Unwinding Conditions for Security in Imperative Languages*

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

[1] Dipartimento di Informatica, Università Ca' Foscari di Venezia
via Torino 155, 30172 Venezia, Italy
[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 study unwinding conditions for the definition of non-interference properties of a simple imperative language, admitting parallel executions on a shared memory. We present different classes of programs obtained by instantiating a general unwinding framework and show that all the programs in these classes satisfy the non-interference principle. Moreover, we introduce a subclass of secure programs which is compositional with respect to the language constructors and we discuss verification techniques.

1 Introduction

The problem of ensuring that a given program respects the security level of its variables has been deeply investigated for a variety of programming languages and by many authors. We refer the reader to [15] for a clear and wide overview of the various proposals. All of these proposals accomplish the non-interference principle introduced by Goguen and Meseguer [8] which asserts that secret input data cannot be inferred through the observation of non confidential outputs. Beside the approaches based on formal methods for controlling information flow we find formalizations of non-interference in terms of behavioural equivalences, e.g., [6, 14], type-systems, e.g., [15, 17, 18], and logical formulations, e.g., [2, 3].

In the context of process algebra, security properties are often expressed in terms of *unwinding conditions* [9] which demand properties of individual actions and are easier to handle with respect to global conditions. Intuitively, an unwinding condition requires that each high level transition is simulated in such a way that a low level observer cannot infer whether a 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.

* This work has been partially supported by the EU Contract IST-2001-32617 "Models and Types for Security in Mobile Distributed Systems" (MyThS) and the FIRB project RBAU018RCZ "Interpretazione astratta e model checking per la verifica di sistemi embedded".

In our previous works (see [4] for an overview) we studied many information flow security properties for the *Security Process Algebra* (SPA) [6] and characterized them in terms of unwinding conditions. In particular, we introduced a generalized unwinding condition which can be instantiated to define security properties and we identified classes of secure processes which can be constructed in a compositional way.

In this paper we show how our framework can be used also to define non-interference security properties for a simple imperative language, admitting parallel executions on a shared memory. We extend the language **IMP** defined in [19] by partitioning the locations (variables) into two levels: a public level and a confidential one and by adding a parallel composition operator.

We present a generalized unwinding condition for our language and study three different classes of programs obtained by instantiating the unwinding framework. These three instances are based on a notion of low level bisimulation and allow us to express timing-sensitive security properties for imperative languages. In particular, we show that all the programs in these classes satisfy the non-interference principle. Moreover, we introduce a subclass of secure programs which is compositional with respect to the language constructors. This class is useful to define proof systems which allow one both to verify and to build programs which are secure by construction.

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 three different instantiations of it. We also prove a soundness theorem with respect to the standard non-interference property. In Section 4 we define a compositional class of secure programs and discuss its verification. Finally, in Section 5 we draw some conclusions.

2 The Language: Syntax and Semantics

The language we consider is an extension of the **IMP** language defined in [19] 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. We present an operational semantics and a notion of behavioral equivalence for our language which will be at the basis of our security properties. The aim of our properties is to detect 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. In our operational semantics programs are associated to *labelled transition systems*, i.e., graphs with labels on the edges and on the nodes. The labels on the nodes correspond to the states of the locations and are used in the definition of the behavioral equivalence. The labels on the edges denote the level (high or low) of transitions, i.e., they individuate the transitions which depend on the values of high level locations.

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

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

where $r \in \mathbb{R}$ and $X \in \mathbb{L} \cup \mathbb{H}$. The set **Bexp** of boolean expressions is defined by:

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

where $a_0, a_1 \in \mathbf{Aexp}$.

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

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

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

where $a \in \mathbf{Aexp}$, $X \in \mathbb{L} \cup \mathbb{H}$, and $b \in \mathbf{Bexp}$.

A *high program* is a program which only uses high level locations (i.e., it syntactically contains only high level variables). We denote by $\mathbf{Prog_H}$ the set of all high programs.

