Compositional Information Flow Security for Concurrent Programs
Compositional Information Flow Security for Concurrent Programs

Annalisa Bossi\textsuperscript{1}, Carla Piazza\textsuperscript{1,2}, and Sabina Rossi\textsuperscript{1}

\textsuperscript{1} Dipartimento di Informatica, Università Ca’ Foscari di Venezia
via Torino 155, 30172 Venezia, Italy
\textsuperscript{2} Dipartimento di Matematica ed Informatica, Università degli Studi di Udine
via Le Scienze 206, 33100 Udine, Italy
\{bossi,piazza,srossi\}@dsi.unive.it

Abstract. We present a general unwinding framework for the definition of information flow security properties of concurrent programs, described in a standard imperative language enriched with parallelism. 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.

1 Introduction

The protection of confidential data in computing systems has long been recognized as a difficult and daunting problem. In order to guarantee the confidentiality of sensitive data it is necessary to analyze how information flows so that secrets are not transmitted 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 confidential) information from affecting lower level (less confidential) information. Recently, there has been much work applying this approach in a language-based setting. We refer the reader to \cite{26} for a clear and wide overview about the different approaches. All of these proposals accomplish the non-interference principle \cite{11} which requires that secret input data cannot be inferred through the observation of non confidential outputs. Among the approaches based on formal methods, non-interference has been formalized in terms of behavioural equivalences, e.g., \cite{10,24}, type-systems, e.g., \cite{25,26,33,34}, and logical formulations, e.g., \cite{2,3}.

In this paper we consider the problem of specifying and verifying the noninterference property for concurrent programs, described in a simple imperative language admitting parallel executions on a shared memory. The locations

\textsuperscript{*} This work has been partially supported by the EU project IST-2001-32617 (MyThS), the MIUR FIRB grant RBAU018RCZ and the MIUR PRIN’04 grant 2004013015.
(variables) of the language are partitioned into two levels, a public level and a confidential one.

We start from the observation that, in order to be of practical usefulness, 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 [10] can be naturally characterized in terms of unwinding conditions [12]. In this paper we investigate how to instantiate the unwinding based framework for the definition of non interference 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 [12, 16, 22]. 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 (confidential) 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 define 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 timing-sensitive security properties (in the spirit of [25, 28]) which entail the non-interference principle.

The problem of verification is tracked in Section 4. We focus on one instance of our unwinding condition which is compositional with respect to the language constructors. We define accurate proof methods for the verification of such property which are more precise than previous type-based techniques such as those presented in [1, 7, 25, 31]. 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 do not contain high level variables. Thus, for instance, if $H$ is a high level variable while $L$ is a low level one, the commands

\begin{align}
L &:= H \\
L &:= H - H \\
\text{while } (L + H > H) \text{ do } L := 1 \\
\text{if } (H = 0) \text{ then } L := 1 \text{ else } L := 1
\end{align}

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 properties studied in Sections 3 and 4 entail the standard non-interference principle. However, as already noticed by many authors, e.g., [15, 27, 36], non-interference 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 attempt 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 non-interference 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 timing-sensitive compositional version of the delimited release property defined in [27]. 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 non-interference property. In Section 4 we show that the non-interference 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 verification techniques to deal with such properties. Finally, in Section 6 we discuss related work and draw some conclusions. All the proof of the results presented in this paper are reported in the Appendix\textsuperscript{3}.

2 The Language: Syntax and Semantics

The language we consider is an extension of the IMP language defined in [35] where parallel executions are admitted and 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., at any point of the execution the values in the low level locations have not to depend on high level inputs.

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

\textsuperscript{3} A preliminary version of this paper covering part of Section 3 has been presented at LOPSTR’04.
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 $Z$ be the set of integer numbers, $T = \{\text{true}, \text{false}\}$ be the set of boolean values, $L$ be a set of low level locations and $H$ be a set of high level locations, with $L \cap H = \emptyset$. The set $A_{\text{exp}}$ of arithmetic expressions is defined by the grammar:

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

where $n \in Z$ and $X \in L \cup H$. The set $B_{\text{exp}}$ of boolean expressions is defined by:

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

where $a_0, a_1 \in A_{\text{exp}}$.

We say that an arithmetic expression $a$ is confidential, denoted by $a \in \text{high}$, if there is a high level location which occurs in it. Otherwise we say that $a$ is public, denoted by $a \in \text{low}$. Similarly, we say that a boolean expression $b$ is confidential, denoted by $b \in \text{high}$, if there is a confidential arithmetic expression which occurs in it. Otherwise we say that $b$ is public, denoted by $b \in \text{low}$. This notion of confidentiality, both for arithmetic and boolean expressions, is purely syntactic. 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 $\text{Prog}$ of programs of our language is defined as:

$$P ::= \text{skip} \mid X := a \mid P_0; P_1 \mid \text{while } b \text{ do } P \mid \text{if } b \text{ then } P_0 \text{ else } P_1 \mid P_0 | P_1$$

where $a \in A_{\text{exp}}$, $X \in L \cup H$, and $b \in B_{\text{exp}}$.

We say that an assignment $X := a$ is confidential, denoted by $X := a \in \text{high}$, if either $X$ is a high level location or $a \in \text{high}$. Otherwise we say that $X := a$ is public, denoted by $X := a \in \text{low}$.

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

Given an arithmetic expression $a \in A_{\text{exp}}$ and a state $\sigma$, the evaluation of $a$ in $\sigma$, denoted by $\langle a, \sigma \rangle \rightarrow n$ with $n \in Z$, is defined as in [35]. Similarly, $\langle b, \sigma \rangle \rightarrow v$ with $b \in B_{\text{exp}}$ and $v \in \{\text{true}, \text{false}\}$, denotes the evaluation of a boolean expression $b$ in a state $\sigma$ and is defined as in [35]. Notice that in both cases atomicity of expression evaluation is assumed.

Our operational semantics is expressed in terms of state transitions. A transition from a program $P$ and a state $\sigma$ has the form $\langle P, \sigma \rangle \xrightarrow{\epsilon} \langle P', \sigma' \rangle$ where $P'$ is either a program or the special symbol end (denoting termination) and $\epsilon \in \{\text{high, low}\}$ stating that the transition is either confidential or public. Let $\mathbb{P} = \text{Prog} \cup \{\text{end}\}$ and $\Sigma$ be the set of all the possible states. In Figure 1 we define the operational semantics of $\langle P, \sigma \rangle \in \mathbb{P} \times \Sigma$ by structural induction on $P$. 

4
We write \( \langle P, \sigma \rangle \rightarrow \langle P', \sigma' \rangle \) to denote \( \langle P, \sigma \rangle \overset{\epsilon}{\rightarrow} \langle P', \sigma' \rangle \) with \( \epsilon \in \{\text{low, high}\} \). We write \( \langle P_0, \sigma_0 \rangle \rightarrow^n \langle P_n, \sigma_n \rangle \) with \( n \geq 0 \) for \( \langle P_0, \sigma_0 \rangle \rightarrow \langle P_1, \sigma_1 \rangle \rightarrow \cdots \rightarrow \langle P_{n-1}, \sigma_{n-1} \rangle \rightarrow \langle P_n, \sigma_n \rangle \). Given \( \langle P, \sigma \rangle \in \text{Prog} \times \Sigma \), we denote by \( \text{Reach}(\langle P, \sigma \rangle) \) the set of pairs \( \langle P', \sigma' \rangle \) such that there exists \( n \geq 0 \) and \( \langle P, \sigma \rangle \rightarrow^n \langle P', \sigma' \rangle \). Moreover, we write \( P \overset{\epsilon}{\rightarrow} P' \) with \( \epsilon \in \{\text{low, high}\} \) if \( \langle P, \sigma \rangle \overset{\epsilon}{\rightarrow} \langle P', \sigma' \rangle \) for some \( \sigma \) and \( \sigma' \); we write \( P \rightarrow^n P' \) if \( \langle P, \sigma \rangle \rightarrow^n \langle P', \sigma' \rangle \) for some \( \sigma \) and \( \sigma' \); we denote by \( \text{Reach}(P) \) the set of programs \( P' \) such that \( \langle P', \sigma' \rangle \in \text{Reach}(\langle P, \sigma \rangle) \) for some states \( \sigma \) and \( \sigma' \). We indicate by \( \text{Reach}^*: \text{Prog} \rightarrow \wp(\text{Prog}) \) the transitive closure of the reachability function \( \text{Reach} : \text{Prog} \rightarrow \wp(\text{Prog}) \) and we say that \( \langle F, \psi \rangle \in \text{Reach}^*(\langle P, \sigma \rangle) \) if \( \psi, \sigma \in \Sigma \) and \( F \in \text{Reach}^*(P) \). In other words, a pair \( \langle F, \psi \rangle \) is reachable from \( \langle P, \sigma \rangle \) through the function \( \text{Reach}^* \) if it can be obtained by reducing \( \langle P, \sigma \rangle \) via the operational semantics, allowing arbitrary changes in the memory throughout the derivation. This definition of reachability coincides with the notion of derivative adopted in [4]. Notice that both functions \( \text{Reach} \) and \( \text{Reach}^* \) over \( \text{Prog} \times \Sigma \) are transitive, i.e., for \( R \in \{\text{Reach, Reach}^*\} \), if \( \langle F'', \psi'' \rangle \in R(\langle F', \psi' \rangle) \) and \( \langle F', \psi' \rangle \in R(\langle F, \psi \rangle) \), then \( \langle F'', \psi'' \rangle \in R(\langle F, \psi \rangle) \). These notions of reachability do not depend on the labels of the edges.

Example 1. Consider the following program

\[
P \equiv \text{if } (L = 1) \text{ then } P' \text{ else skip}
\]

where \( P' \) is the program \( P' \equiv \text{if } (L \neq 1) \text{ then } L := 2 \text{ else skip} \). In this case \( \text{Reach}(P) = \{ \langle P, P', \text{skip} \rangle \} \) while \( \text{Reach}^*(P) = \text{Reach}(P) \cup \{ L := 2 \} \). In fact, if \( \sigma \) is such that \( \sigma(L) = 1 \) we get that \( \langle P, \sigma \rangle \rightarrow \langle P', \sigma \rangle \). If now we take \( \sigma' \) such that \( \sigma'(L) \neq 1 \) we get that \( \langle P', \sigma' \rangle \rightarrow L := 2 \), i.e., \( L := 2 \in \text{Reach}^*(P) \). Hence, \( \langle L := 2, \psi \rangle \in \text{Reach}^*(\langle P, \sigma \rangle) \) 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. Consider the following program \( P \equiv \text{if } (H \leq 3) \text{ then } L := L + 1 \text{ else } L := L + 2 \). Let \( \sigma_1, \sigma_2 \) 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

\[
\begin{align*}
\langle P, \sigma_1 \rangle_{\text{high}} & \rightarrow \langle L := L + 1, \sigma_1 \rangle_{\text{low}} \\
\langle P, \sigma_2 \rangle_{\text{high}} & \rightarrow \langle L := L + 2, \sigma_2 \rangle_{\text{low}} \\
\langle \text{end}, \sigma_1[L/\sigma_1(L) + 1] \rangle & \rightarrow \langle \text{end}, \sigma_2[L/\sigma_2(L) + 2] \rangle
\end{align*}
\]

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 \( L \).

