texttt{await}(M=0)\{S\}$ is concerned, we need to prove the side conditions of the `await` rule, i.e., $(M=0) \backsimeq_l (M=0)$ and $S \backsimeq_l S$. Since M is a low level variable, the first condition is satisfied. The second one has been proved in Example 4.4. We can conclude that $P'_0 \in \mathcal{W}(\approx_l)$. Similarly, the proof system allows us to prove that $P'_1 \in \mathcal{W}(\approx_l)$. Finally, by applying the rule for parallel composition we obtain that $P' \in \mathcal{W}(\approx_l)$. □

The following example shows that there are programs which are in $\mathbf{SIMP}^*$ but not in $\mathcal{W}(\approx_l)$.

EXAMPLE 4.17. Let

$$P \equiv \texttt{while}(\text{true})\ \{L := 1\};\ L := H$$

Since $L := H$ is not in $\mathit{Reach}^*(P)$ we get that $P \in \mathbf{SIMP}^*$. However, we cannot prove it using our proof system. In fact, the rule for $P_0; P_1$ requires that both P_0 and P_1 be secure independently of their reachability.

Consider now the program

$$P \equiv\ \texttt{if}(H^2 \neq 2)\ \{L := 1\}\ \texttt{else}\ \{L := 2\}$$

It holds that $H^2 \neq 2$ is always true over the integers, hence it follows that $P \in \mathbf{SIMP}^*$. However, $(H^2 \neq 2) \not\backsimeq_l (H^2 \neq 2)$ and $L := 1$ is not low level equivalent to $L := 2$, hence $P \notin \mathcal{W}(\approx_l)$. □

5 Delimited Information Release

In the previous sections we have presented a method for specifying and verifying security properties of programs which prevent any flow of information from high to low level locations. However, as observed by many authors, e.g., [17, 29, 41], noninterference is too strong for practical applications. Indeed, many realistic programs do allow some release, or *declassification*, of secret information (e.g., password checking, information purchase, and spreadsheet computation). In this section we show how our generalized unwinding condition can be instantiated in order to obtain security properties for concurrent programs which intentionally release some information. We also extend the proof systems of previous section to the analysis of such properties.

We consider a finite set D of arithmetic and boolean expressions which are constructed by using only high level variables. The set D represents the set of all the high level expressions which can be declassified during the execution.

Definition 5.1. A set D of arithmetic and boolean expressions is said to be *declassifiable* if it is finite and all the expressions in it contain only high level variables. □

EXAMPLE 5.2. Let $D = \{H_1 > 5, H_1 + H_2\}$. D is a declassifiable set. Intuitively, it represents the fact that, concerning the values of the secret variables H_1 and H_2, a low level user is allowed to know whether H_1 is greater than 5 or not, as well as the total value of the sum $H_1 + H_2$. Hence, the program

$$P \equiv \texttt{if}(H_1 > 5)\ \{\texttt{skip}\}\ \texttt{else}\ \{L := H_1 + H_2\}$$

should be considered secure.

Notice that, the user can observe if $H_1 \leq 5$ and the sum $H_1 + H_2$. Thus the user can infer $H_2 \geq \ell - 5$, where ℓ is the value of L at the end of the execution. Hence the information which is downgraded through the set D exceeds D itself. □

The previous example shows that not only the information in the set D but any information obtained by combining elements of D is downgraded to the low level user. Intuitively, D represents a finite abstraction of all information that is actually downgraded. We define the concretization $\gamma(D)$ of D representing all the declassifiable expressions which are deducible from D.

Definition 5.3. Let D be a declassifiable set. The *concretization* $\gamma(D)$ *of* D is the smallest set such that:

(1) $D \subseteq \gamma(D)$,

(2) if $e, e' \in \mathbf{Aexp} \cup \mathbf{Bexp}$, $e' \in \gamma(D)$ and for all states σ, $\langle e, \sigma\rangle \rightarrow v$ if and only if $\langle e', \sigma\rangle \rightarrow v$, then $e \in \gamma(D)$,

(3) if e is defined through the grammar for $\mathbf{Aexp} \cup \mathbf{Bexp}$ given in Section 2 with $a_0, a_1, b, b_0, b_1, X \in \gamma(D)$ then $e \in \gamma(D)$.

□

EXAMPLE 5.4. Consier the declassifiable set D of Example 5.2. We have that, e.g., $2H_1 - 10 > 0$ and $H_2 + H_1 + 1$ belong to $\gamma(D)$, while $H_1, H_2 \notin \gamma(D)$. □

By Definition 5.3 it follows that for all set D' such that $D \subseteq D' \subseteq \gamma(D)$, it holds $\gamma(D') = \gamma(D)$.

Our approach is in the spirit of [17, 29] in the sense that we require that only explicitly declassifiable data, i.e., those in $\gamma(D)$, but no further information is released. Notice that, differently from [29], we do not add an explicit *declassify* predicate to the syntax of expressions but instead we consider the set D representing all declassifiable expressions.

Let σ and θ be two states and D be a declassifiable set. We write $\sigma =_{l,D} \theta$ if $\sigma =_l \theta$ and $\langle d, \sigma \rangle \to c$ if and only if $\langle d, \theta \rangle \to c$, for all $d \in D$. By Definition 5.3, it follows that $\sigma =_{l,D} \theta$ if and only if $\sigma =_{l,\gamma(D)} \theta$. Moreover, the following technical property holds.

Lemma 5.5. *For each ψ and π such that $\psi =_{l,D} \pi$, if $\langle F, \psi \rangle \overset{\texttt{low}}{\to} \langle F', \psi' \rangle$, then $\langle F, \pi \rangle \overset{\texttt{low}}{\to} \langle F', \pi' \rangle$ with $\pi' =_{l,D} \psi'$.* □

Proof. This is a consequence of the fact that in D there are not low level variables. The proof follows by structural induction on programs. □

In order to deal with delimited release we introduce the notion of *strong low-D level bisimulation* which is obtained from Definition 2.7 by replacing $=_l$ with $=_{l,D}$. We say that two programs $P, Q \in \mathbb{P}$ are *strongly low-D level bisimilar*, denoted by $P \sim_{l,D} Q$ if there exists a low-D level bisimulation $\mathcal{B}$ such that $(P, Q) \in \mathcal{B}$.

Lemma 5.6. *If $P \sim_{l,D} P$, then for all $F \in Reach^*(P)$ it holds that $F \sim_{l,D} F$.* □

Proof. If $F \in Reach^*(P)$, then there exists $n \geq 0$ and $P_0, \ldots, P_n$, $\sigma_0, \ldots, \sigma_n$, $\theta_0, \ldots, \theta_n$ such that $P_0 \equiv P$, $P_n \equiv F$ and for each $1 \leq i \leq n$ it holds $\langle P_{i-1}, \sigma_{i-1} \rangle \to \langle P_i, \theta_i \rangle$. We prove that $P_n \sim_{l,D} P_n$ by induction on n.

Base. If $n = 0$, then $P_n \equiv P$, hence we immediately get the thesis.