Example 1. Consider the program $P \equiv L := H$, where H is a high level location and L is a low level location. P consists of a unique assignment instruction. Its effect is to assign to the low level location L the value contained in the high level location H. Hence, after the execution of P the low level user can read the high level data contained in H by reading L. □

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

Given an arithmetic expression $a \in \mathbf{Aexp}$ and a state σ, the evaluation of a in σ, denoted by $\langle a, \sigma \rangle \to r$ with $r \in \mathbb{R}$, is defined as in [19]. Similarly, $\langle b, \sigma \rangle \to v$ with $b \in \mathbf{Bexp}$ and $v \in \{\texttt{true}, \texttt{false}\}$, denotes the evaluation of a boolean expression b in a state σ and is defined as in [19].

The operational semantics of our programs is expressed in terms of state transitions. A transition from a program P and a state σ has the form $\langle P, \sigma \rangle \xrightarrow{\epsilon} \langle P', \sigma' \rangle$ where P' is either a program or `end` (termination) and $\epsilon \in \{\texttt{high}, \texttt{low}\}$ stating that the transition is either confidential or public. Let $\mathbb{P} = \mathbf{Prog} \cup \{\texttt{end}\}$

$$\frac{}{\langle \texttt{skip}, \sigma\rangle \overset{\texttt{low}}{\to} \langle \texttt{end}, \sigma\rangle} \qquad \frac{\langle a, \sigma\rangle \to r}{\langle X := a, \sigma\rangle \overset{\epsilon}{\to} \langle \texttt{end}, \sigma[X/r]\rangle}\ a \in \epsilon$$

$$\frac{\langle P_0, \sigma\rangle \overset{\epsilon}{\to} \langle P_0', \sigma'\rangle}{\langle P_0; P_1, \sigma\rangle \overset{\epsilon}{\to} \langle P_0'; P_1, \sigma'\rangle}\ P_0' \not\equiv \texttt{end} \qquad \frac{\langle P_0, \sigma\rangle \overset{\epsilon}{\to} \langle \texttt{end}, \sigma'\rangle}{\langle P_0; P_1, \sigma\rangle \overset{\epsilon}{\to} \langle P_1, \sigma'\rangle}$$

$$\frac{\langle b, \sigma\rangle \to \texttt{true}}{\langle \texttt{if}\ b\ \texttt{then}\ P_0\ \texttt{else}\ P_1, \sigma\rangle \overset{\epsilon}{\to} \langle P_0, \sigma\rangle}\ b \in \epsilon$$

$$\frac{\langle b, \sigma\rangle \to \texttt{false}}{\langle \texttt{if}\ b\ \texttt{then}\ P_0\ \texttt{else}\ P_1, \sigma\rangle \overset{\epsilon}{\to} \langle P_1, \sigma\rangle}\ b \in \epsilon$$

$$\frac{\langle b, \sigma\rangle \to \texttt{false}}{\langle \texttt{while}\ b\ \texttt{do}\ P, \sigma\rangle \overset{\epsilon}{\to} \langle \texttt{end}, \sigma\rangle}\ b \in \epsilon$$

$$\frac{\langle b, \sigma\rangle \to \texttt{true}}{\langle \texttt{while}\ b\ \texttt{do}\ P, \sigma\rangle \overset{\epsilon}{\to} \langle P; \texttt{while}\ b\ \texttt{do}\ P, \sigma\rangle}\ b \in \epsilon$$

$$\frac{\langle P_0, \sigma\rangle \overset{\epsilon}{\to} \langle P_0', \sigma'\rangle}{\langle P_0 \| P_1, \sigma\rangle \overset{\epsilon}{\to} \langle P_0' \| P_1, \sigma'\rangle}\ P_0' \not\equiv \texttt{end} \qquad \frac{\langle P_0, \sigma\rangle \overset{\epsilon}{\to} \langle \texttt{end}, \sigma'\rangle}{\langle P_0 \| P_1, \sigma\rangle \overset{\epsilon}{\to} \langle P_1, \sigma'\rangle}$$

$$\frac{\langle P_1, \sigma\rangle \overset{\epsilon}{\to} \langle P_1', \sigma'\rangle}{\langle P_0 \| P_1, \sigma\rangle \overset{\epsilon}{\to} \langle P_0 \| P_1', \sigma'\rangle}\ P_1' \not\equiv \texttt{end} \qquad \frac{\langle P_1, \sigma\rangle \overset{\epsilon}{\to} \langle \texttt{end}, \sigma'\rangle}{\langle P_0 \| P_1, \sigma\rangle \overset{\epsilon}{\to} \langle P_0, \sigma'\rangle}$$

Fig. 1. The operational semantics.

and Σ be the set of all the possible states. In Fig. 1 we define the operational semantics of $\langle P, \sigma\rangle \in \mathbb{P} \times \Sigma$ by structural induction on P.