We are interested in a notion of behavioural equivalence which equates two programs if they are indistinguishable for a low level observer.
Fig. 1. The operational semantics.
Example 3. Consider the programs $H := 1; L := 1$ and $H := 2; L := H - 1$.
Given a state $\sigma$ the LTS’s associated to the two programs are respectively

$$
\begin{align*}
\langle H := 1; L := 1, \sigma \rangle \quad & \quad \langle H := 2; L := H - 1, \sigma \rangle \\
\downarrow \text{high} & \quad \downarrow \text{high} \\
\langle L := 1, \sigma[H/1] \rangle & \quad \langle L := H - 1, \sigma[H/2] \rangle \\
\downarrow \text{low} & \quad \downarrow \text{low} \\
\langle \text{end}, \sigma[H/1, L/1] \rangle & \quad \langle \text{end}, \sigma[H/2, L/1] \rangle
\end{align*}
$$

The two program executions 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 1. (Low Level Bisimulation)** A binary symmetric relation $B$ over $\mathbb{P} \times \Sigma$ is a low level bisimulation if for each $((P, \sigma), (Q, \theta)) \in B$ it holds that:

- $\sigma \equiv_1 \theta$, i.e., the states coincide on low level locations;
- if $(P, \sigma) \rightarrow (P', \sigma')$, then there exists $(Q', \theta')$ such that $(Q, \theta) \rightarrow (Q', \theta')$ and $(\langle P', \sigma' \rangle, \langle Q', \theta' \rangle) \in B$.

Two pairs $(P, \sigma)$ and $(Q, \theta) \in \mathbb{P} \times \Sigma$ are low level bisimilar, denoted by $(P, \sigma) \approx_1 (Q, \theta)$ if there exists a low level bisimulation $B$ such that $((P, \sigma), (Q, \theta)) \in B$.

Two programs $P$ and $Q$ are said to be low level bisimilar, denoted by $P \approx_1 Q$, if for each $\sigma, \theta \in \Sigma$ it holds that if $\sigma \equiv_1 \theta$, then $(P, \sigma) \approx_1 (Q, \theta)$.

A partial equivalence relation (per for short) [29] is a symmetric and transitive relation.

**Lemma 1.** The relation $\approx_1 \subseteq (\mathbb{P} \times \Sigma)^2$ is the largest low level bisimulation and it is an equivalence relation. The relation $\approx_1 \subseteq \mathbb{P}^2$ is a partial equivalence relation.

In [28] a stronger low level bisimulation is introduced to reason on concurrent and multi-threaded programs.

**Definition 2. (Strong Low Level Bisimulation)** A binary symmetric relation $B$ over $\mathbb{P}$ is a strong low level bisimulation if for each $(P, Q) \in B$ it holds that

- for all $\sigma, \theta \in \Sigma$ such that $\sigma \equiv_1 \theta$, if $(P, \sigma) \rightarrow (P', \sigma')$, then there exists $Q'$ and $\theta'$ such that $(Q, \theta) \rightarrow (Q', \theta')$, $\sigma' \equiv_1 \theta'$ and $(P', Q') \in B$.

Two programs $P, Q \in \mathbb{P}$ are strongly low level bisimilar, denoted by $P \sim_1 Q$ if there exists a low level bisimulation $B$ such that $(P, Q) \in B$.

**Lemma 2.** [28] The relation $\sim_1 \subseteq \mathbb{P}^2$ is a partial equivalence relation.

Both the relations $\approx_1$ and $\sim_1$ 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 $\sigma$ and $\theta$ such that $\sigma(H) = 1, \theta(H) = 2, \sigma(L) = \theta(L)$. In this case $\sigma \equiv_1 \theta$. However, $\langle L := H, \sigma \rangle \rightarrow \langle \text{end}, \sigma[L/1] \rangle$ and $\langle L := H, \theta \rangle \rightarrow \langle \text{end}, \theta[L/2] \rangle$ where $\sigma[L/1] \neq \theta[L/2]$. Therefore neither $L := H \approx_1 L := H$ nor $L := H \sim_1 L := H$. 

7
Example 4. Consider the programs of Example 3: $P \equiv H := 1; L := 1$ and $Q \equiv H := 2; L := H - 1$. 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., $P \not\simeq l Q$. This reflects the fact that if one considers, for instance, the program $R \equiv H := 5$ then the programs $P \parallel R$ and $Q \parallel R$ do not exhibit the same behaviour from the low level point of view. In fact, starting from a state $\sigma$, any execution of $P \parallel R$ always terminates in a state $\sigma_1$ such that $\sigma_1(L) = 1$. On the other hand, there exists one execution of $Q \parallel R$ which terminate in a state $\sigma_2$ such that $\sigma_2(L) = 4$, i.e., $\sigma_1 \neq l \sigma_2$.

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 timing behavior. Specifically, each step performed by one program is simulated by exactly one step performed by the other program.

Lemma 3. Let $P$ and $Q$ be two programs and $\sigma, \theta \in \Sigma$.

1. Let $\langle P, \sigma \rangle \simeq_l \langle Q, \theta \rangle$. If $\langle P, \sigma \rangle \rightarrow^n \langle P', \sigma' \rangle$, then there exists $Q'$ and $\theta'$ such that $\langle Q, \theta \rangle \rightarrow^n \langle Q', \theta' \rangle$ and $\langle P', \sigma' \rangle \simeq_l \langle Q', \theta' \rangle$, and vice versa.

2. Let $P \sim_l Q$ and $\sigma \equiv_l \theta$. If $\langle P, \sigma \rangle \rightarrow^n \langle P', \sigma' \rangle$, then there exists $Q'$ and $\theta'$ such that $\langle Q, \theta \rangle \rightarrow^n \langle Q', \theta' \rangle$, $\sigma' \equiv_l \theta'$ and $P' \sim_l Q'$, and vice versa.

3 Unwinding Conditions for Security of IMP

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 [19]. 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 non-interference 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 $\equiv$ which equates two states if they are indistinguishable for a low level observer;
- a binary relation $\equiv$ which equates two pairs $\langle P, \sigma \rangle$ and $\langle Q, \theta \rangle$ if they are indistinguishable for a low level observer;
- a reachability function $R$ associating to each pair $\langle P, \sigma \rangle$ the set of 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 $\langle F, \psi \rangle \xrightarrow{\text{high}} \langle G, \varphi \rangle$ 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 $\{\langle F, \pi \rangle \mid \pi \equiv \psi\}$ (whose states are indistinguishable for a low level user) may perform a transition reaching an element of the set $\{\langle R, \rho \rangle \mid \langle R, \rho \rangle \equiv \langle G, \phi \rangle\}$ (whose elements are all indistinguishable with respect to the given binary relation $\equiv$).
Definition 3. (Generalized Unwinding) Let $\equiv$ be a binary relation over $\mathcal{P} \times \Sigma$ and $\mathcal{R}$ be a function from $\mathcal{Prog} \times \Sigma$ to $\varphi(\mathcal{Prog} \times \Sigma)$. We define the unwinding class $\mathcal{W}(\equiv, \equiv, \mathcal{R})$ by:

$$
\mathcal{W}(\equiv, \equiv, \mathcal{R}) \overset{\text{def}}{=} \{ \langle P, \sigma \rangle \in \mathcal{Prog} \times \Sigma \mid \forall \langle F, \psi \rangle \in \mathcal{R}(\langle P, \sigma \rangle), \text{ if } \langle F, \psi \rangle \overset{\text{high}}{\rightarrow} \langle G, \varphi \rangle \text{ then } \forall \pi \in \Sigma \text{ such that } \pi \equiv \psi \exists \langle R, \rho \rangle : \langle F, \pi \rangle \rightarrow \langle R, \rho \rangle \text{ and } \langle G, \varphi \rangle \equiv \langle R, \rho \rangle \}.
$$

In the above definition we do not assume any condition on the reachability function $\mathcal{R}$. 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 4. Let $\mathcal{R}$ be a transitive reachability function from $\mathcal{Prog} \times \Sigma$ to $\varphi(\mathcal{Prog} \times \Sigma)$ and $\langle P, \sigma \rangle \in \mathcal{Prog} \times \Sigma$. If $\langle P, \sigma \rangle \in \mathcal{W}(\equiv, \equiv, \mathcal{R})$, then $\langle F, \psi \rangle \in \mathcal{W}(\equiv, \equiv, \mathcal{R})$ for all $\langle F, \psi \rangle \in \mathcal{R}(\langle P, \sigma \rangle)$.

Hereafter we discuss some instantiations of our generalized unwinding condition.

The class of secure imperative programs $\mathcal{SIMP}_{\approx_l}$ is obtained by instantiating Definition 3 with the low level bisimilarities $\approx_l$ for $\equiv$ and $\approx_l$ for $\equiv$, and the function $\text{Reach}$ for $\mathcal{R}$.

Definition 4. ($\mathcal{SIMP}_{\approx_l}$) A program $P$ belongs to the class $\mathcal{SIMP}_{\approx_l}$ if for each state $\sigma$, $\langle P, \sigma \rangle \in \mathcal{W}(\approx_l, \approx_l, \text{Reach})$.

Example 5. Consider the program $P \equiv L := H; L := 1$. We can prove that it does not hold $\langle P, \sigma \rangle \in \mathcal{W}(\approx_l, \approx_l, \text{Reach})$ 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 $\sigma = \theta, \text{ but after the execution of the first high transition we reach the states } \sigma' \text{ and } \theta' \text{ with } \sigma'(L) = 1 \neq \theta'(L) = 2$. 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 $P \equiv H := 4; L := 1; \text{ if } (L = 1) \text{ then skip else } L := H$. $P$ is in the class $\mathcal{SIMP}_{\approx_l}$. In fact, the first branch of the conditional is always executed independently of the value in the high level location.

Since function $\text{Reach}$ is transitive, by Lemma 4 we get that $\mathcal{W}(\approx_l, \approx_l, \text{Reach})$ is persistent, i.e., if a program $P$ starting in a state $\sigma$ is secure, 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 $\mathcal{SIMP}_{\approx_l}$, then also each program $P'$ reachable from $P$ is in $\mathcal{SIMP}_{\approx_l}$. This is illustrated in the following example.

Example 6. Consider the program $P \equiv L := 0; P'$ where

$$
P' \equiv \text{ if } L = 1 \text{ then } L := H \text{ else skip.}
$$

It holds that $P \in \mathcal{SIMP}_{\approx_l}$ since, for each state $\sigma$, $\langle P, \sigma \rangle$ will never perform any high transition. However, the program $P'$ is reachable from $P$ but $P' \notin \mathcal{SIMP}_{\approx_l}$. 

9
A more restrictive class of secure imperative programs can be introduced by instantiating our generalized unwinding with the reachability function $Reach^*$.