Inductive step. Suppose that the thesis holds for $n = m$ and consider $n = m+1$. We have that $\langle P_m, \sigma_m \rangle \to \langle P_{m+1}, \theta_{m+1} \rangle$. Since by inductive hypothesis it holds $P_m \sim_{l,D} P_m$ and it holds that $\sigma_m =_{l,D} \sigma_m$ and $\langle P_m, \sigma_m \rangle \to \langle P_{m+1}, \theta_{m+1} \rangle$ we get that $\langle P_m, \sigma_m \rangle \to \langle Q, \mu \rangle$ with $P_{m+1} \sim_{l,D} Q$ and $\theta_{m+1} =_{l,D} \mu$. Since $\sim_{l,D}$ is a partial equivalence relation from $P_{m+1} \sim_{l,D} Q$ we get $P_{m+1} \sim_{l,D} P_{m+1}$. □

Moreover, the relation $\ddot{\sim}_{l,D}$ over $\mathbb{P} \times \Sigma$ is defined as follows:

Definition 5.7. Let D be a declassifiable set. The relation $\ddot{\sim}_{l,D}$ over $\mathbb{P} \times \Sigma$ is defined by: $\langle P, \sigma \rangle \ddot{\sim}_{l,D} \langle Q, \theta \rangle$ if $\sigma =_{l,D} \theta$ and $P \sim_{l,D} Q$. □

The next technical property will be used in the following.

Lemma 5.8. *Let P and Q be two programs and σ and θ be two states. Let $\langle P, \sigma \rangle \ddot{\sim}_{l,D} \langle Q, \theta \rangle$. If $\langle P, \sigma \rangle \to^n \langle P', \sigma' \rangle$, then there exists Q' and θ' such that $\langle Q, \theta \rangle \to^n \langle Q', \theta' \rangle$ and $\langle P', \sigma' \rangle \ddot{\sim}_{l,D} \langle Q', \theta' \rangle$, and viceversa.* □

Proof. By induction on n.

- Base: $n = 1$. We immediately have the thesis by definition of $\ddot{\sim}_{l,D}$.

- Inductive step: $n = m+1$ and we proved the thesis for m. We have that $\langle P,\sigma\rangle \rightarrow^m \langle P'',\sigma''\rangle \rightarrow \langle P',\sigma'\rangle$. By inductive hypothesis we get $\langle Q,\theta\rangle \rightarrow^m \langle Q'',\theta''\rangle$ with $\langle P'',\sigma''\rangle \ddot{\sim}_{l,D} \langle Q'',\theta''\rangle$. By definition of strong low-D level bisimulation we get the thesis.

□

Lemma 5.9. *Let D be a declassifiable set. The relations $\sim_{l,D}$ and $\ddot{\sim}_{l,D}$ are partial equivalence relations.* □

Proof. The fact that $\sim_{l,D}$ is symmetric is a consequence of the fact that each strong low-D level bisimulation is symmetric. Moreover, $\sim_{l,D}$ is transitive since the composition of two strong low-D level bisimulations is still a strong low-D level bisimulation. □

We study the class of secure imperative programs $\mathbf{SIMP}^*_D$ which is obtained by instantiating our unwinding condition with the low-D level relations $=_{l,D}$ for $\doteq$ and $\ddot{\sim}_{l,D}$ for $\ddot{\div}$, and the relation $Reach^*$ for $\mathcal{R}$.

Definition 5.10. **($\mathbf{SIMP}^*_D$)** Let D be a declassifiable set. A program P belongs to the class $\mathbf{SIMP}^*_D$ if for each state σ, $\langle P,\sigma\rangle \in \mathcal{W}(=_{l,D}, \ddot{\sim}_{l,D}, Reach^*)$. □

EXAMPLE 5.11. Consider the program

$$P \equiv \texttt{if}(H=0)\ \{L:=0; H:=0\}\ \texttt{else}\ \{L:=1; H:=1\}$$

and the set $D = \{H=0\}$. In this case $P \in \mathbf{SIMP}^*_D$. In fact, for all states σ and θ such that $\sigma =_{l,D} \theta$, if $\langle P,\sigma\rangle \stackrel{\texttt{high}}{\rightarrow} \langle L:=0; H:=0,\sigma\rangle$ then also $\langle P,\theta\rangle \stackrel{\texttt{high}}{\rightarrow} \langle L:=0; H:=0,\theta\rangle$ and $\{L:=0; H:=0\} \sim_{l,D} \{L:=0; H:=0\}$. Analogously, if $\langle P,\sigma\rangle \stackrel{\texttt{high}}{\rightarrow} \langle L:=1; H:=1,\sigma\rangle$ then also $\langle P,\theta\rangle \stackrel{\texttt{high}}{\rightarrow} \langle L:=1; H:=1,\theta\rangle$ and $\{L:=1; H:=1\} \sim_{l,D} \{L:=1; H:=1\}$.

Consider now the program $P \equiv H_1 := H_2; L := H_1$ where H_1 and H_2 are high level variables and $D = \{H_1\}$. In this case $P \notin \mathbf{SIMP}^*_D$. In fact, given a state σ, $\langle P,\sigma\rangle \stackrel{\texttt{high}}{\rightarrow} \langle L := H_1, \sigma[H_1/H_2]\rangle$. However, it does not hold that for any θ such that $\sigma =_{l,D} \theta$, $\langle P,\theta\rangle \stackrel{\texttt{high}}{\rightarrow} \langle L := H_1, \theta[H_1/H_2]\rangle$ with $\sigma[H_1/H_2] =_{l,D} \theta[H_1/H_2]$. This happens whenever $\sigma(H_2) \neq \theta(H_2)$. Indeed, because of the assignment $H_1 := H_2$, after the execution of P, a low level user can infer the value of H_2 just by observing the value of L.

Observe that also the program $P \equiv L := H_1; H_1 := H_2$ with $D = \{H_1\}$ does not belong to $\mathbf{SIMP}^*_D$, since the second assignment assigns a high value to a downgraded variable. To understand why this is insecure we can consider the program $\texttt{co}\ P \| Q\ \texttt{oc}$ with $Q \equiv L_1 := H_1$. By observing L and L_1, a low level user can infer the value of the secret variable H_2; in fact whenever the values of L and L_1 are different, the value of L_1 is equal to that of H_2. □

The following property holds.

Lemma 5.12. *Let P be a program and D be a declassifiable set. If $P \in \mathbf{SIMP}^*_D$, then for all $P' \in Reach^*(P)$, $P' \in \mathbf{SIMP}^*_D$.* □

Proof. This proof is similar to that of Lemma 3.17. □

The next theorem follows from the definition of $\gamma(D)$ and shows that the set $\gamma(D)$ is exactly the information that can be released by a program P in $\mathbf{SIMP}^*_D$.

Theorem 5.13. *Let D be a declassifiable set. $P \in \mathbf{SIMP}^*_D$ if and only if $P \in \boldsymbol{SIMP}^*_{\gamma(D)}$.* □

Proof. It follows from the fact that by Definition 5.3, for all states σ and θ, $\sigma =_{l,D} \theta$ if and only if $\sigma =_{l,\gamma(D)} \theta$. □

The class $\mathbf{SIMP}^*_D$ satisfies compositional properties similar to those of Theorem 3.21 for the class $\mathbf{SIMP}^*$. In particular it is compositional with respect to the sequential and parallel operators. We can also prove that $\mathcal{W}(=_{l,D}, \ddot{\sim}_{l,D}, \mathit{Reach}^*)$ is persistent in the sense that if $\langle P,\sigma\rangle$ is in $\mathcal{W}(=_{l,D}, \ddot{\sim}_{l,D}, \mathit{Reach}^*)$, then also each pair $\langle P',\sigma'\rangle \in \mathit{Reach}^*(\langle P,\sigma\rangle)$ is in $\mathcal{W}(=_{l,D}, \ddot{\sim}_{l,D}, \mathit{Reach}^*)$. Moreover, if P is in $\mathbf{SIMP}^*_D$, then also each $P' \in \mathit{Reach}^*(P)$ is in $\mathbf{SIMP}^*_D$.