For each pair $\langle P, \sigma\rangle$, where P is a program and σ is a state, the semantic rules define a *labelled transition system* (LTS) whose nodes are elements of $\mathbb{P} \times \Sigma$ and whose edges are labelled with `high` or `low`. The notion of reachability does not depend on the labels of the edges. We use $\langle P, \sigma\rangle \to \langle P', \sigma'\rangle$ to denote $\langle P, \sigma\rangle \overset{\epsilon}{\to} \langle P', \sigma'\rangle$ with $\epsilon \in \{\texttt{low}, \texttt{high}\}$. We write $\langle P_0, \sigma_0\rangle \to^n \langle P_n, \sigma_n\rangle$ for $\langle P_0, \sigma_0\rangle \to \langle P_1, \sigma_1\rangle \to \cdots \to \langle P_{n-1}, \sigma_{n-1}\rangle \to \langle P_n, \sigma_n\rangle$. Given $\langle P, \sigma\rangle \in \mathbf{Prog} \times \Sigma$, we denote by $\mathit{Reach}(\langle P, \sigma\rangle)$ 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 denote by $Reach(P)$ the set of programs P' such that $\langle P', \sigma'\rangle \in Reach(\langle P, \sigma\rangle)$ for some states σ and σ'.

Example 2. Assume L is a low level location and σ is a state such that $\sigma(L) = 1$. Consider the program $P_1 \equiv$ `while` $(L \leq 1)$ `do` $L := L + 1$; we obtain the following LTS

$$\begin{array}{c} \langle P_1, \sigma\rangle \\ \downarrow_{\texttt{low}} \\ \langle L := L + 1;\ \texttt{while}\ (L \leq 1)\ \texttt{do}\ L := L + 1, \sigma\rangle \\ \downarrow_{\texttt{low}} \\ \langle \texttt{while}\ (L \leq 1)\ \texttt{do}\ L := L + 1, \sigma[L/2]\rangle \\ \downarrow_{\texttt{low}} \\ \langle \texttt{end}, \sigma[L/2]\rangle \end{array}$$

Consider now the program $P_2 \equiv$ `if` $(H \leq 3)$ `then` $L := L+1$ `else` $L := L+2$ where H is a high level location. Let σ_1, σ_2 be states such that $\sigma_1(H) \leq 3$ and $\sigma_2(H) > 3$. The LTS's associated to the pairs $\langle P_2, \sigma_1\rangle$ and $\langle P_2, \sigma_2\rangle$ are

$$\begin{array}{c} \langle P_2, \sigma_1\rangle \\ \downarrow_{\texttt{high}} \\ \langle L := L + 1, \sigma_1\rangle \\ \downarrow_{\texttt{low}} \\ \langle \texttt{end}, \sigma_1[L/\sigma_1(L) + 1]\rangle \end{array} \qquad \begin{array}{c} \langle P_2, \sigma_2\rangle \\ \downarrow_{\texttt{high}} \\ \langle L := L + 2, \sigma_2\rangle \\ \downarrow_{\texttt{low}} \\ \langle \texttt{end}, \sigma_2[L/\sigma_2(L) + 2]\rangle \end{array}$$

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

We are interested in a notion of behavioural equivalence which equates two programs if they are indistinguishable for a low level observer.