**Definition 5.** $(\text{SIMP}^*_{\approx_l})$ A program $P$ belongs to the class $(\text{SIMP}^*_{\approx_l})$ if for each state $\sigma$, $(P,\sigma) \in \mathcal{W}(=_{\approx_l}, \approx_l, Reach^*)$.

**Lemma 5.** $\text{SIMP}^*_{\approx_l} \subseteq \text{SIMP}_{\approx_l}$.

The two classes do not coincide as shown below.

**Example 7.** Consider the program

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

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

Function $Reach^* : \text{Prog} \times \Sigma \rightarrow \mathcal{P}(\text{Prog} \times \Sigma)$ is transitive and then, by Lemma 4, the class $\mathcal{W}(=_{\approx_l}, Reach^*)$ is persistent, i.e., if $(P, \sigma) \in \mathcal{W}(=_{\approx_l}, Reach^*)$, then also each pair $(P', \sigma') \in Reach^*(\langle P, \sigma \rangle)$ is in $\mathcal{W}(=_{\approx_l}, Reach^*)$. Moreover, differently from $(\text{SIMP}_{\approx_l})$, if a program $P$ is in $(\text{SIMP}^*_{\approx_l})$, then also each $P' \in Reach^*(P)$ is in $(\text{SIMP}^*_{\approx_l})$.

**Lemma 6.** Let $P$ be a program. If $P \in \text{SIMP}^*_{\approx_l}$, then for all $P' \in Reach^*(P)$, $P' \in \text{SIMP}^*_{\approx_l}$.

Both $\mathcal{W}(=_{\approx_l}, Reach)$ and $\mathcal{W}(=_{\approx_l}, Reach^*)$ allow us to express timing-sensitive notions of security. This is a consequence of the fact that $\approx_l$ equates programs which exhibit the same timing behavior (see Lemma 3).

**Example 8.** Let us consider the following program:

$$P \equiv \text{if } (H = 0) \text{ then } \{H := H + 1; \text{skip} \} \text{ else } H := 2.$$  

It belongs to neither $(\text{SIMP}_{\approx_l})$ nor $(\text{SIMP}^*_{\approx_l})$. This is due to the fact that if $\langle P, \sigma \rangle \approx_{\text{high}} \langle \{H := H + 1; \text{skip} \}, \sigma \rangle$ for some state $\sigma$, then it does not hold that for each $\theta$ such that $\sigma =_l \theta$ there exists $(R, \theta')$ such that $\langle P, \theta \rangle \rightarrow (R, \theta')$ and $\langle \{H := H + 1; \text{skip} \}, \sigma \rangle \approx_l (R, \theta')$. In fact, if $\theta(H) \neq 0$, $\langle P, \theta \rangle \rightarrow (H := 2, \theta)$ but $\langle \{H := H + 1; \text{skip} \}, \sigma \rangle \not\approx_l (H := 2, \theta)$ because of their different timing behaviour.

In the previous section we observed that the relation $\approx_l$ is not reflexive. However, $\approx_l$ is reflexive on the class $(\text{SIMP}_{\approx_l})$ (and then, by Lemma 5, on $(\text{SIMP}^*_{\approx_l})$).

**Lemma 7.** Let $P$ be a program. If $P \in \text{SIMP}_{\approx_l}$, then $P \approx_l P$.  

10
The converse of Lemma 7 does not hold in general as illustrated in the following example.

**Example 9.** Consider the program

\[
P \equiv \text{if } (H = 1) \text{ then } P_0 \text{ else } P_1
\]

where \( P_0 \equiv \text{while } (H > 1) \text{ do skip and } P_1 \equiv \text{skip.} \) One can prove that \( P \approx_1 P_0 \), i.e., for all states \( \sigma \) and \( \theta \) such that \( \sigma =_l \theta, \langle P, \sigma \rangle \approx_1 \langle P, \theta \rangle \). However, the program \( P \not\in \text{SIMP}_{\approx_1} \). In fact, \( \langle P_0, \sigma \rangle \in \text{Reach}(\langle P, \sigma \rangle) \) and \( \langle P_0, \sigma \rangle \approx_1^{\text{high}} \langle \text{end}, \sigma \rangle \) but it does not hold that for all \( \rho \) such that \( \sigma =_l \rho \) there exist \( R \) and \( \rho' \) such that \( \langle P_0, \rho \rangle \to \langle R, \rho' \rangle \) and \( \langle \text{end}, \sigma \rangle \approx_1 \langle R, \rho' \rangle \). Indeed, if \( \rho(H) > 1 \), \( \langle P_0, \rho \rangle \to \langle \text{skip}; P_0, \rho \rangle \) and \( \langle \text{end}, \sigma \rangle \not\approx_1 \langle \text{skip}; P_0, \rho \rangle \). This is due to the fact that the subprogram \( P_0 \) of \( P \) is not in \( \text{SIMP}_{\approx_1} \).

However, it holds that if \( P' \approx_1 P' \) for all \( P' \in \text{Reach}^*(P) \), then \( P \in \text{SIMP}_{\approx_1}^* \).

**Lemma 8.** Let \( P \) be a program such that \( P' \approx_1 P' \) for all \( P' \in \text{Reach}^*(P) \). It holds \( P \in \text{SIMP}_{\approx_1}^* \).

The next theorem states that property \( \text{SIMP}_{\approx_1}^* \) coincides with the persistent, timing sensitive version of the security property for concurrent programs studied in [4,7,31].

**Theorem 1.** \( P \in \text{SIMP}_{\approx_1}^* \) if and only if \( P' \approx_1 P' \) for all \( P' \in \text{Reach}^*(P) \).

The classes \( \text{SIMP}_{\approx_1} \) and \( \text{SIMP}_{\approx_1}^* \) introduced above are not compositional with respect to the parallel composition constructor as illustrated by the following example.

**Example 10.** Consider the program

\[
P \equiv \text{if } (H = 1) \text{ then } P_0 \text{ else } P_1
\]

where

\[
P_0 \equiv L := 1; \text{ if } (L = 1) \text{ then } L := 2 \text{ else } L := 3
\]

\[
P_1 \equiv L := 1; \text{skip; } L := 2.
\]

We have that for each \( \sigma \) and \( \theta \) such that \( \sigma =_l \theta \) it holds \( \langle P_0, \sigma \rangle \approx_1 \langle P_1, \theta \rangle \), i.e., \( P_0 \approx_1 P_1 \). From this we get that \( P \approx_1 P \). Moreover, it is easy to see that for each \( P' \in \text{Reach}^*(P_0) \cup \text{Reach}^*(P_1) \) it holds \( P' \approx_1 P' \). Hence, by Theorem 1, we can say that \( P \in \text{SIMP}_{\approx_1}^* \). Consider also the program \( Q \equiv L := 0 \). It is immediate to see that \( Q \in \text{SIMP}_{\approx_1}^* \). However, if we consider \( P \parallel Q \) this does not belong to \( \text{SIMP}_{\approx_1}^* \). In fact, if \( \sigma \) and \( \theta \) are such that \( \sigma =_l \theta, \sigma(H) = 1 \) and \( \theta(H) = 0 \) we get that \( \langle P \parallel Q, \sigma \rangle \to \langle P_0 \parallel Q, \sigma \rangle \), while \( \langle P \parallel Q, \theta \rangle \to \langle P_1 \parallel Q, \theta \rangle \). It does not hold \( \langle P_0 \parallel Q, \sigma \rangle \approx_1 \langle P_1 \parallel Q, \theta \rangle \), 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 \( \text{SIMP}^* \) which is obtained by instantiating our generalized unwinding condition with the function \( \text{Reach}^* \) for \( \mathcal{R} \) and the low level bisimilarity \( \sim_l \) defined below for \( \vdash \).

**Definition 6.** The relation \( \sim_l \) over \( P \times \Sigma \) is defined as follows: \( (P, \sigma) \sim_l (Q, \theta) \) if \( \sigma =_l \theta \) and \( P \sim_l Q \).

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

**Definition 7.** (\( \text{SIMP}^* \)) A program \( P \) belongs to the class \( \text{SIMP}^* \) if for each state \( \sigma \), \( (P, \sigma) \in \mathcal{W}(=_l, \sim_l, \text{Reach}^*) \).

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

**Lemma 9.** Let \( P \) be a program. If \( P \in \text{SIMP}^* \), then for all \( P' \in \text{Reach}^*(P) \), \( P' \in \text{SIMP}^* \).

The next theorem shows that the reflexive closure of \( \sim_l \) exactly coincides with the set of programs in \( \text{SIMP}^* \).

**Theorem 2.** \( P \in \text{SIMP}^* \) if and only if \( P \sim_l P \).

Since \( \sim_l \) is exactly the low level bisimulation introduced in [28], the result above also states that the property \( \text{SIMP}^* \) coincides with the security property for multi-threaded programs studied in [1, 25].

The class \( \text{SIMP}^* \) is more restrictive than \( \text{SIMP}^* \approx_l \) and \( \text{SIMP}^* =_l \). This is a consequence of the fact that \( \sim_l \approx_l \).

**Lemma 10.** \( \text{SIMP}^* \subseteq \text{SIMP}^* \approx_l \subseteq \text{SIMP}^* =_l \).

Moreover, \( \text{SIMP}^* \) does not coincide with \( \text{SIMP}^* \approx_l \) as shown by the following example.

**Example 11.** Consider the program \( P \) of Example 10. It holds that \( P \in \text{SIMP}^* \approx_l \). However, it does not hold \( P \in \text{SIMP}^* \). To show this it is sufficient to prove that \( P \not\sim_l P \). Indeed, if \( \sigma \) and \( \theta \) are such that \( \sigma =_l \theta \), \( \sigma(H) = 1 \), and \( \theta(H) = 0 \), we get that \( (P, \sigma) \rightarrow (P_0, \sigma) \) and \( (P, \theta) \rightarrow (P_1, \theta) \). It does not hold \( P_0 \sim_l P_1 \) since \( (P_0, \sigma) \rightarrow (P'_0, \sigma[L/1]) \), where \( P'_0 \equiv \text{if } (L = 1) \text{ then } L := 2 \text{ else } L := 3 \) and \( (P_1, \theta) \rightarrow (P'_1, \theta[L/1]) \), where \( P'_1 \equiv \text{skip; } L := 2 \) and it is plain that \( P'_0 \not\sim_l P'_1 \).
The class $\text{SIMP}^*$ is compositional with respect to the language constructors.

**Theorem 3.** 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 $P_0$ and $P_1$ are in $\text{SIMP}^*$, then also the following programs are in $\text{SIMP}^*$: (1) $\text{skip}$; (2) $L := a_l$, $H := a_h$, and $H := a_l$; (3) $P_0; P_1$; (4) if $b_l$ then $P_0$ else $P_1$; (5) if $b_h$ then $P_0$ else $P_1$, whenever $P_0 \sim_l P_1$; (6) while $b_l$ do $P_0$; (7) $P_0 | P_1$.

Theorem 3 does not provide a procedure to decide whether $P \in \text{SIMP}^*$. This is due to the request $P_0 \sim_l P_1$ in item (5). Moreover, a program $P$ could be in $\text{SIMP}^*$ even if it does not satisfy the conditions of Theorem 3.