The class of programs P such that $P \sim_{l,D} P$ exactly coincides with the set of programs in the class $\mathbf{SIMP}^*_D$.

Theorem 5.14. *Let D be a declassifiable set. $P \in \mathbf{SIMP}^*_D$ if and only if $P \sim_{l,D} P$.* □

Proof. $\Rightarrow$) Consider the binary relation

$$\mathcal{S} = \{(P,P) \mid P \in \mathbf{SIMP}^*_D\} \cup \{(P,Q) \mid P \sim_{l,D} Q\}$$

We show that $\mathcal{S}$ is a strong D-low level bisimulation. This follows from the following cases. Let σ and θ be two states such that $\sigma =_{l,D} \theta$.

If $\langle P,\sigma\rangle \overset{\mathtt{high}}{\rightarrow} \langle P',\sigma'\rangle$, since $P \in \mathbf{SIMP}^*_D$, $\langle P,\sigma\rangle \in \mathcal{W}(=_{l,D}, \ddot{\sim}_{l,D}, \mathit{Reach}^*)$ and also $\langle P,\theta\rangle \rightarrow \langle P'',\theta'\rangle$ with $\langle P',\sigma'\rangle \ddot{\sim}_{l,D} \langle P'',\theta'\rangle$, i.e., $\sigma' =_{l,D} \theta'$ and $P' \sim_{l,D} P''$. Hence, by definition of $\mathcal{S}$, $(P',P'') \in \mathcal{S}$.

If $\langle P,\sigma\rangle \overset{\mathtt{low}}{\rightarrow} \langle P',\sigma'\rangle$, then by Lemma 5.5 we have that $\langle P,\theta\rangle \overset{\mathtt{low}}{\rightarrow} \langle P',\theta'\rangle$ with $\sigma' =_{l,D} \theta'$. By Lemma 5.12, we have that $P' \in \mathbf{SIMP}^*_D$ and then, by definition of $\mathcal{S}$, $(P',P') \in \mathcal{S}$, i.e., the thesis.

$\Leftarrow$) Let P be a program and $P \sim_{l,D} P$. Let $\sigma,\psi \in \Sigma$ and $\langle F,\psi\rangle \in \mathit{Reach}^*(\langle P,\sigma\rangle)$. Then $F \in \mathit{Reach}^*(P)$ and, by Lemma 5.6, $F \sim_{l,D} F$. Let π be such that $\psi =_{l,D} \pi$.

If $\langle F,\psi\rangle \overset{\mathtt{high}}{\rightarrow} \langle G,\varphi\rangle$, then, since $F \sim_{l,D} F$, $\langle F,\pi\rangle \rightarrow \langle R,\rho\rangle$ with $\varphi =_{l,D} \rho$ and $G \sim_{l,D} R$, i.e., the thesis. □

The next theorem shows that the family of security properties $\mathbf{SIMP}^*_D$ implies the *delimited release* property studied in [29] for sequential programs.

Theorem 5.15. (Soundness) *Let D be a declassifiable set and P be a program. If $P \in \mathbf{SIMP}^*_D$, then for all states σ and θ such that $\sigma =_{l,D} \theta$,*

$$\langle P,\sigma\rangle \rightarrow^n \langle \mathtt{end},\sigma'\rangle \text{ if and only if } \langle P,\theta\rangle \rightarrow^n \langle \mathtt{end},\theta'\rangle \text{ with } \sigma' =_l \theta'.$$

□

Proof. By Theorem 5.14, since $\sigma =_{l,D} \theta$, we have that $\langle P,\sigma\rangle \ddot{\sim}_{l,D} \langle P,\theta\rangle$. Then, by 5.8, we get that $\langle P,\theta\rangle \rightsquigarrow \langle P',\theta'\rangle$ with $\langle P',\theta'\rangle \ddot{\sim}_{l,D} \langle \mathtt{end},\sigma'\rangle$. Hence we immediately have $\sigma' =_{l,D} \theta'$. So by definition of $\ddot{=}_{l,D}$ we get $\sigma' =_l \theta'$. Moreover, since `end` is not strong low-D level bisimilar to any program, it must be $P' \equiv \mathtt{end}$. □

We can decide whether a program belongs to a class $\mathbf{SIMP}^*_D$ by extending the proof systems presented in Section 4 as described below.

First, by exploiting the Tarski decidability result for first-order formulae over the reals we define a decidable binary relation $\dot{=}_{l,D}$ over programs which entails $\sim_{l,D}$. The relation $=_{l,D}$ is extended to real states in the natural way.

Definition 5.16. **($\doteq_{l,D}$ over Aexp$\cup$Bexp$\cup\mathbb{P}$)** Let D be a declassifiable set. Let $a_1, a_2 \in$ **Aexp**. We say that a_1 and a_2 are *low-D level equivalent*, denoted by $a_1 \doteq_{l,D} a_2$, if for all real states σ^r, θ^r such that $\sigma^r =_{l,D} \theta^r$ it holds

$$\langle a_1,\sigma^r\rangle \to r \quad \text{if and only if} \quad \langle a_2,\theta^r\rangle \to r.$$

Let $b_1, b_2 \in$ **Bexp**. We say that b_1 and b_2 are *low-D level equivalent*, denoted by $b_1 \doteq_{l,D} b_2$, if for all σ^r, θ^r such that $\sigma^r =_{l,D} \theta^r$ it holds

$$\langle b_1,\sigma^r\rangle \to v \quad \text{if and only if} \quad \langle b_2,\theta^r\rangle \to v.$$

Let $\mathtt{await}(\text{true})\{S_1\}$ and $\mathtt{await}(\text{true})\{S_2\}$ be two programs. We say that S_1 and S_2 are *low-D level equivalent*, denoted by $S_1 \doteq_{l,D} S_2$, if for all real states σ^r, θ^r such that $\sigma^r =_{l,D} \theta^r$ it holds

$$\text{if } \langle S_1,\sigma^r\rangle \rightsquigarrow \langle \mathtt{end},\sigma^r_1\rangle \text{ and } \langle S_2,\theta^r\rangle \rightsquigarrow \langle \mathtt{end},\theta^r_1\rangle \text{ then } \sigma^r_1 =_{l,D} \theta^r_1.$$

□

Lemma 5.17. *Let $a_1, a_2 \in$ **Aexp**, $b_1, b_2 \in$ **Bexp**, and $S_1, S_2 \in \mathbb{P}$ such that $a_1 \doteq_l a_2$, $b_1 \doteq_l b_2$, and $S_1 \doteq_l S_2$. If σ and θ are two states (over the integers) such that $\sigma =_{l,D} \theta$, then:*

- *$\langle a_1,\sigma\rangle \to n$ if and only if $\langle a_2,\theta\rangle \to n$;*
- *$\langle b_1,\sigma\rangle \to v$ if and only if $\langle b_2,\theta\rangle \to v$;*
- *if $\langle S_1,\sigma\rangle \rightsquigarrow \langle \mathtt{end},\sigma_1\rangle$ and $\langle S_2,\theta\rangle \rightsquigarrow \langle \mathtt{end},\theta_1\rangle$ then $\sigma_1 =_{l,D} \theta_1$.*

□

Lemma 5.18. *Let D be a declassifiable set. The relation $\doteq_{l,D}$ is decidable.* □

Proof. Let $a_1, a_2 \in$ **Aexp** be two arithmetic expressions. Let $\overline{L}$ and $\overline{H}$ be the low and high level variables, respectively, occurring in a_1, a_2 and D. We consider the case in which in D there are only arithmetic expressions. We have that the validity of $a_1 \doteq_{l,D} a_2$ is equivalent to the validity of the following first-order formula over the reals

$$\forall \overline{L},\overline{H},\overline{K}(\textstyle\bigwedge_{d\in D} d[\overline{H}] = d[\overline{H}/\overline{K}]) \to (a_1[\overline{L},\overline{H}] = a_2[\overline{L},\overline{H}/\overline{K}])$$

As a consequence of the decidability of the first-order theory of real numbers we get the thesis. If D contains also boolean expressions we only have to replace $=$ with $\leftrightarrow$ on the boolean expressions.