Example 3. Consider the programs $H := 1; L := 1$ and $H := 2; L := 1$, where H is a high level location while L is a low level location. Given a state σ the LTS's associated to the two programs are respectively

$$\begin{array}{c} \langle H := 1; L := 1, \sigma\rangle \\ \downarrow_{\texttt{low}} \\ \langle L := 1, \sigma[H/1]\rangle \\ \downarrow_{\texttt{low}} \\ \langle \texttt{end}, \sigma[H/1, L/1]\rangle \end{array} \qquad \begin{array}{c} \langle H := 2; L := 1, \sigma\rangle \\ \downarrow_{\texttt{low}} \\ \langle L := 1, \sigma[H/2]\rangle \\ \downarrow_{\texttt{low}} \\ \langle \texttt{end}, \sigma[H/2, L/1]\rangle \end{array}$$

We would like to consider this two programs equivalent for a low level observer which can only read the values in the low level locations. □

We consider two programs equivalent from the low level point of view if they are *low level bisimilar* as defined below.

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

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

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

A *partial equivalence relation* (*per*, for short) [16] is a symmetric and transitive relation.

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

Proof. If $\langle P, \sigma\rangle \sim_l \langle Q, \theta\rangle$, then there exists a low level bisimulation $\mathcal{B}$ such that it holds $(\langle P, \sigma\rangle, \langle Q, \theta\rangle) \in \mathcal{B}$. Hence if $\langle P, \sigma\rangle \to \langle P', \sigma'\rangle$ we have that $\langle Q, \theta\rangle \to \langle Q', \theta'\rangle$ with $(\langle P', \sigma'\rangle, \langle Q', \theta'\rangle) \in \mathcal{B}$, i.e., $\langle P', \sigma'\rangle \sim_l \langle Q', \theta'\rangle$. So we have that $\sim_l$ 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 $\sim_l$ is reflexive and symmetric. The fact that $\sim_l$ is transitive follows from the fact that if $\mathcal{B}_1, \mathcal{B}_2$ are low level bisimulations then the relation $\mathcal{B}_1 \circ \mathcal{B}_2$, where $\circ$ is the composition of relations, is still a low level bisimulation.

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

The relation $\simeq_l$ is not reflexive. For example, the program $L := H$ is not low level bisimilar to itself, as the low equality of states can be broken by a computation step.

Example 4. Consider the programs $P \equiv H := H + 1; L := L + 1$ and $Q \equiv H := H + 2; L := L + 1$, where H is a high level location and L is a low level location. 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. □

The notion of bisimulation as observation equivalence assumes that during each computation step a user can read the values in the locations. If we are working with a pure imperative language this assumption could seem too strong, since usually the values are read only at the end of the computation. However, if we consider parallel executions, during each step of the computation one of the parallel components could store the partial results of the other components.

Example 5. Let $P \equiv L := H; L := 1$ and $Q \equiv H := H; L := 1$, where H is a high level location and L is a low level location. The programs P and Q could be considered equivalent if one assumes that the low level user can observe the low

level locations only at the end of the computation. However, they are not low level bisimilar. Indeed, if $R \equiv L_1 := L$ with L_1 being a low level location, then the programs $P \| R$ and $Q \| R$ are not equivalent from the low level point view. In fact, there is one execution of $P \| R$ in which the low level user can discover the high level value of H by reading L_1. This is never possible in $Q \| R$. □

The relation $\sim_l$ equates programs which exhibit the same timing behavior. This is stated by the following lemma.

Lemma 2. *Let P and Q be two programs and σ and θ be two states such that $\langle P, \sigma \rangle \sim_l \langle Q, \theta \rangle$. If $\langle P, \sigma \rangle \rightarrow^n \langle P', \sigma' \rangle$ then there exists Q' and θ' such that $\langle Q, \theta \rangle \rightarrow^n \langle Q', \theta' \rangle$ and $\langle P', \sigma' \rangle \sim_l \langle Q', \theta' \rangle$, and viceversa.*

Proof. By induction on n.