**Example 12.** Consider $P \equiv \text{if } (L = 1) \text{ then } L := H - H \text{ else } L := 2$.

The program $L := H - H$ is in $\text{SIMP}^*$, since for each state $\sigma$ it holds $\langle L := H - H, \sigma \rangle \rightarrow \langle \text{end}, \sigma[L/0] \rangle$. However it does not satisfy any of the conditions of Theorem 3. As a consequence, by applying Theorem 3 we cannot prove that $P \in \text{SIMP}^*$.

In the next section we exploit the decidability of first-order formulae over the reals to get sound and accurate proof systems both for $\sim_l$ and for $\text{SIMP}^*$.

We conclude this section by showing that all the security properties introduced above imply the timing-sensitive (lockstep) non-interference principle [25, 31].

**Theorem 4. (Soundness)** Let $P$ be a program such that $P \in \text{SIMP}_{\sim_l}$ (resp. $\text{SIMP}^*_{\sim_l}$, $\text{SIMP}^*$). For each state $\sigma$ and $\theta$ such that $\sigma =_l \theta$,

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

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 $\sigma$, the LTS associated to $\langle P, \sigma \rangle$ could be infinite.

**Example 13.** Consider the program $P \equiv \text{while } (1 = 1) \text{ do } L := L + 1$ and a state $\sigma$ such that $\sigma(L) = 1$. The LTS associated to $\langle P, \sigma \rangle$ consists of an infinite chain of derivation steps to different pairs:

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

$\rightarrow \langle P, \sigma[L/n] \rangle \rightarrow \ldots$

Another difficulty arises from the fact that even if we restrict ourselves to terminating programs, the relation $\sim_l$ is not decidable.

**Lemma 11.** The relation $\sim_l \subseteq (\mathbb{P})^2$ is undecidable.
In order to cope with this problem, we exploit Tarski decidability result for first order formulae over the reals [32] to define a decidable binary relation ≏_l over programs which entails ∼_l. Let \( \mathbb{R} \) be the set of real numbers. A real state is a function \( \sigma^r : \mathbb{L} \cup \mathbb{H} \to \mathbb{R} \), i.e., a state in which the variables range over the reals. Two real states \( \sigma^r \) and \( \theta^r \) are low level equivalent, \( \sigma^r =_l \theta^r \), if they assign the same values to the low level variables. We start by defining a decidable binary relation \( \equiv_l \) over expressions.

**Definition 8. (\( \equiv_l \) over \( A_{\text{exp}} \) and \( B_{\text{exp}} \))** Let \( a_1, a_2 \in A_{\text{exp}} \). We say that \( a_1 \) and \( a_2 \) are low level equivalent, denoted by \( a_1 \equiv_l a_2 \), if for all real states \( \sigma^r, \theta^r \) such that \( \sigma^r =_l \theta^r \) it holds

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

Analogously, if \( b_1, b_2 \in B_{\text{exp}} \), we say that \( b_1 \) and \( b_2 \) are low level equivalent, denoted by \( b_1 \equiv_l b_2 \), if for all real states \( \sigma^r, \theta^r \) such that \( \sigma^r =_l \theta^r \) it holds

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

Notice that the relation \( \equiv_l \) is symmetric but not reflexive.

**Example 14.** Consider the expression \( a = H + 1 \). It does not hold that \( a \equiv_l a \). In fact, if we consider the states \( \sigma^r \) and \( \theta^r \) 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 \( \langle a, \theta^r \rangle \to 3 \).

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 12.** Let \( a_1, a_2 \in A_{\text{exp}}, b_1, b_2 \in B_{\text{exp}} \) such that \( a_1 \equiv_l a_2 \) and \( b_1 \equiv_l b_2 \).

If \( \sigma \) and \( \theta \) 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 \( \langle b_2, \theta \rangle \to v \).

The converse of Lemma 12 is not true, i.e., it can be the case that two expressions are always mapped to the same value by two states which are low level equivalent, but they are not \( \equiv_l \)-equivalent.

**Example 15.** Let \( b \) be \( H^2 \neq 2 \). It does not hold \( b \equiv_l b \). 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 \( \sigma \) it holds \( \langle b, \sigma \rangle \to true \).

**Lemma 13.** The relation \( \equiv_l \subseteq (A_{\text{exp}} \times A_{\text{exp}}) \cup (B_{\text{exp}} \times B_{\text{exp}}) \) is decidable.

Based on the definition of \( \equiv_l \) over arithmetic and boolean expressions, we define the relation \( \equiv_l \) over \( \mathbb{F}^2 \) 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., \( l(\text{end}) = l(\text{skip}) = l(X := a) = 1, l(P_0; P_1) = l(P_0) + l(P_1) + 1, l(\text{if } b \text{ then } P_0 \text{ else } P_1) = l(P_0) + l(P_1) + 2, l(\text{while } b \text{ do } P) = l(P) + 1. \)
Fig. 2. The relation $\succeq_i$ rules for skip, end, $\Leftarrow$, $|$; and if.
Definition 9. \((\equiv_l \text{ over } \mathbb{P}^2)\) The binary relation \(\equiv_l\) over \(\mathbb{P}^2\) is defined by the rules given in Figure 2 and Figure 3.

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

\[
\frac{P \equiv_l R \text{ and } R \equiv_l Q}{P \equiv_l Q}
\]

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 \(\equiv_l\) as follows: if we have to test \(P \equiv_l Q\) we first find all the subprograms of \(P\) and \(Q\) which are \(\equiv_l\)-equivalent with \(\text{skip}\); we replace them by \(\text{skip}\) in \(P\) and \(Q\); we test \(\equiv_l\) on the programs obtained in this way.

Lemma 14. The relation \(\equiv_l \subseteq \mathbb{P} \times \mathbb{P}\) is decidable and it entails the relation \(\sim_l\).

As a consequence of Theorem 2, we can exploit the proof system for \(\equiv_l\) to check if a program is in \(\text{SIMP}^*\).

Theorem 5. If \(P \equiv_l P\) then \(P \in \text{SIMP}^*\).

Example 16. Let the program

\[P \equiv \text{if } (H = 0)\text{then } \{L := 1; H := 1\} \text{ else } \{L := 1; H := 2\} .\]

By applying the proof system for \(\equiv_l\) defined above, one can easily check that \(P \equiv_l P\); and then \(P \in \text{SIMP}^*\).
Consider the program $P \equiv \text{while} (L + H > H) \text{do} L := 1$ of the introduction. By applying the Tarski decidability result for first order formulae over the reals, we can prove that $(L + H > H) \equiv_l (L + H > H)$. Hence, since $L := 1 \equiv_l L := 1$, by using our proof system one can derive that $P \equiv_l P$, i.e., $P \in \text{SIMP}^*$.

The decidability of $\equiv_l$ depends on Tarski’s result. However, the result in [32] based on quantifier elimination 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 created the first practical quantifier elimination software $\text{Qepcad}$ [9]. A quantifier elimination procedure based on Collins’ algorithm [8] and called $\text{Cylindrical Decomposition}$ has been integrated in Mathematica starting from Version 5.0. Since, our formulae have no quantifier alternations the symbolic computation module of Marple (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 $\equiv_l$, we denote by $c(P)$ the time complexity of evaluating the $\equiv_l$ equivalences on the expressions occurring in $P$. Notice that $c(P)$ 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. Our formulae are closed and have only universal quantifications, hence the number $q$ of quantifier alternations in our analysis is always zero. So, the methods proposed by Grigoriev [13] and Renegar [23] in our case are polynomial in $d$ and exponential in $v$.

**Theorem 6.** Let $P$ be a program. The complexity of deciding $P \equiv_l P$ is $O(c(P) \ast l(P))$.

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

**Definition 10.** ($\mathcal{W}(\equiv_l)$) The class $\mathcal{W}(\equiv_l)$ is defined by the rules given in Figure 4.

**Lemma 15.** The class $\mathcal{W}(\equiv_l) \subseteq \mathbb{P}$ is decidable. Moreover, $P \in \mathcal{W}(\equiv_l)$ if and only if $P \equiv_l P$.

As a consequence, we get the following result.

**Theorem 7.** If $P \in \mathcal{W}(\equiv_l)$, then $P \in \text{SIMP}^*$.
Fig. 4. The class $\mathcal{W}(\approx_i)$. 
The following example shows that there are programs which are in $\text{SIMP}^*$ but not in $\mathcal{W}(\cong_l)$.

**Example 17.** Let $P \equiv \{\text{while } \text{true do } L := 1\}; L := H$. Since $L := H$ is not in $\text{Reach}^*(P)$ we get that $P \in \text{SIMP}^*$. 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 $P \equiv \text{if } (H^2 \neq 2) \text{ then } L := 1 \text{ else } L := 2$. It holds that $H^2 \neq 2$ is always true over the integers, hence it follows that $P \in \text{SIMP}^*$. However, $(H^2 \neq 2) \not\equiv_l (H^2 \neq 2)$ and $L := 1$ is not low level equivalent to $L := 2$, hence $P \not\in \mathcal{W}(\cong_l)$.

Moreover, there are programs which can be proved to be secure using our proof system, but not using the type systems described in, e.g., [1, 25].

**Example 18.** Let $P \equiv \text{if } (H^6 + 2H^5 - 5H^4 - H^2 + 2H + 1 \geq 0) \text{ then } L := 1 \text{ else } L := 2$. This program is in $\mathcal{W}(\cong_l)$ since the boolean condition is always true and the first branch of the if constructor is always taken.

## 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., [15, 27, 36], non-interference 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 timing-sensitive 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 11.** 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 19.** 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, and the total value of the sum $H_1 + H_2$. Hence, the program $P \equiv \text{if } (2H_1 - 10 > 0) \text{ then skip else } L := H_1 + H_2$ should be considered secure. Notice that, when $H_1 \leq 5$, by observing the execution of $P$ the low level user can also infer that $H_2 \geq \ell - 5$, where $\ell$ is the value of $L$ at the end of the execution. This example shows that any information obtained by combining elements of $D$ can be downgraded to the low level user.
Intuitively, $D$ represents a finite abstraction of all information that can be actually downgraded. We define the concretization $\gamma(D)$ of $D$ representing all the declassifiable expressions which are deducible from $D$.

**Definition 12.** 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 $e, e' \in \text{Aexp} \cup \text{Bexp}$, $e' \in \gamma(D)$ and for all states $\sigma$, $\langle e, \sigma \rangle \rightarrow v$ if and only if $\langle e', \sigma \rangle \rightarrow v$, then $e \in \gamma(D)$, (3) if $e$ is defined through the grammar for $\text{Aexp} \cup \text{Bexp}$ given in Section 2 with $a_0, a_1, b, b_0, b_1, X \in \gamma(D)$ then $e \in \gamma(D)$.