The case of boolean expressions is similar.

Let $S_1, S_2 \in \mathbb{P}$. For any sequence of assignments S consider the formula $\psi_S[\overline{X}, \overline{X'}]$ as defined in the proof of Lemma 4.7. The validity of $S_1 \mathrel{\dot{=}}_{l,D} S_2$ is equivalent to the validity of the following first-order formula over the reals

$$\begin{array}{l}\forall \overline{L}, \overline{L'}, \overline{L''}, \overline{H}, \overline{K} \\ \quad ((\bigwedge_{d \in D} d[\overline{H}] = d[\overline{H}/\overline{K}] \wedge \psi_{S_1}[\overline{L}, \overline{H}, \overline{L'}, \overline{H'}] \wedge \psi_{S_2}[\overline{L}, \overline{H}/\overline{K}, \overline{L'}/\overline{L''}, \overline{H'}/\overline{K'}]) \rightarrow \\ \qquad\qquad (\overline{L'} = \overline{L''} \wedge \bigwedge_{d \in D} d[\overline{H}/\overline{H'}] = d[\overline{H}/\overline{K'}]))\end{array}$$

□

Lemma 5.19. *Let H and K be two high level locations and D be a declassifiable set. $H := a_0 \mathrel{\dot{=}}_{l,D} K := a_1$ entails $H := a_0 \sim_{l,D} K := a_1$.* □

Proof. This is an immediate consequence of Lemma 5.17. □

Definition 5.20. **($\dot{\approx}_{l,D}$ over $\mathbb{P}$)** Let D be a declassifiable set. The binary relation $\dot{\approx}_{l,D}$ over $\mathbb{P}$ is obtained from the rules given for $\dot{\approx}_l$ in Tables 2 and 3 by replacing $\dot{\approx}_l$ and $\mathrel{\dot{=}}_l$ with $\dot{\approx}_{l,D}$ and $\mathrel{\dot{=}}_{l,D}$, respectively and by adding the side condition $H := a_0 \mathrel{\dot{=}}_{l,D} K := a_1$ in the second rule for the assignment command. □

Lemma 5.21. *The relation $\dot{\approx}_{l,D} \subseteq \mathbb{P} \times \mathbb{P}$ is decidable and entails $\sim_{l,D}$.* □

Proof. The decidability is an immediate consequence of Lemma 5.18 and of the fact that $\dot{\approx}_{l,D}$ is defined by structural induction on the syntax of the programs.

The second part of the proof is similar to that of Lemma 4.9. The only case which is different is the case $H := a_0 \dot{\approx}_{l,D} K := a_1$ which now is a consequence of Lemma 5.19. □

We finally define the decidable class $\mathcal{W}(\dot{\approx}_{l,D})$ of secure programs by modifying our previous proof system as follows: the class $\mathcal{W}(\dot{\approx}_{l,D})$ is obtained from the rules given in Table 4 by replacing $\dot{\approx}_l$ and $\mathrel{\dot{=}}_l$ with $\dot{\approx}_{l,D}$ and $\mathrel{\dot{=}}_{l,D}$, respectively and by adding the side condition $H := a \mathrel{\dot{=}}_{l,D} H := a$ in the second line, second rule for the assignment command.

Theorem 5.22. *If $P \in \mathcal{W}(\dot{\approx}_{l,D})$, then $P \in \mathbf{SIMP}^*_D$.* □

Proof. This is similar to the proof of Lemma 4.14. □

EXAMPLE 5.23. Consider the program

$$P \equiv \texttt{if}(H = 0)\ \{L := 0; H := 0\}\ \texttt{else}\ \{L := 1; H := 1\}$$

and the set $D = \{H = 0\}$. It is easy to prove that both $\{L := 0; H := 0\} \in W(\dot{\approx}_{l,D})$ and $\{L := 1; H := 1\} \in W(\dot{\approx}_{l,D})$. Moreover, we can prove that $(H = 0) \dot{\approx}_{l,D} (H = 0)$. Hence, by applying the rules of $W(\dot{\approx}_{l,D})$, we get that $P \in W(\dot{\approx}_{l,D})$, i.e., $P \in \mathbf{SIMP}^*_D$. □

EXAMPLE 5.24. Let us consider a database in which low level users can only write (both on low and high level locations), while high level users can only read. In particular, the write operations are performed through assignments to global variables, denoted by L_{glob} and H_{glob}, while the read operations are performed through assignments to local variables, denoted by H_{loc}. Moreover, we want to ensure mutual exclusion among the low level users, and that high level users as a class exclude the low level ones (i.e., at any time either one low lever user or many high level users have access to the database and the high level users have priority on the low level ones).

As in [3], we use the standard semaphore notation: if S is a variable, $P(S)$ is a shorthand for $\texttt{await}(S > 0)\ \{S := S-1\}$ which is used to delay a process until the value of S is positive and then to decrement S; similarly, $V(S)$ stands for $S := S+1$ and is used to increment the value of the semaphore.

We can model each low level user with a program of the form

$$
\begin{array}{l}
\texttt{LowW}(a_l, a_h) \equiv \\
\quad \texttt{while}(\text{true})\ \{ \\
\qquad P(W); \qquad\qquad //\text{get lock to write} \\
\qquad L_{glob} := a_l; \qquad //\text{write on the database} \\
\qquad H_{glob} := a_h; \\
\qquad V(W) \qquad\qquad //\text{release lock} \\
\quad \}
\end{array}
$$

where a_l is a low level expression and a_h is a high level expression, and $//$ is used for comments. Notice that the fact that the low level user write a_h on a high level location does not mean that he can see the value of a_h, i.e., he only knows the expression but not its value. The variable W is a low level variable initialized to 1. When W is equal to 1 and a low level writer asks to write on the database he turns W to 0, this ensures that the other low level users cannot write, then he writes on the database and he sets again W to 1.

As far high level users are concerned, we can model them as follows

$$
\begin{array}{l}
\texttt{HighR}(H_{loc}, H_{glob}) \equiv \\
\quad \texttt{while}(\text{true})\ \{ \\
\qquad P(MH); \qquad\qquad //\text{get lock to} \ldots \\
\qquad NH := NH + 1; \qquad //\ldots \text{increment nr. of readers and} \ldots \\
\qquad \texttt{if}(NH = 1)\ \{P(W)\}; \qquad //\ldots \text{if first reader, exclude the writers} \\
\qquad V(MH); \qquad\qquad //\text{release lock} \\
\qquad H_{loc} := H_{glob}; \qquad //\text{read from the data base} \\
\qquad P(MH); \qquad\qquad //\text{get lock to} \ldots \\
\qquad \texttt{if}(NH = 1)\ \{V(W)\}; \qquad //\ldots \text{if last reader, release lock on writers and} \\
\qquad NH := NH - 1; \qquad //\ldots \text{decrement the nr. of readers} \ldots \\
\qquad V(MH); \qquad\qquad //\text{release lock} \\
\quad \}
\end{array}
$$

where MH is a low level variable and NH is a high level variable. MH is used to ensure that the writers have mutually exclusive access to NH and W. NH represents the number of active readers. The first high level user who asks to read have to prevent

low level users from writing on the database while he is reading. It is possible to have more than one reader reading the database at the same time. The last reader has to release the lock, so that the writers can write on the database.