- Base: $n = 1$. We immediately have the thesis by definition of $\sim_l$.
- 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 $\langle P'', \sigma'' \rangle \sim_l \langle Q'', \theta'' \rangle$. By definition of bisimulation we get the thesis. □

Example 6. Consider the programs $P \equiv \texttt{if}\ (L = 0)\ \texttt{then}\ L := L{+}1\ \texttt{else}\ L := 2$ and $Q \equiv \texttt{if}\ (L = 0)\ \texttt{then}\ \{L := L + 1; \texttt{skip}\}\ \texttt{else}\ L := 2$. Although, for all σ and θ such that $\sigma =_L \theta$, P and Q execute exactly the same assignment commands, $P \not\sim_l Q$. In fact the two programs exhibit different timing behaviours due to the presence of the `skip` command in the first branch of Q. □

3 Unwinding Conditions for Security of IMP

In [4] we introduced a general framework to define classes of secure processes written in the SPA language, an extension of Milner's CCS [12]. The framework is based on a generalized unwinding condition which is a local persistent property parametric with respect to a low behavioral equivalence, a transition relation independent from the high level behavior and a reachability relation. We proved that many non-interference properties can be seen as instances of this framework. In all the considered cases, the three relations are defined on the processes LTS's and thus the corresponding unwinding classes depend only on the operational semantics of processes. Following a similar approach, we introduce a generalized unwinding condition to define classes of programs which is parametric with respect to

- an observation equivalence relation $\doteqdot$ which equates two pairs $\langle P, \sigma \rangle$ and $\langle Q, \theta \rangle$ if they are indistinguishable for a low level observer,
- a binary relation $\hookrightarrow$ which, from the low level point of view, is independent from the values of high locations, and
- a reachability function $\mathcal{R}$ associating to each pair $\langle P, \sigma \rangle$ the set of pairs $\langle F, \psi \rangle$ which, in some sense, are reachable from $\langle P, \sigma \rangle$.

Definition 2 (Generalized Unwinding). *Let $\doteqdot$ be a binary equivalence relation over* $\mathbf{Prog} \times \Sigma$, $\hookrightarrow$ *be a binary relation over* $\mathbf{Prog} \times \Sigma$ *and* $\mathcal{R}$ *be a function from* $\mathbf{Prog} \times \Sigma$ *to* $\wp(\mathbf{Prog} \times \Sigma)$. *We define the* unwinding class $\mathcal{W}(\doteqdot, \hookrightarrow, \mathcal{R})$ *as follows:*

$$\mathcal{W}(\doteqdot, \hookrightarrow, \mathcal{R}) \stackrel{\text{def}}{=} \{\langle P, \sigma\rangle \in \mathbf{Prog} \times \Sigma \mid \forall\ \langle F, \psi\rangle \in \mathcal{R}(\langle P, \sigma\rangle)$$
$$\textit{if } \langle F, \psi\rangle \stackrel{\texttt{high}}{\rightarrow} \langle G, \varphi\rangle \textit{ then } \exists \langle M, \mu\rangle \textit{ such that } \langle F, \psi\rangle \hookrightarrow \langle M, \mu\rangle \textit{ and}$$
$$\langle G, \varphi\rangle \doteqdot \langle M, \mu\rangle\}.$$

The intuition behind the unwinding condition is that any high level transition should be simulated by a high independent transition guaranteeing that the high level transitions have no influence on the low level observation.

We say that the function $\mathcal{R}$ is *transitive* if $\langle F'', \psi''\rangle \in \mathcal{R}(\langle F', \psi'\rangle)$ and $\langle F', \psi'\rangle \in \mathcal{R}(\langle F, \psi\rangle)$ imply $\langle F'', \psi''\rangle \in \mathcal{R}(\langle F, \psi\rangle)$, i.e., it is a transitive relation. If $\mathcal{R}$ is transitive, the generalized unwinding condition defined above allows us to specify properties which are closed under $\mathcal{R}$. In this sense we say that our properties are *persistent*. The next lemma follows immediately by Definition 2.