**Example 20.** Consider the declassifiable set $D$ of Example 19. 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 12 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 [15, 27] 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 [27], 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 $\sigma$ and $\theta$ be two states and $D$ be a declassifiable set. We write $\sigma =_{L,D} \theta$ if $\sigma =_L \theta$ and $\langle d, \sigma \rangle \rightarrow c$ if and only if $\langle d, \theta \rangle \rightarrow c$, for all $d \in D$. By Definition 12, it follows that $\sigma =_{L,D} \theta$ if and only if $\sigma =_{L,\gamma(D)} \theta$. In order to deal with delimited release we introduce the notion of *strong low-$D$ level bisimulation* which is obtained from Definition 2 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 $(P, Q) \in \mathcal{B}$. Moreover, the relation $\sim_{l,D}$ over $\mathbb{P} \times \Sigma$ is defined by: $\langle P, \sigma \rangle \sim_{l,D} \langle Q, \theta \rangle$ if $\sigma =_{L,D} \theta$ and $P \sim_{l,D} Q$.

**Definition 13.** Let $D$ be a declassifiable set. The relation $\sim_{l,D}$ over $\mathbb{P} \times \Sigma$ is defined as follows: $\langle P, \sigma \rangle \sim_{l,D} \langle Q, \theta \rangle$ if $\sigma =_{L,D} \theta$ and $P \sim_{l,D} Q$.

**Lemma 16.** Let $D$ be a declassifiable set. The relations $\sim_{l,D}$ and $\sim_{l,D}$ are partial equivalence relations.

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

**Definition 14. (SIMP$^*_D$)** Let $D$ be a declassifiable set. A program $P$ belongs to the class $\text{SIMP}^*_D$ if for each state $\sigma$, $\langle P, \sigma \rangle \in \mathcal{W}^D(=_{L,D}, \sim_{l,D}, \text{Reach}^*)$.

**Example 21.** Consider the program

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

and the set $D = \{H = 0\}$. In this case $P \in \text{SIMP}^*_D$. In fact, for all states $\sigma$ and $\theta$ such that $\sigma =_{L,D} \theta$, if $\langle P, \sigma \rangle \rightarrow^* (L := 0; H := 0, \sigma)$ then also $\langle P, \theta \rangle \rightarrow^*$...
\(\langle L := 0; H := 0, \theta \rangle\) and \(\{ L := 0; H := 0 \}\) \(\sim_{l,D} \{ L := 0; H := 0 \}\). Analogously, if \(\langle P, \sigma \rangle \xrightarrow{\text{high}} \langle L := 1; H := 1, \sigma \rangle\) then also \(\langle P, \theta \rangle \xrightarrow{\text{high}} \langle L := 1; H := 1, \theta \rangle\) 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 \(P \notin \textup{SIMP}_D\). In fact, given a state \(\sigma\), \(\langle P, \sigma \rangle \xrightarrow{\text{high}} \langle L := H_1, \sigma[H_1/H_2] \rangle\). However, it does not hold that for any \(\theta\) such that \(\sigma =_{L,D} \theta, \langle P, \theta \rangle \xrightarrow{\text{high}} \langle L := H_1, \theta[H_1/H_2] \rangle\) 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 now that the program \(P \equiv L := H_1; H_1 := H_2\) with \(D = \{ H_1 \}\) does not belong to \(\textup{SIMP}_D\), even if a low level observer can only infer the value of \(H_1\). This reflects the fact that, if we consider \(P|Q\) with \(Q \equiv L_1 := H_1\) then there are computations in which, by observing \(L_1\), the low level user can deduce the value of the secret variable \(H_2\).

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 \(\textup{SIMP}_D^*\).

**Theorem 8.** Let \(D\) be a declassifiable set. \(P \in \textup{SIMP}_D^*\) if and only if \(P \in \textup{SIMP}_D^{\gamma(D)}\).

The class \(\textup{SIMP}_D^*\) satisfies compositional properties similar to those of Theorem 3 for the class \(\textup{SIMP}^*\). In particular it is compositional with respect to the sequential and parallel operators. We can also prove that \(W^D(=_{L,D}, \sim_{l,D}, \text{Reach}^*)\) is persistent in the sense that if \(\langle P, \sigma \rangle\) is in \(W^D(=_{L,D}, \sim_{l,D}, \text{Reach}^*)\), then also each pair \(\langle P', \sigma' \rangle \in \text{Reach}^*(\langle P, \sigma \rangle)\) is in \(W^D(=_{L,D}, \sim_{l,D}, \text{Reach}^*)\).

Moreover, if \(P\) is in \(\textup{SIMP}_D^*\), then also each \(P' \in \text{Reach}^*(P)\) is in \(\textup{SIMP}_D^*\).

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

**Theorem 9.** Let \(D\) be a declassifiable set. \(P \in \textup{SIMP}_D^*\) if and only if \(P \sim_{l,D} P\).

We show that the family of security properties \(\textup{SIMP}_D^*\) implies a timing-sensitive version of the delimited release property studied in [27] for sequential programs.

**Theorem 10.** (Soundness) Let \(D\) be a declassifiable set and \(P\) be a program. If \(P \in \textup{SIMP}_D^*\), then for all states \(\sigma\) and \(\theta\) such that \(\sigma =_{L,D} \theta\),

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

We can decide whether a program belongs to a class \(\textup{SIMP}_D^*\) 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 \(\approx_{l,D}\) over programs which entails \(\sim_{l,D}\). The relation \(=_{L,D}\) is extended to real states in the natural way.
Definition 15. Let \(a_1, a_2 \in \text{Aexp}\) and \(D\) be a declassifiable set. We say that \(a_1\) and \(a_2\) are low-D level equivalent, denoted by \(a_1 \equiv_{l,D} a_2\), if for all real states \(\sigma^r, \theta^r\) such that \(\sigma^r = L,D \theta^r\) it holds \(\langle a_1, \sigma^r \rangle \rightarrow r\) if and only if \(\langle a_2, \theta^r \rangle \rightarrow r\). Analogously, if \(b_1, b_2 \in \text{Bexp}\) we say that \(b_1\) and \(b_2\) are low-D level equivalent, denoted by \(b_1 \equiv_{l,D} b_2\), if for all \(\sigma^r, \theta^r\) such that \(\sigma^r = L,D \theta^r\) it holds \(\langle b_1, \sigma^r \rangle \rightarrow v\) if and only if \(\langle b_2, \theta^r \rangle \rightarrow v\).

Lemma 17. Let \(D\) be a declassifiable set. The relation \(\equiv_{l,D} \subseteq (\text{Aexp} \times \text{Aexp}) \cup (\text{Bexp} \times \text{Bexp})\) is decidable.

The following definition is used to define a binary relation \(\bowtie_{l,D}\) over \(\mathbb{P}^2\) which entails \(\sim_{l,D}\).

Definition 16. \((\bowtie_{l,D})\) Let \(D\) be a declassifiable set and \(H, K\) be high level locations. We say that the assignments \(H := a_0\) and \(K := a_1\) are low-D level equivalent, denoted \(H := a_0 \bowtie_{l,D} K := a_1\), if for all real states \(\sigma^r, \theta^r\) such that \(\sigma^r = L,D \theta^r\) and for all \(d \in D\) it holds \(\langle d, \sigma^r[H/\sigma^r(a_0)] \rangle \rightarrow c\) if and only if \(\langle d, \sigma^r[K/\sigma^r(a_1)] \rangle \rightarrow c\).

Lemma 18. The relation \(\bowtie_{l,D}\) is decidable and \(H := a_0 \bowtie_{l,D} K := a_1\) entails \(H := a_0 \sim_{l,D} K := a_1\).

Definition 17. \((\bowtie_{l,D})\) Let \(D\) be a declassifiable set. The binary relation \(\bowtie_{l,D}\) over \(\mathbb{P}^2\) is obtained from the rules given for \(\bowtie_1\) by replacing \(\bowtie_1\) with \(\bowtie_{l,D}\) and adding the side condition \(H := a_0 \bowtie_{l,D} K := a_1\) in the second rule for the assignment command.

Lemma 19. The relation \(\bowtie_{l,D} \subseteq \mathbb{P} \times \mathbb{P}\) is decidable and entails \(\sim_{l,D}\).

We finally define the decidable class \(W(\bowtie_{l,D})\) of secure programs by modifying our previous proof system as follows: the class \(W(\bowtie_{l,D})\) is obtained from the rules given in Figure 4 by replacing \(\bowtie_1\) with \(\bowtie_{l,D}\) and adding the side condition \(H := a \bowtie_{l,D} H := a\) in the second line, second rule for the assignment command.

Theorem 11. If \(P \in W(\bowtie_{l,D})\), then \(P \in \text{SIMP}_D^*\).

Example 22. Consider the program

\[
P \equiv \text{if } (H = 0) \text{ then } \{L := 0; H := 0\} \text{ else } \{L := 1; H := 1\}
\]

and the set \(D = \{H = 0\}\) of Example 21. It is easy to prove that both \(\{L := 0; H := 0\} \in W(\bowtie_{l,D})\) and \(\{L := 1; H := 1\} \in W(\bowtie_{l,D})\). Moreover, we can prove that \((H = 0) \bowtie_{l,D} (H = 0)\). Hence, by applying the rules of \(W(\bowtie_{l,D})\), we get that \(P \in W(\bowtie_{l,D})\), i.e., \(P \in \text{SIMP}_D^*\).
In this paper we introduce a generalized unwinding schema for the definition of non-interference properties of programs described in a simple imperative language admitting parallel executions on a shared memory. 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, timing-sensitive, non-interference properties for concurrent programs.

There is a widespread literature on secure information flow in imperative languages (see [26]). Many works concern the definition of timing-insensitive non-interference properties controlling the end-to-end behaviour of programs. In the setting of concurrency, these kind of properties have been studied by Volpano and Smith in [31] and by Boudol and Castellani in [7]. They define type systems to ensure the property of non-interference expressed in terms of the following weak low bisimulation

\[ \approx_L : \text{let } \rightarrow \text{ stand for one or zero transitions, thus } (\langle P, \sigma \rangle, \langle Q, \theta \rangle) \in \approx_L \text{ if } \sigma =_l \theta \text{ and whenever } (\langle P, \sigma \rangle) \rightarrow (\langle P', \sigma' \rangle) \text{ then there exists } (Q', \theta') \text{ such that } (Q, \theta) \rightarrow (Q', \theta') \text{ and } ((\langle P', \sigma' \rangle, \langle Q', \theta' \rangle) \in \approx_L, \text{ and viceversa. The relation } \approx_L \text{ is a partial equivalence relation and a program } P \text{ is secure if } (\langle P, \sigma \rangle) \approx_L (\langle P, \theta \rangle) \text{ for all } \sigma \text{ and } \theta \text{ such that } \sigma =_l \theta. \]

We can instantiate our generalized unwinding condition with \( \approx_L \) obtaining, for instance, the classes \( W (=_l, \approx_L, \text{Reach}) \) and \( W (=_l, \approx_L, \text{Reach}^*) \) which both imply the property studied in [7, 31]. 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., \( P_1 \equiv \text{while } (H = 0) \text{ do skip} \) and \( P_2 \equiv L := 1 \), give rise to insecure programs when composed through, e.g., \( P_1; P_2 \) and \( P_1|P_2 \). In this paper we focus on compositional properties and then we overlook those timing-insensitive properties.