It is easy to see that high level readers release information to low level writers. In fact, the instructions $\texttt{if}(NH = 1)\ \{P(W)\}$ and $\texttt{if}(NH = 1)\ \{V(W)\}$ have high level boolean conditions which determine changes on a low level variable. Intuitively, a low level user which is waiting to write can infer that either another low level user is writing or at least one high level user is reading. This is a high level information. since it is the disjunction of a low with a high part. However, the low level writer cannot infer how many readers are active, i.e., he cannot infer the value of NH.

Let us consider the set $D = \{NH = 1\}$. Our proof system can be used to prove that both LowW and HighR are in the class $\mathbf{SIMP}^*_D$, hence any parallel composition of readers and writer is also in $\mathbf{SIMP}^*_D$. □

6 Conclusion and Related Work

In this paper we introduce a generalized unwinding schema for the definition of noninterference properties of programs described in a simple imperative language with a parallel operator and an atomic block constructor. We study different instances of our unwinding condition also accounting for intentional information release. Moreover, we define accurate proof techniques for the verification of compositional noninterference properties for concurrent programs. Indeed, as far as expressions and sequential programs are concerned, our system is complete over the real numbers.

There is a widespread literature on secure information flow in imperative languages (see [28]). Many works concern the definition of noninterference properties controlling the end-to-end behaviour of programs. In the setting of concurrency, these kind of properties have been studied by, e.g., Volpano and Smith in [34] and by Boudol and Castellani in [7]. They both define type systems to ensure the property of noninterference expressed in terms of the following *weak* low bisimulation $\approx^L$: let $\twoheadrightarrow$ stand for one or zero transitions, thus $(\langle P,\sigma\rangle,\langle Q,\theta\rangle) \in \approx^L$ if $\sigma =_l \theta$ and whenever $\langle P,\sigma\rangle \rightarrow \langle P',\sigma'\rangle$, then there exists $\langle Q',\theta'\rangle$ such that $\langle Q,\theta\rangle \twoheadrightarrow \langle Q',\theta'\rangle$ and $(\langle P',\sigma'\rangle,\langle Q',\theta'\rangle) \in \approx^L$, and viceversa. The relation $\approx^L$ is a partial equivalence relation and a program P is secure if $\langle P,\sigma\rangle \approx^L \langle P,\theta\rangle$ for all σ and θ such that $\sigma =_l \theta$. We can instantiate our generalized unwinding condition with $\approx^L$ obtaining, for instance, the classes $W(=_l,\approx^L,\rightsquigarrow)$ and $W(=_l,\approx^L,Reach^*)$ which both imply the property studied in [7, 34]. Unfortunately, such properties are not compositional, neither with respect to the parallel composition operator nor with respect to the sequential one. This is due to the fact that they do not take into account termination and then secure programs like, e.g., $P_1 \equiv \texttt{while}(H = 0)\ \{\texttt{skip}\}$ and $P_2 \equiv L := 1$, give rise to insecure programs when composed through, e.g., $P_1;P_2$ and $\texttt{co}\ P_1 \| P_2\ \texttt{oc}$.

Later work by Volpano and Smith [38] investigates security for multithreaded programs with a probabilistic scheduler which selects a thread from the pool of live threads with a fixed probability distribution. However scheduling policies may vary from implementation to implementation and this motivated the work of Sabelfeld and Sands [30] which argue for scheduler independent security, that is a notion of security robust

with respect to a wide class of potentially probabilistic schedulers. This is achieved by relying on a compositional strong low-bisimulation such that any two strongly low-bisimilar thread pools must be of equal size and must create/kill exactly the same number of processes at each step under any scheduler. In the present paper we do not focus on scheduler independent security. However, our approach is flexible enough to model scheduler independent properties. Indeed it is suffi cient to parameterize the $\xrightarrow{\varepsilon}$ relation with a list of indexes S keeping track of each single thread execution. The new relation $\xrightarrow{\varepsilon}_S$ can be obtained by modifying the rules in Table 1 as follows: the relation $\xrightarrow{\varepsilon}$ is replaced by $\xrightarrow{\varepsilon}_{[\,]}$, where S is the empty list, in all the rules except the fi rst one for the parallel operator $\texttt{co}\ P_1 \| \ldots P_i \ldots \| P_n\ \texttt{oc}$ which becomes:

$$\frac{\langle P_i, \sigma\rangle \xrightarrow{\varepsilon}_S \langle P_i', \sigma'\rangle}{\langle \texttt{co}\ P_1 \| \ldots P_i \ldots \| P_n\ \texttt{oc}, \sigma\rangle \xrightarrow{\varepsilon}_{i,S} \langle \texttt{co}\ P_1 \| \ldots P_i' \ldots \| P_n\ \texttt{oc}, \sigma'\rangle}$$

Bisimulation equivalence relations defi ned on the new LTS's thus require that the same lists of indexes in an arrow are simulated, ensuring that the same threads are executed at each step. In particular, if we restrict ourselves to programs without any occurrence of the `await` operator, we can easily reformulate in terms of an unwinding condition the strong security property for multithreaded programs studied by Sabelfeld and Mantel in [27] and by Agat in [1].

The security properties presented in this paper capture internal timing leaks provided that programs do not contain `await` statements. Timing-sensitive bisimulations have been considered by various authors, e.g., [1, 16, 30, 27, 33], for modelling timing attacks which include the ability to observe the timing behavior of the system. We argue that our approach can also be extended to deal with timing sensitive security properties. It is suffi cient to modify the rules in Table 1 in order to model the timing behavior of atomic blocks in the `await` statements. This is achieved by replacing $\xrightarrow{\varepsilon}$ with $\xrightarrow{\varepsilon}_1$ in all the rules except the one for `await` which becomes

$$\frac{\langle b, \sigma\rangle \rightarrow \text{true} \qquad \langle S, \sigma\rangle \overset{\varepsilon_2}{\hookrightarrow}_n \langle \texttt{end}, \sigma'\rangle}{\langle \texttt{await}(b)\ \{S\}, \sigma\rangle \overset{\varepsilon_1 \cup \varepsilon_2}{\rightarrow}_{n+1} \langle \texttt{end}, \sigma'\rangle} \quad b \in \varepsilon_1$$

where $\overset{\varepsilon}{\hookrightarrow}_n$ is defi ned as $\langle S_0, \sigma_0\rangle \rightarrow_1 \langle S_1, \sigma_1\rangle \rightarrow_1 \cdots \rightarrow_1 \langle S_n, \sigma_n\rangle$, i.e., n denotes the number of transition steps from $\langle S_0, \sigma_0\rangle$ to $\langle S_n, \sigma_n\rangle$.

Security properties for programs admitting downgrading have been recently studied by several authors, see the recent survey [32]. In this paper, we follow the approach of [17, 29]: instead of studying "who can downgrade the data" or "how much information can be downgraded" we study "which information can be released". We assume that the programmers may specify which data can be downgraded. This is expressed by the set D of arithmetic and boolean high level expressions we use to parameterize the defi nition of the secure class $\mathbf{SIMP}^{\star}_{D}$. For instance, if $H\%2$ belongs to D then only the parity of H can be leaked to public. We show that our generalized unwinding condition can be instantiated in order to provide a compositional security property which generalizes pure noninterference and accurately describes the effects due to downgrading.