Lemma 3. *Let $\mathcal{R}$ be a transitive reachability function and $\langle P, \sigma\rangle \in \mathbf{Prog} \times \Sigma$. If $\langle P, \sigma\rangle \in \mathcal{W}(\doteqdot, \hookrightarrow, \mathcal{R})$ then $\langle F, \psi\rangle \in \mathcal{W}(\doteqdot, \hookrightarrow, \mathcal{R})$ for all $\langle F, \psi\rangle \in \mathcal{R}(\langle P, \sigma\rangle)$.*

Proof. Let $\mathcal{R}$ be transitive, $\langle P, \sigma\rangle \in \mathcal{W}(\doteqdot, \hookrightarrow, \mathcal{R})$, and $\langle F, \psi\rangle \in \mathcal{R}(\langle P, \sigma\rangle)$. If $\langle F', \psi'\rangle \in \mathcal{R}(\langle F, \psi\rangle)$, then by transitivity we have that $\langle F', \psi'\rangle \in \mathcal{R}(\langle P, \sigma\rangle)$. Hence we get that if $\langle F', \psi'\rangle \stackrel{\texttt{high}}{\rightarrow} \langle G', \varphi'\rangle$ then $\langle F', \psi'\rangle \hookrightarrow \langle M', \mu'\rangle$ with $\langle G', \varphi'\rangle \doteqdot \langle M', \mu'\rangle$, i.e., the thesis. □

Below we instantiate our generalized unwinding condition by exploiting the notion of low level bisimulation $\sim_l$ as behavioral equivalence and by introducing a suitable high independent transition relation $\dashrightarrow$.

Definition 3 ($\dashrightarrow$). *The relation $\dashrightarrow$ on* $\mathbf{Prog} \times \Sigma$ *is defined as follows:* $\langle F, \psi\rangle \dashrightarrow \langle M, \mu\rangle$ *if for each π such that $\pi =_L \psi$ there exist R and ρ such that $\langle F, \pi\rangle \rightarrow \langle R, \rho\rangle$ and $\langle R, \rho\rangle \sim_l \langle M, \mu\rangle$.*

Example 7. Let $F \equiv \texttt{if}\ (H > 1)\ \texttt{then}\ M\ \texttt{else}\ R$ where $M \equiv H := 1; L := L + 1$ and $R \equiv H := 2; L := L + 1$, and ψ be such that $\psi(H) > 1$. In this case $\langle F, \psi\rangle \dashrightarrow \langle M, \psi\rangle$. Indeed, for each π such that $\pi =_L \psi$ either $\langle F, \pi\rangle \rightarrow \langle M, \pi\rangle$ or $\langle F, \pi\rangle \rightarrow \langle R, \pi\rangle$ and both $\langle M, \pi\rangle \sim_l \langle M, \psi\rangle$ and $\langle R, \pi\rangle \sim_l \langle M, \psi\rangle$.

Consider now the program $F \equiv L := 2; R$ and $R \equiv \texttt{if}\ (H > 1)\ \texttt{then}\ \{H := 1; L := 2\}\ \texttt{else}\ \{H := 2; L := 1\}$. In this case does not exist any $\langle M, \mu\rangle$ such that $\langle F, \psi\rangle \dashrightarrow \langle M, \mu\rangle$. Indeed, if ψ and π are two states such that $\psi =_L \pi$, $\psi(H) > 1$ and $\pi(H) \leq 1$, then $\langle F, \psi\rangle \rightarrow \langle R, \psi[L/2]\rangle$ and $\langle F, \pi\rangle \rightarrow \langle R, \pi[L/2]\rangle$ but $\langle R, \psi[L/2]\rangle \not\sim_l \langle R, \pi[L/2]\rangle$. □

By Definition 3 and by transitivity of $\sim_l$ we get the following characterization of our unwinding condition.

Proposition 1. *Let $\mathcal{R}$ be a reachability function, P be a program, and σ be a state. $\langle P,\sigma\rangle \in \mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R})$ if and only if for each $\langle F,\psi\rangle \in \mathcal{R}(\langle P,\sigma\rangle)$ it holds that if $\langle F,\psi\rangle \stackrel{\texttt{high}}{\rightarrow} \langle G,\varphi\rangle$ then for each π such that $\pi =_L \psi$ there exist R and ρ such that $\langle F,\pi\rangle \rightarrow \langle R,\rho\rangle$ and $\langle R,\rho\rangle \sim_l \langle G,\varphi\rangle$.*

As far as the function $\mathcal{R}$ is concerned, we consider three different instantiations: $\mathcal{R}_{lts}$ which coincides with the rechability relation *Reach* in the LTS, $\mathcal{R}_{hpar}$ which intuitively represents reachability under the parallel composition with any high level program, and $\mathcal{R}_{par}$ which denotes reachability under the parallel composition with any program.