Timing-sensitive bisimulations have been considered by various authors, e.g., [1, 14, 28, 25, 30], for modelling timing attacks which include the ability to observe the timing behavior of the system. In particular, the security property characterized by our \( \text{SIMP}^* \) class exactly corresponds to the strong security property studied by, e.g., Sabelfeld and Mantel in [25] and by Agat in [1].

Security properties for programs admitting downgrading have been recently studied by several authors, e.g., [6, 20, 21, 27, 36, 17]. In this paper, we follow the approach of [15, 27]: 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 definition of the secure class \( \text{SIMP}^*_D \). 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 non-interference 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 [27].

As far as verification is concerned, we provide techniques which are more precise than the type-based proof methods presented in all the above mentioned works. 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, while $(L + H > H)$ do $L := 1$ is secure. This cannot be captured by previous type systems.

References

Appendix

Proof of Lemma 1 If \( \langle P, \sigma \rangle \approx_l \langle Q, \theta \rangle \), then there exists a low level bisimulation \( \mathcal{B} \) such that it holds \( \langle \langle P, \sigma \rangle, \langle Q, \theta \rangle \rangle \in \mathcal{B} \). Hence if \( \langle P, \sigma \rangle \rightarrow \langle P', \sigma' \rangle \) we have that \( \langle Q, \theta \rangle \rightarrow \langle Q', \theta' \rangle \) with \( \langle \langle P', \sigma' \rangle, \langle Q', \theta' \rangle \rangle \in \mathcal{B} \), 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 \( \approx_l \subseteq \mathcal{P}^2 \) is symmetric and transitive since \( \approx_l \) is symmetric and transitive.

Claim. Let \( P \) be a program. Either \( P \approx_l P \) (\( P \sim_l P \)) or for each \( P' \in \mathcal{P} \) it holds \( P \not\approx_l P' \) (\( P \not\sim_l P' \)).

Proof of Lemma 3 (1). By induction on \( n \).

\begin{itemize}
  \item Base: \( n = 1 \). We immediately have the thesis by definition of \( \approx_l \).
  \item 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.
\end{itemize}

The proof of (2) is analogous.

Proof of Lemma 4 Let \( R \) be transitive, \( \langle P, \sigma \rangle \in \mathcal{W}(\approx_l, \not\approx_l, R) \), and \( \langle F, \psi \rangle \in R(\langle P, \sigma \rangle) \). If \( \langle F', \psi' \rangle \in R(\langle F, \psi \rangle) \), then by transitivity we have that \( \langle F', \psi' \rangle \in R(\langle P, \sigma \rangle) \). Hence we get that if \( \langle F', \psi' \rangle \rightarrow^{\text{high}} \langle G', \varphi' \rangle \), then for each \( \pi \) such that \( \psi' =_l \pi' \) there exists \( \langle R', \rho' \rangle \) such that \( \langle F', \pi' \rangle \rightarrow \langle R', \rho' \rangle \) with \( \langle G', \varphi' \rangle \not\approx_l \langle R', \rho' \rangle \), i.e., the thesis.

Proof of Lemma 5 This is a consequence of the fact that for each \( P \) and \( \sigma \) it holds that
\[
\text{Reach}(\langle P, \sigma \rangle) \subseteq \{ \langle Q, \theta \rangle \mid Q \in \text{Reach}^*(P) \}.
\]
**Proof of Lemma 6** Let \( P' \in \text{Reach}^*(P) \), i.e., \( \langle P', \sigma' \rangle \in \text{Reach}^*(\langle P, \sigma \rangle) \) for some \( \sigma \) and \( \sigma' \). By definition of \( \text{Reach}^* \), \( \langle P', \theta \rangle \in \text{Reach}^*(\langle P, \sigma \rangle) \) for each state \( \theta \). Hence, by persistence of \( W(=_{\leq 1}, \approx_{= 1}, \text{Reach}^*) \), \( \langle P', \theta \rangle \in W(=_{\leq 1}, \approx_{= 1}, \text{Reach}^*) \), i.e., \( P' \in \text{SIMP}_{\approx_{= 1}} \).

The following claim will be used below.

Claim. For each \( \psi \) and \( \pi \) such that \( \psi \equiv \pi \), if \( \langle F, \psi \rangle \xrightarrow{\log} \langle F', \psi' \rangle \), then \( \langle F, \pi \rangle \xrightarrow{\log} \langle F', \pi' \rangle \) with \( \pi' =_1 \psi' \).

**Proof.** By structural induction on programs. \( \square \)

**Proof of Lemma 7** Assume that \( P \in \text{SIMP}_{\approx_{= 1}} \). Then for all states \( \sigma \) and \( \theta \), \( \langle P, \sigma \rangle, \langle P, \theta \rangle \in W(=_{\leq 1}, \approx_{= 1}, \text{Reach}) \). Hence, in order to prove that \( P \approx_{= 1} P \), it is sufficient to show that for all \( \sigma \) and \( \theta \) such that \( \langle P, \sigma \rangle, \langle P, \theta \rangle \in W(=_{\leq 1}, \approx_{= 1}, \text{Reach}) \) and \( \sigma =_1 \theta \), it holds \( \langle P, \sigma \rangle \approx_{= 1} \langle P, \theta \rangle \). Consider the binary relation

\[
S = \{(\langle P, \sigma \rangle, \langle P, \theta \rangle) \mid \langle P, \sigma \rangle, \langle P, \theta \rangle \in W(=_{\leq 1}, \approx_{= 1}, \text{Reach}), \sigma =_1 \theta\} \cup \{(\langle P, \sigma \rangle, \langle Q, \theta \rangle) \mid \langle P, \sigma \rangle \approx_{= 1} \langle Q, \theta \rangle\}.
\]

We show that \( S \) is a low level bisimulation \( \approx_{= 1} \).

If \( \langle P, \sigma \rangle \xrightarrow{\text{high}} \langle P', \sigma' \rangle \), then from the fact that \( \langle P, \sigma \rangle \in W(=_{\leq 1}, \approx_{= 1}, \text{Reach}) \) we have that \( \langle P, \sigma \rangle \rightarrow \langle P'', \sigma'' \rangle \) with \( \langle P'', \sigma'' \rangle \approx_{= 1} \langle P'', \theta'' \rangle \). Hence, by definition of \( S \), \( \langle \langle P'', \sigma'' \rangle, \langle P'', \theta'' \rangle \rangle \in S \).

If \( \langle P, \sigma \rangle \xrightarrow{\log} \langle P', \sigma' \rangle \), then by Claim 6 we have that \( \langle P, \theta \rangle \xrightarrow{\log} \langle P', \theta' \rangle \) with \( \sigma' =_1 \theta' \). By Lemma 4, since \( \text{Reach} \) is transitive, we have that, both \( \langle P', \sigma' \rangle, \langle P', \theta' \rangle \in W(=_{\leq 1}, \approx_{= 1}, \text{Reach}) \) and \( \langle P', \theta' \rangle \in W(=_{\leq 1}, \approx_{= 1}, \text{Reach}) \). Hence we have that \( \langle \langle P', \sigma' \rangle, \langle P', \theta' \rangle \rangle \in S \), i.e., the thesis. \( \square \)

**Proof of Lemma 8** We prove that for all \( \sigma \in \Sigma \), \( \langle P, \sigma \rangle \in W(=_{\leq 1}, \approx_{= 1}, \text{Reach}^*) \). Let \( \langle F, \psi \rangle \in \text{Reach}^*(\langle P, \sigma \rangle) \). Then, by definition of \( \text{Reach}^* \), \( F \in \text{Reach}^*(P) \). Hence, by hypothesis, \( F \approx_{= 1} F \).

If \( \langle F, \psi \rangle \xrightarrow{\text{high}} \langle G, \varphi \rangle \), then, since \( F \approx_{= 1} F \), for each \( \pi \in \Sigma \) such that \( \psi =_1 \pi \) it holds \( \langle F, \pi \rangle \rightarrow \langle R, \rho \rangle \) with \( \langle G, \varphi \rangle \approx_{= 1} \langle R, \rho \rangle \), i.e., the thesis. \( \square \)

**Proof of Theorem 1** \( \Leftarrow \) This has been proved in Lemma 8. \( \Rightarrow \) If \( P \in \text{SIMP}_{\approx_{= 1}} \), then by Lemma 6 we get that for each \( P' \in \text{Reach}^*(P) \) it holds \( P' \in \text{SIMP}_{\approx_{= 1}} \). Hence, by Lemma 5 we have that for each \( P' \in \text{Reach}^* \) it holds \( P' \in \text{SIMP}_{\approx_{= 1}} \). By Lemma 7 we can conclude that for each \( P' \in \text{Reach}^* \) it holds \( P' \approx_{= 1} P' \).

**Proof of Lemma 9** Let \( P' \in \text{Reach}^*(P) \), i.e., \( \langle P', \sigma' \rangle \in \text{Reach}^*(\langle P, \sigma \rangle) \) for some \( \sigma \) and \( \sigma' \). By definition of \( \text{Reach}^* \), \( \langle P', \theta \rangle \in \text{Reach}^*(\langle P, \sigma \rangle) \) for each state \( \theta \). Hence, by persistence of \( W(=_{\leq 1}, \approx_{= 1}, \text{Reach}^*) \), \( \langle P', \theta \rangle \in W(=_{\leq 1}, \approx_{= 1}, \text{Reach}^*) \), i.e., \( P' \in \text{SIMP}^* \).

The following claim will be used below.


Claim. If \( P \sim_1 P \), then for all \( F \in \text{Reach}^*(P) \) it holds that \( F \sim_1 F \).

Proof. If \( F \in \text{Reach}^*(P) \), 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 \( \langle P_{i-1}, \sigma_{i-1} \rangle \rightarrow \langle P_i, \theta_i \rangle \). We prove that \( P_n \sim_1 P_n \) by induction on \( n \).

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

Step. We proved the thesis for \( n = m \) and we consider \( n = m + 1 \). We have that \( \langle P_m, \sigma_m \rangle \rightarrow \langle P_{m+1}, \theta_{m+1} \rangle \). Since by inductive hypothesis it holds \( P_m \sim_1 P_m \) and it holds that \( \sigma_m \equiv \sigma_m \) and \( \langle P_m, \sigma_m \rangle \rightarrow \langle P_{m+1}, \theta_{m+1} \rangle \) we get that \( \langle P_m, \sigma_m \rangle \rightarrow \langle Q, \mu \rangle \) with \( P_{m+1} \sim_1 Q \) and \( \theta_{m+1} = \mu \). By Claim 6 from \( P_{m+1} \sim_1 Q \) we get \( P_{m+1} \sim_1 P_{m+1} \). \( \square \)

Proof of Theorem 2 \( \Rightarrow \) Consider the binary relation

\[
S = \{(P, P) \mid P \in \text{SIMP}^*\} \cup \{(P, Q) \mid P \sim_1 Q\}
\]

We show that \( S \) is a strong low level bisimulation \( \sim_1 \). This follows from the following cases. Let \( \sigma \) and \( \theta \) be two states such that \( \sigma =_1 \theta \).