In particular we prove a soundness theorem with respect to the delimited release property defined by Sabelfeld and Myers in [29]. Our approach differs from most previous works on declassification in a language-based security setting in which constraints, at a linguistic level, are imposed to control the declassification operation. For instance, Volpano and Smith in [36, 39] restrict downgrading to occur by means of specific one-way functions. Another example of constrained downgrading is robust declassification which was proposed, and then studied in a series of papers [21, 22, 23, 41] by Myers *et. al.* The idea of robust declassification is to model a downgrading operation by extending the reading clearances assigned to the owner of an object, and to control it by requiring that this operation runs under appropriate authority guarantee.

Finally, we observe that the properties we have defined in terms of unwinding conditions characterize the security of programs against so-called *passive attackers*, i.e., low level users which try to infer the values of the high level variables just by observing the values of the low level ones. On the contrary, in defining noninterference one usually explicitly characterize the class of *active attackers*, i.e., malicious users or programs which try to directly transmit confidential information to the low level observers. Some authors have proved that there is a connection between properties characterizing passive attacks and properties involving active attacks [41]. In our approach an active attacker can be seen as a high program which intentionally manipulates high level variables. We can prove that if P is a secure program belonging to the class $\mathbf{SIMP}^*$ then a low level user cannot distinguish P running in parallel with different (malicious) high programs P_H and P_K exhibiting the same timing behaviour (i.e., $P_H \sim_l P_K$). More precisely, we can prove that if $P \in \mathbf{SIMP}^*$ then $P\|P_H \sim_l P\|P_K$ for all P_H and P_K containing only high level variables and such that $P_H \sim_l P_K$. Intuitively, this theorem states that if a program P belongs to $\mathbf{SIMP}^*$ then even if the values of the high level variables are changed during the computation, a low level user will never observe any difference on the values of low variables.

More in general, following the approach in [23], we can say that a *fair* attach is any program belonging to the class $\mathbf{SIMP}^*$. In this case we obtain that if $P \in \mathbf{SIMP}^*$ then $P\|R \sim_l P\|Q$ for all $R, Q \in \mathbf{SIMP}^*$ and such that $R \sim_l Q$.

As far as the verification of our properties is concerned, we provide techniques which are more precise than the type-based proof methods presented in, e.g., [1, 7, 27, 29, 30, 34, 38]. Indeed, as explained in the introduction, by exploiting the Tarski decidability result for first-order formulae over the reals, we can infer, for instance, that a program like, $\texttt{while}(L+H>H)\ \{L := 1\}$ is secure. This cannot be captured by previous type systems.

In a recent paper [10], Dam considers a while language extended with parallel composition and studies noninterference properties based on the notion of strong low bisimulation introduced by Sabelfeld and Sands. He proves that strong low bisimulation is decidable on his language, provided that the arithmetic expressions belong to a decidable equational theory. The main differences between his work and the one presented here are the followings. Our language is more expressive since we admit atomic constructions through the `await` statement which allows us to implement mutual exclusion. Moreover, our arithmetic expressions range over an undecidable theory and we exploit an abstraction over the reals to introduce a sound proof system. Our proof system is not complete even if we consider a decidable equational theory. This is

mainly due to the lack of rules for structural congruence. It is not difficult to add such rules to our system to strength our results. Finally, the focus of our work is the introduction of a general unwinding framework for the definition of different flow security properties.

References

[1] J. Agat. Transforming out Timing Leaks. In *Proc. of ACM Symposium on Principles of Programming Languages (POPL'00)*, pages 40–53. ACM Press, 2000.

[2] T. Amtoft and A. Banerjee. Information Flow Analysis in Logical Form. In *Proceedings of the 11th Static Analysis Symposium (SAS'04)*, volume 3148 of *LNCS*, pages 100–115. Springer-Verlag, 2004.

[3] G. R. Andrews. *Foundation of Multithreaded, Parallel, and Distributed Programming*. addison, 2000.

[4] G. Barthe, P. D'Argenio, and T. Rezk. Secure Information Flow by Self Composition. In *Proc. of the 17th IEEE Computer Security Foundations Workshop (CSFW'04)*, pages 100–114. IEEE Computer Society Press, 2004.

[5] A. Bossi, R. Focardi, C. Piazza, and S. Rossi. Verifying Persistent Security Properties. *Computer Languages, Systems and Structures*, 30(3-4):231–258, 2004.

[6] G. Boudol and I. Castellani. Noninterference for Concurrent Programs. In F. Orejas, P. G. Spirakis, and J. van Leeuwen, editors, *Proc. of Int. Colloquium on on Automata, Languages and Programming (ICALP'01)*, volume 2076 of *LNCS*, pages 382–395. Springer-Verlag, 2001.

[7] G. Boudol and I. Castellani. Noninterference for Concurrent Programs and Tread Systems. *Theoretical Computer Science*, 281(1):109–130, 2002.

[8] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In H. Barkhage, editor, *2nd Gl Conf. on Automata Theory and Formal Languages*, volume 33 of *LNCS*, pages 134–183. Springer-Verlag, 1975.

[9] G. E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12:299–328, 1991.

[10] M. Dam. Decidability and Proof Systems for Language-Based Noninterference Relations. In *Proc. of ACM Symposium on Principles of Programming Languages(POPL'06)*. ACM Press, 2006. To appear.

[11] Dorothy E. Denning and Peter J. Denning. Certification of programs for secure information flow. *Commun. ACM*, 20(7):504–513, 1977.

[12] R. Focardi and R. Gorrieri. Classification of Security Properties (Part I: Information Flow). In R. Focardi and R. Gorrieri, editors, *Proc. of Foundations of Security Analysis and Design (FOSAD'01)*, volume 2171 of *LNCS*, pages 331–396. Springer-Verlag, 2001.

[13] J. A. Goguen and J. Meseguer. Security Policies and Security Models. In *Proc. of the IEEE Symposium on Security and Privacy (SSP'82)*, pages 11–20. IEEE Computer Society Press, 1982.

[14] J. A. Goguen and J. Meseguer. Unwinding and Inference Control. In *Proc. of the IEEE Symposium on Security and Privacy (SSP'84)*, pages 75–86. IEEE Computer Society Press, 1984.

[15] D. Grigoriev. Complexity of Deciding Tarski Algebra. *Journal of Symbolic Computation*, 5:65–108, 1988.

[16] K. G. Larsen and A. Skou. Bisimulation through Probabilistic Testing. In *Proc. of ACM Symposium on Principles of Programming Languages (POPL'89)*, pages 344–352. ACM Press, 1989.

[17] P. Li and S. Zdancewic. Downgrading Policies and Relaxed Noninterference. In *Proc. of ACM Symposium on Principles of Programming Languages (POPL'05)*, pages 158–170. ACM Press, 2005.

[18] H. Mantel. Unwinding Possibilistic Security Properties. In *Proc. of the European Symposium on Research in Computer Security (ESoRiCS'00)*, volume 2895 of *LNCS*, pages 238–254. Springer-Verlag, 2000.

[19] Y. V. Matiyasevich. Enumerable sets are diophantine. *Soviet Mathematics Doklady*, 11(2):354–358, 1970.

[20] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[21] A. C. Myers. JFlow: Practical Mostly-Static Information Flow Control. In *Proc. of ACM Symposium on Principles of Programming Languages (POPL'99)*, pages 228–241. ACM Press, 1999.

[22] A. C. Myers and B. Liskov. Protecting Privacy Using the Decentralized Label Model. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 9(4):410–442, 2000.

[23] A. C. Myers, A. Sabelfeld, and S. Zdancewic. Enforcing Robust Declassifi cation. In *Proc. of the 17th IEEE Computer Security Foundations Workshop (CSFW'04)*, pages 172–186. IEEE Computer Society, 2004.