If \( \langle P, \sigma \rangle \mathrel{\text{high}} \langle P', \sigma' \rangle \), then since \( P \in \text{SIMP}^* \), \( \langle P, \sigma \rangle \in \mathcal{W}(\equiv_1, \sim_1, \text{Reach}^*) \) and then \( \langle P, \theta \rangle \rightarrow \langle P', \theta' \rangle \) with \( \theta' \equiv \theta'' \) and \( P' \sim_1 P'' \).

Hence, by definition of \( S \), \( (P', P'') \in S \).

If \( \langle P, \sigma \rangle \mathrel{\text{low}} \langle P', \sigma' \rangle \), then by Claim 6 we have that \( \langle P, \theta \rangle \mathrel{\text{low}} \langle P', \theta' \rangle \) with \( \sigma' \equiv \theta' \). By Lemma 9, we have that \( P' \in \text{SIMP}^* \) and then, by definition of \( S \), \( (P', P') \in S \), i.e., the thesis.

\( \Leftarrow \) Let \( P \) be a program such that \( P \sim_1 P \). Let \( \sigma, \psi \in \Sigma \) and \( \langle F, \psi \rangle \in \text{Reach}^*(\langle P, \sigma \rangle) \). Then \( F \in \text{Reach}^*(P) \) and, by Claim 6, \( F \sim_1 F \). Let \( \pi \) be such that \( \psi =_1 \pi \).

If \( \langle F, \psi \rangle \mathrel{\text{high}} \langle G, \varphi \rangle \), then, since \( F \sim_1 F \), \( \langle F, \pi \rangle \rightarrow \langle R, \rho \rangle \) with \( \varphi =_1 \rho \) and \( G \sim_1 R \), i.e., the thesis. \( \square \)

Proof of Lemma 10 We only have to prove that \( \text{SIMP}^* \subseteq \text{SIMP}^*_{\equiv_1} \). If \( P \in \text{SIMP}^* \) by Lemma 9 we get that for each \( P' \in \text{Reach}^*(P) \) it holds \( P' \in \text{SIMP}^* \).

Hence by Theorem 2 we have that for each \( P' \in \text{Reach}^*(P) \) it holds \( P' \sim_1 P' \).

Since \( \sim_1 \subseteq \equiv_1 \), we get that for each \( P' \in \text{Reach}^*(P) \) it holds \( P' \sim_1 P' \), i.e., by Theorem 1, the thesis. \( \square \)

Proof of Theorem 3 The only interesting cases are items (3) and (7). We exploit the fact that \( P \in \text{SIMP}^* \) iff \( P \sim_1 P \). We prove that if \( P, P', Q, Q' \) are such that \( P \sim_1 P' \) and \( Q \sim_1 Q' \), then \( P; Q \sim_1 P; Q \) and \( P|Q \sim_1 P|Q \).

From these the thesis immediately follows. It is sufficient to show that

\[
S = \{(P; Q, P'; Q'), (P|Q, P'|Q') \mid P \sim_1 P' \text{ and } Q \sim_1 Q'\} \cup \{(P, Q) \sim_1 P|Q\}
\]

is a strong low level bisimulation.

Let \( \sigma =_1 \theta \). If \( \langle P; Q, \sigma \rangle \rightarrow \langle P_1; Q, \sigma_1 \rangle \), then \( \langle P, \sigma \rangle \rightarrow \langle P_1, \sigma_1 \rangle \). Hence, since \( P \sim_1 P' \), we get that \( \langle P', \theta \rangle \rightarrow \langle P'_1, \theta_1 \rangle \) with \( P'_1 \sim_1 P'_1 \) and \( \sigma_1 =_1 \theta_1 \). From this last we get \( \langle P'_1; Q', \theta \rangle \rightarrow \langle P'_1; Q', \theta_1 \rangle \) with \( (P'_1; Q, P'_1; Q') \in S \). All the remaining cases are similar. \( \square \)
Proof of Theorem 4 It is sufficient to prove the theorem for $P \in \text{SIMP}_{\equiv_l}$. Then the thesis follows from the fact that $\text{SIMP}^* \subseteq \text{SIMP}^*_{\equiv_l} \subseteq \text{SIMP}_{\equiv_l}$.

By Lemma 7, since $\sigma \equiv_l \theta$, we have that $\langle P, \sigma \rangle \equiv_l \langle P, \theta \rangle$. Then, by Lemma 3, we get that $\langle P, \theta \rangle$ reaches a pair $\langle P', \theta' \rangle$ with $\langle P', \theta' \rangle \equiv_l \langle \text{end}, \sigma' \rangle$. Hence we immediately have $\sigma' \equiv_l \theta'$. Moreover, since $\text{end}$ is not bisimilar to any program, it must be $P' \equiv \text{end}$. \qed

Proof of Lemma 11 A diophantine equation is an equation $\text{deq}$ 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 $\text{deq}$, which consists in deciding whether $\text{deq}$ has integer solutions, has been proved to be undecidable [18]. We prove that given an arbitrary diophantine equation $\text{deq}$ we can reduce the 10th Hilbert Problem over $\text{deq}$ to the problem $P_{\text{deq}} \sim_l P_{\text{deq}}'$ for an opportune program $P_{\text{deq}}$. This is sufficient to prove that $\sim_l$ is undecidable.

Let $\text{deq} \equiv p(X_1, \ldots, X_n) = 0$ be a diophantine equation. Consider the program defined as

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

where $X_1, \ldots, X_n, L$ are low level variables and $H$ is a high level variable. $P_{\text{deq}}$ is a program, since $p(X_1, \ldots, X_n)$ is an arithmetic expression of our language. If $P_{\text{deq}} \sim_l P_{\text{deq}}'$, then $L := H$ is not in $\text{Reach}^* (P_{\text{deq}})$ 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., $\text{deq}$ does not admit integer solutions. On the other hand if $\text{deq}$ does not admit integer solutions, then there does not exist a state $\sigma$ such that $(p(X_1, \ldots, X_n) = 0, \sigma) \rightarrow \text{true}$, hence $L := H$ is not in $\text{Reach}^* (P_{\text{deq}})$ and $P_{\text{deq}} \sim_l P_{\text{deq}}'$ is true. \qed

Proof of Lemma 13 Let $a_1, a_2 \in \text{Aexp}$ be two arithmetic expressions. Let $L_1, \ldots, L_n$ and $H_1, \ldots, H_m$ be the low and high level variables, respectively, occurring in $a_1 + a_2$. We use $a[X_1, \ldots, X_k]$ to denote the fact that $a$ could contain the variables $X_1, \ldots, X_k$. We use the notation $a[X_1, \ldots, X_{i-1}, X_i/Y_1, \ldots, X_k/Y_k]$ to refer to the expression obtained by replacing in $a$ the variables $X_1, \ldots, X_k$ with $Y_1, \ldots, Y_k$, respectively. We have that the validity of $a_1 \equiv_l a_2$ is equivalent to the validity of the following first order formula over the reals

$$\forall L_1, \ldots, L_n, H_1, \ldots, H_m, K_1, \ldots, K_m
(a_1[L_1, \ldots, L_n, H_1, \ldots, H_m] = a_2[L_1, \ldots, L_n, H_1/K_1, \ldots, H_m/K_m].$$

As a consequence of the decidability of the first order theory of real numbers (see [32]) we get the thesis in the case of arithmetic expressions.

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

$$\forall L_1, \ldots, L_n, H_1, \ldots, H_m, K_1, \ldots, K_m
(b_1[L_1, \ldots, L_n, H_1, \ldots, H_m] \leftrightarrow b_2[L_1, \ldots, L_n, H_1/K_1, \ldots, H_m/K_m].)$$
Hence, exploiting again the result in [32] we get the thesis also in the case of boolean expressions.

Proof of Lemma 14 The fact that the relation \( \approx_l \subseteq \mathbb{P} \times \mathbb{P} \) is decidable is an immediate consequence of Lemma 13 and of the fact that \( \approx_l \) is defined by structural induction on the syntax of the programs.

In order to prove that \( \approx_l \) entails the relation \( \sim_l \) consider the binary relation \( S \) defined as

\[
S = \{ (P, Q) \mid P \approx_l Q \} \cup \{ (P, Q) \mid P \sim_l Q \}.
\]

We prove that \( S \) is a strong low level bisimulation. We proceed by structural induction on \( P \). Let \( \sigma, \theta \) be two states such that \( \sigma =_l \theta \).

Let \( P \equiv \text{end}(\text{skip}) \). In this case we have \( Q \equiv \text{end}(\text{skip}) \) 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 \( a_0 \approx_l a_1 \).

Since \( a_0 \approx_l a_1 \) by Lemma 12 we have that \( \langle a_0, \sigma \rangle \to n \) and \( \langle a_1, \theta \rangle \to n \). Hence, it holds that \( \langle P, \sigma \rangle \to \langle \text{end}, \sigma[L/n] \rangle \) and \( \langle Q, \theta \rangle \to \langle \text{end}, \theta[L/n] \rangle \) 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 we have \( Q \equiv K := a_1 \) with \( K \in \mathbb{K} \).

Let \( \langle a_0, \sigma \rangle \to n \) and \( \langle a_1, \theta \rangle \to m \). It holds that \( \langle P, \sigma \rangle \to \langle \text{end}, \sigma[H/n] \rangle \) and \( \langle Q, \theta \rangle \to \langle \text{end}, \theta[K/m] \rangle \) 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 \( P_0 \approx_l Q_0 \) and \( P_1 \approx_l Q_1 \). If \( \langle P_0, \sigma \rangle \to \langle \text{end}, \sigma' \rangle \), then \( \langle P, \sigma \rangle \to \langle P_1, \sigma' \rangle \). By inductive hypothesis on \( P_0 \) and \( Q_0 \) we have that \( \langle Q_0, \sigma \rangle \to \langle \text{end}, \theta' \rangle \) with \( \sigma' =_l \theta' \). Hence \( \langle Q, \theta \rangle \to \langle Q_0; Q_1, \theta' \rangle \) and \( \langle P, \sigma \rangle \to \langle P_0; P_1, \sigma' \rangle \). By inductive hypothesis on \( P_0 \) and \( Q_0 \) we have that \( \langle Q_0, \sigma \rangle \to \langle Q_0; Q_1, \theta' \rangle \) with \( \langle P_0, Q_0 \rangle \in S \) and \( \sigma' =_l \theta' \). Hence \( \langle Q, \theta \rangle \to \langle Q_0; Q_1, \theta' \rangle \) and \( \langle P_0; P_1, Q_0; Q_1 \rangle \in S \), i.e., the thesis.

Let \( P \equiv P_0 \mid P_1 \). This case is similar to the previous one.