[24] David von Oheimb. Information flow control revisited: Noninfluence = Noninterference + Nonleakage. In *Proc. of the 9th European Symposium on Research in Computer Security (ESORICS'04)*, volume 3193 of *LNCS*, pages 225–243. Springer, 2004.

[25] J. Renegar. On the Computational Complexity and Geometry of the First-order Theory of the Reals, parts I-III. *Journal of Symbolic Computation*, 13:255–352, 1992.

[26] P.Y.A. Ryan and S. Schneider. Process Algebra and Non-Interference. *Journal of Computer Security*, 9(1/2):75–103, 2001.

[27] A. Sabelfeld and H. Mantel. Static Confidentiality Enforcement for Distributed Programs. In M. V. Hermenegildo and G. Puebla, editors, *Proc. of Int. Static Analysis Symposium (SAS'02)*, volume 2477 of *LNCS*, pages 376–394. Springer-Verlag, 2002.

[28] A. Sabelfeld and A. C. Myers. Language-Based Information-Flow Security. *IEEE Journal on Selected Areas in Communication*, 21(1):5–19, 2003.

[29] A. Sabelfeld and A. C. Myers. A Model for Delimited Information Release. In *Proceedings of the International Symposium on Software Security, Theories and Systems (ISSS 2003)*, volume 3233 of *LNCS*, pages 174–191. Springer-Verlag, 2003.

[30] A. Sabelfeld and D. Sands. Probabilistic Noninterference for Multi-threaded Programs. In *Proc. of the IEEE Computer Security Foundations Workshop (CSFW'00)*, pages 200–215. IEEE Computer Society Press, 2000.

[31] A. Sabelfeld and D. Sands. A Per Model of Secure Information Flow in Sequential Programs. *Higher-Order and Symbolic Computation*, 14(1):59–91, 2001.

[32] A. Sabelfeld and D. Sands. Dimensions and Principles of Declassification. In *Proc. of the IEEE Computer Security Foundations Workshop (CSFW'05)*, pages 255–269. IEEE Computer Society Press, 2005.

[33] G. Smith. A New Type System for Secure Information Flow. In *Proc. of the IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 115–125. IEEE Computer Society, 2001.

[34] G. Smith and D. M. Volpano. Secure Information Flow in a Multi-threaded Imperative Language. In *Proc. of ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'98)*, pages 355–364. ACM Press, 1998.

[35] A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. CA: University of California Press, 2nd edition, 1951.

[36] D. Volpano. Secure Introduction of One-Way Functions. In *Proc. of the IEEE Computer Security Foundations Workshop (CSFW'00)*, pages 246–254. IEEE Computer Society Press, 2000.

[37] D. M. Volpano and G. Smith. A Type-Based Approach to Program Security. In *TAPSOFT*, pages 607–621, 1997.

[38] D. M. Volpano and G. Smith. Probabilistic Noninterference in a Concurrent Language. *Journal of Computer Security*, 7(2-3):231 – 253, 1999.

[39] D. M. Volpano and G. Smith. Verifying Secrets and Relative Secrecy. In *Proc. of ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'00)*, pages 268–276. ACM Press, 2000.

[40] G. Winskel. *The formal semantics of programming languages*. The MIT Press, 1993.

[41] S. Zdancewic and A. C. Myers. Robust Declassifi cation. In *Proc. of the IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 15–23. IEEE Computer Society Press, 2001.

$$\frac{}{\langle \texttt{skip}, \sigma\rangle \xrightarrow{\texttt{low}} \langle \texttt{end}, \sigma\rangle} \qquad \frac{\langle a, \sigma\rangle \to n}{\langle X := a, \sigma\rangle \xrightarrow{\varepsilon} \langle \texttt{end}, \sigma[X/n]\rangle}\ a \in \varepsilon$$

$$\frac{\langle P_0, \sigma\rangle \xrightarrow{\varepsilon} \langle P_0', \sigma'\rangle}{\langle P_0; P_1, \sigma\rangle \xrightarrow{\varepsilon} \langle P_0'; P_1, \sigma'\rangle}\ P_0' \not\equiv \texttt{end} \qquad \frac{\langle P_0, \sigma\rangle \xrightarrow{\varepsilon} \langle \texttt{end}, \sigma'\rangle}{\langle P_0; P_1, \sigma\rangle \xrightarrow{\varepsilon} \langle P_1, \sigma'\rangle}$$

$$\frac{\langle b, \sigma\rangle \to \text{true}}{\langle \texttt{if}(b)\ \{P_0\}\ \texttt{else}\ \{P_1\}, \sigma\rangle \xrightarrow{\varepsilon} \langle P_0, \sigma\rangle}\ b \in \varepsilon$$

$$\frac{\langle b, \sigma\rangle \to \text{false}}{\langle \texttt{if}(b)\ \{P_0\}\ \texttt{else}\ \{P_1\}, \sigma\rangle \xrightarrow{\varepsilon} \langle P_1, \sigma\rangle}\ b \in \varepsilon$$

$$\frac{\langle b, \sigma\rangle \to \text{true}}{\langle \texttt{while}(b)\ \{P\}, \sigma\rangle \xrightarrow{\varepsilon} \langle P; \texttt{while}(b)\ \{P\}, \sigma\rangle}\ b \in \varepsilon$$

$$\frac{\langle b, \sigma\rangle \to \text{false}}{\langle \texttt{while}(b)\ \{P\}, \sigma\rangle \xrightarrow{\varepsilon} \langle \texttt{end}, \sigma\rangle}\ b \in \varepsilon$$

$$\frac{\langle b, \sigma\rangle \to \text{true} \qquad \langle S, \sigma\rangle \overset{\varepsilon_2}{\rightsquigarrow} \langle \texttt{end}, \sigma'\rangle}{\langle \texttt{await}(b)\ \{S\}, \sigma\rangle \xrightarrow{\varepsilon_1 \cup \varepsilon_2} \langle \texttt{end}, \sigma'\rangle}\ b \in \varepsilon_1$$

$$\frac{\langle b, \sigma\rangle \to \text{false}}{\langle \texttt{await}(b)\ \{S\}, \sigma\rangle \xrightarrow{\varepsilon} \langle \texttt{await}(b)\ \{S\}, \sigma\rangle}\ b \in \varepsilon$$

$$\frac{\langle P_i, \sigma\rangle \xrightarrow{\varepsilon} \langle P_i', \sigma'\rangle}{\langle \texttt{co}\ P_1 \| \ldots \| P_i \| \ldots \| P_n\ \texttt{oc}, \sigma\rangle \xrightarrow{\varepsilon} \langle \texttt{co}\ P_1 \| \ldots \| P_i' \| \ldots \| P_n\ \texttt{oc}, \sigma'\rangle}$$

$$\frac{}{\langle \texttt{co end} \| \ldots \| \texttt{end} \| \ldots \| \texttt{end oc}, \sigma\rangle \xrightarrow{\texttt{low}} \langle \texttt{end}, \sigma\rangle}$$

Table 1: The operational semantics.

$$\frac{}{\texttt{skip} \eqslantless_l \texttt{skip}} \qquad \frac{}{\texttt{end} \eqslantless_l \texttt{end}}$$

$$\frac{P_0 \eqslantless_l Q_0 \text{ and } P_1 \eqslantless_l Q_1}{P_0;P_1 \eqslantless_l Q_0;Q_1} \qquad \frac{P_i \eqslantless_l Q_i \quad 1 \leq i \leq n}{\texttt{co}\ P_1 \| \ldots \| P_n\ \texttt{co} \eqslantless_l \texttt{co}\ Q_1 \| \ldots \| Q_n\ \texttt{co}}$$

$$\frac{}{L := a_0 \eqslantless_l L := a_1}\ L \in \mathbb{L} \text{ and } a_0 \eqslantless_l a_1 \qquad \frac{}{H := a_0 \eqslantless_l K := a_1}\ H, K \in \mathbb{H}$$

$$\frac{P_0 \eqslantless_l Q_0}{\texttt{if}(b_0)\ \{P_0\}\ \texttt{else}\ \{P_1\} \eqslantless_l \texttt{if}(b_1)\ \{Q_0\}\ \texttt{else}\ \{Q_1\}}\ b_0 \eqslantless_l b_1 \eqslantless_l \text{true}$$

$$\frac{P_1 \eqslantless_l Q_1}{\texttt{if}(b_0)\ \{P_0\}\ \texttt{else}\ \{P_1\} \eqslantless_l \texttt{if}(b_1)\ \{Q_0\}\ \texttt{else}\ \{Q_1\}}\ b_0 \eqslantless_l b_1 \eqslantless_l \text{false}$$

$$\frac{P_0 \eqslantless_l Q_0 \text{ and } P_1 \eqslantless_l Q_1}{\texttt{if}(b_0)\ \{P_0\}\ \texttt{else}\ \{P_1\} \eqslantless_l \texttt{if}(b_1)\ \{Q_0\}\ \texttt{else}\ \{Q_1\}}\ b_0 \eqslantless_l b_1$$

$$\frac{P_0 \eqslantless_l Q_1}{\texttt{if}(b_0)\ \{P_0\}\ \texttt{else}\ \{P_1\} \eqslantless_l \texttt{if}(b_1)\ \{Q_0\}\ \texttt{else}\ \{Q_1\}}\ b_0 \eqslantless_l \neg b_1 \eqslantless_l \text{true}$$

$$\frac{P_1 \eqslantless_l Q_0}{\texttt{if}(b_0)\ \{P_0\}\ \texttt{else}\ \{P_1\} \eqslantless_l \texttt{if}(b_1)\ \{Q_0\}\ \texttt{else}\ \{Q_1\}}\ b_0 \eqslantless_l \neg b_1 \eqslantless_l \text{false}$$

$$\frac{P_0 \eqslantless_l Q_1 \text{ and } P_1 \eqslantless_l Q_0}{\texttt{if}(b_0)\ \{P_0\}\ \texttt{else}\ \{P_1\} \eqslantless_l \texttt{if}(b_1)\ \{Q_0\}\ \texttt{else}\ \{Q_1\}}\ b_0 \eqslantless_l \neg b_1$$

$$\frac{P_0 \eqslantless_l Q_0 \text{ and } Q_0 \eqslantless_l Q_1 \text{ and } P_1 \eqslantless_l Q_1}{\texttt{if}(b_0)\ \{P_0\}\ \texttt{else}\ \{P_1\} \eqslantless_l \texttt{if}(b_1)\ \{Q_0\}\ \texttt{else}\ \{Q_1\}}$$

$$\frac{P_0 \eqslantless_l P_0}{\texttt{if}(b_0)\ \{P_0\}\ \texttt{else}\ \{P_1\} \eqslantless_l \texttt{skip};P_0}\ b_0 \eqslantless_l \text{true}$$

$$\frac{P_1 \eqslantless_l P_1}{\texttt{if}(b_0)\ \{P_0\}\ \texttt{else}\ \{P_1\} \eqslantless_l \texttt{skip};P_1}\ b_0 \eqslantless_l \text{false}$$

$$\frac{P_0 \eqslantless_l P_1}{\texttt{if}(b_0)\ \{P_0\}\ \texttt{else}\ \{P_1\} \eqslantless_l \texttt{skip};P_0}$$

Table 2: The rules of relation $\eqslantless_l$ for skip, end, ; ,co $\|$ oc, :=, and if.

$$\frac{P_0 \eqslantless_l Q_0}{\texttt{while}(b_0)\ \{P_0\} \eqslantless_l \texttt{while}(b_1)\ \{Q_0\}}\quad b_0 \eqslantless_l b_1$$

$$\frac{}{\texttt{while}(b_0)\ \{P_0\} \eqslantless_l \texttt{while}(b_1)\ \{Q_0\}}\quad b_0 \eqslantless_l b_1 \eqslantless_l \text{false}$$

$$\frac{}{\texttt{while}(b_0)\ \{P_0\} \eqslantless_l \texttt{skip}}\quad b_0 \eqslantless_l \text{false}$$

$$\frac{}{\texttt{await}(b_0)\ \{S_0\} \eqslantless_l \texttt{await}(b_1)\ \{S_1\}}\quad b_0 \eqslantless_l b_1 \text{ and } S_0 \eqslantless_l S_1$$

$$\frac{Q \eqslantless_l P}{P \eqslantless_l Q}\quad l(P) < l(Q)$$

Table 3: The rules of relation $\eqslantless_l$ for `while`, `await`, and symmetry.

$$\frac{}{\texttt{end} \in \mathcal{W}(\precsim_l)} \qquad \frac{}{\texttt{skip} \in \mathcal{W}(\precsim_l)}$$

$$\frac{}{L := a \in \mathcal{W}(\precsim_l)}\ L \in \mathbb{L} \text{ and } a \eqsim_l a \qquad \frac{}{H := a \in \mathcal{W}(\precsim_l)}\ H \in \mathbb{H}$$

$$\frac{P_0 \in \mathcal{W}(\precsim_l) \text{ and } P_1 \in \mathcal{W}(\precsim_l)}{P_0; P_1 \in \mathcal{W}(\precsim_l)} \qquad \frac{P_i \in \mathcal{W}(\precsim_l) \quad 1 \leq i \leq n}{\texttt{co}\ P_1 \| \ldots \| P_n\ \texttt{oc} \in \mathcal{W}(\precsim_l)}$$

$$\frac{P_0 \in \mathcal{W}(\precsim_l)}{\texttt{if}(b)\ \{P_0\}\ \texttt{else}\ \{P_1\} \in \mathcal{W}(\precsim_l)}\ b \eqsim_l \text{true}$$

$$\frac{P_1 \in \mathcal{W}(\precsim_l)}{\texttt{if}(b)\ \{P_0\}\ \texttt{else}\ \{P_1\} \in \mathcal{W}(\precsim_l)}\ b \eqsim_l \text{false}$$

$$\frac{P_0 \in \mathcal{W}(\precsim_l) \text{ and } P_1 \in \mathcal{W}(\precsim_l)}{\texttt{if}(b)\ \{P_0\}\ \texttt{else}\ \{P_1\} \in \mathcal{W}(\precsim_l)}\ b \eqsim_l b \qquad \frac{P_0 \precsim_l P_1}{\texttt{if}(b)\ \{P_0\}\ \texttt{else}\ \{P_1\} \in \mathcal{W}(\precsim_l)}$$

$$\frac{}{\texttt{while}(b)\ \{P\} \in \mathcal{W}(\precsim_l)}\ b \eqsim_l \text{false} \qquad \frac{P \in \mathcal{W}(\precsim_l)}{\texttt{while}(b)\ \{P\} \in \mathcal{W}(\precsim_l)}\ b \eqsim_l b$$

$$\frac{}{\texttt{await}(b)\ \{S\} \in \mathcal{W}(\precsim_l)}\ b \eqsim_l b \text{ and } S \eqsim_l S$$

Table 4: The class $\mathcal{W}(\precsim_l)$.