Let \( P \equiv \text{if } b_0 \text{ then } P_0 \text{ else } P_1 \). Let \( Q \equiv \text{if } b_1 \text{ then } Q_0 \text{ else } Q_1 \) with \( P_0 \approx_l Q_0, P_1 \approx_l Q_1 \), and \( b_0 \approx_l b_1 \). Since \( b_0 \approx_l b_1 \) by Lemma 12 we have that \( \langle b_0, \sigma \rangle \to v \) if \( \langle b_1, \theta \rangle \to v \). Let us assume that \( \langle b_0, \sigma \rangle \to v \) 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 \( \langle P_0, Q_0 \rangle \in S \), i.e., the thesis. The case \( \langle b_0, \sigma \rangle \to v \) false is similar. Let \( Q \equiv \text{skip}; P_0 \) with \( b_0 \approx_l \text{true} \) and \( P_0 \approx_l P_0 \).

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 \( P \equiv \text{while } b_0 \text{ do } P_0 \). Let \( Q \equiv \text{while } b_1 \text{ do } Q_0 \) with \( b_0 \approx_l b_1 \) and \( P_0 \approx_l Q_0 \). If \( \langle P, \sigma \rangle \to \langle \text{end}, \sigma \rangle \), then \( \langle b_0, \sigma \rangle \to \text{false} \). Hence by Lemma 12 \( \langle b_1, \theta \rangle \to \text{false} \) and \( \langle Q, \sigma \rangle \to \langle \text{end}, \theta \rangle \), 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 \( \langle Q, \sigma \rangle \to \langle Q_0; Q, \theta \rangle \), with \( P_0 \approx_l Q_0; Q \), i.e., with \( \langle P_0; P, Q_0; Q \rangle \in S \).

The remaining cases are similar.

Proof of Theorem 6 In the worst case at each step two rules which require a checking of the form \( b_0 \approx_l b_1 \) are applicable. Moreover, at each step we apply only the first matching rule and \( l(P) \) decreases of at least 1.
Proof of Lemma 15 The fact that $\mathcal{W}(\simeq_{l}) \subseteq \mathcal{P}$ is decidable is an immediate consequence of Lemma 13, Lemma 14, and of the fact that $\mathcal{W}(\simeq_{l})$ is defined by structural induction on the syntax of the programs.

It is easy to prove that if $a, b \in 1_{\mathcal{W}}$, then $a \simeq_{l} a$ and $b \simeq_{l} b$. Hence, if $P \in \mathcal{W}(\simeq_{l})$, then it holds $P \simeq_{l} P$, since in Figure 4 we only instantiate some of the rules of $\simeq_{l}$. On the other hand, if $P \simeq_{l} P$, then the only rules which can have been applied are that which occur in Figure 4, hence $P \in \mathcal{W}(\simeq_{l})$. □

Proof of Lemma 16 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 now need to prove two claims.

Claim. Let $P$ be a program and $D$ be a declassifiable set. If $P \in \mathsf{SIMP}^{*}_{D}$, then for all $P' \in \mathsf{Reach}^{*}(P), P' \in \mathsf{SIMP}_{D}^{*}$.

Proof. This proof is similar to that of Lemma 9. □

Claim. For each $\psi$ and $\pi$ such that $\psi =_{L,D} \pi$, if $(F, \psi) \xrightarrow{1_{\mathcal{W}}} (F', \psi')$, then $(F, \pi) \xrightarrow{1_{\mathcal{W}}} (F', \pi')$ with $\pi' =_{L,D} \psi'$.

Proof. This is a consequence of the fact that in $D$ there are no low level variables. The proof follows by structural induction on programs. □

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

Proof. If $F \in \mathsf{Reach}^{*}(P)$, 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 $(P_{i-1}, \sigma_{i-1}) \rightarrow (P_{i}, \theta_{i})$. 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.

Step. We proved the thesis for $n = m$ and we consider $n = m + 1$. We have that $(P_{m}, \sigma_{m}) \rightarrow (P_{m+1}, \theta_{m+1})$. Since by inductive hypothesis it holds $P_{m} \sim_{l, D} P_{m}$ and it holds that $\sigma_{m} =_{L,D} \sigma_{m}$ and $(P_{m}, \sigma_{m}) \sim_{l, D} (P_{m+1}, \theta_{m+1})$ we get that $(P_{m}, \sigma_{m}) \sim_{l, D} (Q, \mu)$ 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 $P_{m+1} \sim_{l, D} P_{m+1}$. □

Proof of Theorem 8 It follows from the fact that by Definition 12, for all states $\sigma$ and $\theta, \sigma =_{L,D} \theta$ if and only if $\sigma =_{L, \gamma(D)} \theta$.

Proof of Theorem 9 \(\Rightarrow\) Consider the binary relation

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

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

If $(P, \sigma) \xrightarrow{\text{high}} (P', \sigma')$, then since $P \in \mathsf{SIMP}_{D}^{*}$, $(P, \sigma) \in \mathcal{W}(=_{L,D}, \sim_{l, D}, \mathsf{Reach}^{*})$ and then $(P, \theta) \rightarrow (P'', \theta')$ with $(P', \sigma') \sim_{l, D} (P'', \theta')$, i.e., $\sigma' =_{L,D} \theta'$ and $P' \sim_{l, D} P''$. Hence, by definition of $S$, $(P', P'') \in S$.  

31
If \( \langle P, \sigma \rangle \xrightarrow{\text{low}} \langle P', \sigma' \rangle \), then by Claim 6 we have that \( \langle P, \theta \rangle \xrightarrow{\text{low}} \langle P', \theta' \rangle \) with \( \sigma' =_{L,D} \theta' \). By Claim 6, we have that \( P' \in \text{SIMP}_D \) and then, by definition of \( S, \langle P', P' \rangle \in S \), i.e., the thesis.

\( \Leftarrow \) Let \( P \) be a program such that \( P \sim_{L,D} \). Let \( \sigma, \psi \in \Sigma \) and \( \langle F, \psi \rangle \in \text{Reach}^∗(\langle P, \sigma \rangle) \). Then \( F \in \text{Reach}^∗(P) \) and, by Claim 6, \( F \sim_{L,D} F \). Let \( \pi \) be such that \( \psi =_{L,D} \pi \).

If \( \langle F, \psi \rangle \xrightarrow{\text{high}} \langle G, \varphi \rangle \), 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.

**Claim.** Let \( P \) and \( Q \) be two programs and \( \sigma \) and \( \theta \) be two states. Let \( \langle P, \sigma \rangle \sim_{L,D} \langle Q, \theta \rangle \). If \( \langle P, \sigma \rangle \rightarrow^n \langle P', \sigma' \rangle \), then there exists \( Q' \) and \( \theta' \) such that \( \langle Q, \theta \rangle \rightarrow^n \langle Q', \theta' \rangle \) and \( \langle P', \sigma' \rangle \sim_{L,D} \langle Q', \theta' \rangle \), and viceversa.

**Proof.** By induction on \( n \).

- Base: \( n = 1 \). We immediately have the thesis by definition of \( \sim_{L,D} \).
- 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 \( \langle P'', \sigma'' \rangle \sim_{L,D} \langle Q'', \theta'' \rangle \). By definition of strong low-D level bisimulation we get the thesis.

**Proof of Theorem 10** By Theorem 9, since \( \sigma =_{L,D} \theta \), we have that \( \langle P, \sigma \rangle \sim_{L,D} \langle P, \theta \rangle \). Then, by Claim 6, we get that \( \langle P, \theta \rangle \) reaches a pair \( \langle P', \theta' \rangle \) with \( \langle P', \theta' \rangle \sim_{L,D} \langle \text{end}, \sigma' \rangle \). Hence we immediately have \( \sigma' =_{L,D} \theta' \). So by definition of \( =_{L,D} \) we get \( \sigma' = \theta' \). Moreover, since \( \text{end} \) is not strong low-D level bisimilar to any program, it must be \( P' \equiv \text{end} \).

**Proof of Lemma 17** Let \( a_1, a_2 \in \text{Aexp} \) be two arithmetic expressions. Let \( L_1, \ldots, L_n \) and \( H_1, \ldots, H_m \) 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 \( a_1 \equiv_{L,D} a_2 \) is equivalent to the validity of the following first order formula over the reals

\[
\forall L_1, \ldots, L_n, H_1, \ldots, H_m, K_1, \ldots, K_m
\langle \mathcal{A}_{d \in D} [d[H_1, \ldots, H_m] = [d[H_1/K_1, \ldots, H_m/K_m]] \rightarrow
\langle a_1[L_1, \ldots, L_n, H_1, \ldots, H_m] = [a_2[L_1, \ldots, L_n, H_1/K_1, \ldots, H_m/K_m]].
\]

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.

**Claim.** Let \( D \) be a declassifiable set, \( a_1, a_2 \in \text{Aexp} \) and \( b_1, b_2 \in \text{Bexp} \) such that \( a_1 \equiv_{L,D} a_2 \) and \( b_1 \equiv_{L,D} b_2 \). If \( \sigma \) and \( \theta \) are two states (over the integers) such that \( \sigma =_{L,D} \theta \), then: (1) \( \langle a_1, \sigma \rangle \rightarrow n \) if and only if \( \langle a_2, \theta \rangle \rightarrow n \), and (2) \( \langle b_1, \sigma \rangle \rightarrow \mathcal{V} \) if and only if \( \langle b_2, \theta \rangle \rightarrow \mathcal{V} \).

32
Proof of Lemma 18  It is sufficient to consider the case in which in $D$ there are only arithmetic expressions. Let $L_1, \ldots, L_n$ be the low level variables occurring in $a_0 + a_1$ and $H_1, \ldots, H_m, H, K$ be the high level variables occurring in $a_0, a_1$ and $D$. The validity of $H := a_0 \equiv_{l,D} K := a_1$ is equivalent to the validity of the following first order formula over the reals

$$\forall L_1, \ldots, L_n, H_1, \ldots, H_m, H, K, T_1, \ldots, T_m, T, R$$

$$(\bigwedge_{d \in D} d[H_1, \ldots, H_m, H, K] =$$

$$d[H_1/T_1, \ldots, H_m/T_m, H/T, K/R]) \rightarrow$$

$$(\bigwedge_{d \in D} d[H_1, \ldots, H_m, H/\hat{a}_0, K] =$$

$$d[H_1/T_1, \ldots, H_m/T_m, H/T, K/\hat{a}_1]).$$

where $\hat{a}_0$ is $a_0[L_1, \ldots, L_n, H_1, \ldots, H_m, H, K]$ and $\hat{a}_1$ is $a_1[L_1, \ldots, L_n, H_1/T_1, \ldots, H_m/T_m, H/T, K/R]$.

The fact that $H := a_0 \equiv_{l,D} K := a_1$ entails $H := a_0 \sim_{l,D} K := a_1$ is a consequence of Claim 6. \hfill $\square$

Proof of Lemma 19  The decidability is an immediate consequence of Lemma 17, 18 and of the fact that $\equiv_{l,D}$ is defined by structural induction on the syntax of the programs.

The second part of the proof is similar to that of Lemma 14. The only case which is different is the case $H := a_0 \equiv_{l,D} K := a_1$ which now is a consequence of Lemma 18. \hfill $\square$

Proof of Theorem 11  This is similar to the proof of Lemma 15 \hfill $\square$