The class of secure imperative programs $\mathbf{SIMP}_{lts}$ defined below is based on the function $\mathcal{R}_{lts}$.

Definition 4 ($\mathbf{SIMP}_{lts}$). *Let $\mathcal{R}_{lts}$ be the function Reach. A program P is in $\mathbf{SIMP}_{lts}$ if for each state σ, $\langle P,\sigma\rangle \in \mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{lts})$.*

Example 8. Consider the program $Q \equiv H := L$, where H is a high level location and L is a low level location. The program Q is in $\mathbf{SIMP}_{lts}$. In fact, the low level execution is not influenced by the values in the high level location.

Consider again the program $P \equiv L := H; L := 1$ of Example 5, where H is a high level location and L is a low level location. It is easy to prove that for any $\sigma \in \Sigma$, $\langle P,\sigma\rangle \notin \mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{lts})$. In fact, let for instance $\sigma(H) = 1$, $\sigma(L) = 0$, $\theta(H) = 2$, $\theta(L) = 0$. It holds that $\sigma =_L \theta$, but after the execution of the first high level transition we reach the states σ' and θ', where $\sigma'(L) = 1 \neq \theta'(L) = 2$.

Consider now $R \equiv H := 4; L := 1;$ `if` $(L = 1)$ `then skip else` $L := H$. The program R belongs to $\mathbf{SIMP}_{lts}$. In fact, the first branch of the conditional is always executed independently of the value in the high level location. □

Since $\mathcal{R}_{lts}$ is transitive, by Lemma 3 we get that $\mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{lts})$ is persistent, i.e., if a program P starting in a state σ 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 a program P is in $\mathbf{SIMP}_{lts}$ then also each program P' reachable from P is in $\mathbf{SIMP}_{lts}$. This is illustrated in the following example.

Example 9. Let $P \equiv L := 0;$ `if` $L := 1$ `then` $L := H$ `else skip`. It holds that $P \in \mathbf{SIMP}_{lts}$ since, for each state σ, $\langle P,\sigma\rangle$ will never perform any high transition. Moreover, the program $P' \equiv$ `if` $L := 1$ `then` $L := H$ `else skip` is reachable from P but it does not belong to $\mathbf{SIMP}_{lts}$. □

We now introduce a more restrictive class of secure imperative programs, namely $\mathbf{SIMP}_{hpar}$, which is based on the reachability function $\mathcal{R}_{hpar}$ defined below.

Definition 5. *The function $\mathcal{R}_{hpar}$ from $\mathbf{Prog}\times\Sigma$ to $\wp(\mathbf{Prog}\times\Sigma)$ is defined by: $\mathcal{R}_{hpar}(\langle P_0,\sigma_0\rangle) = \{\langle P_n,\theta_n\rangle \mid n \geq 0,\ \exists P_1,\ldots,P_{n-1},\ \exists \sigma_1,\ldots,\sigma_n, \exists\, \theta_0,\ldots,\theta_{n-1}$ such that $\sigma_i =_L \theta_i$ and $\langle P_i,\theta_i\rangle \rightarrow \langle P_{i+1},\sigma_{i+1}\rangle$ for $i \in [0\,..\,n-1]$ and $\sigma_n =_L \theta_n\}$.*

Intuitively, $\langle F,\psi\rangle \in \mathcal{R}_{hpar}(\langle P,\sigma\rangle)$ if $\langle F,\psi\rangle$ is reachable from $\langle P \| P_H,\sigma\rangle$ where P_H is a high level program.

Lemma 4. *Let P be a program and σ be a state. $\langle F, \psi\rangle \in \mathcal{R}_{hpar}(\langle P, \sigma\rangle)$ if and only if F is a subprogram of P and $\langle F, \psi\rangle \in Reach(\langle P \| P_H, \sigma\rangle)$ for some high program P_H.*

Proof. (sketch) $\Leftarrow$) The parallel composition of two programs performs the interleaving of the actions of the two components. Hence, when executing $\langle P \| P_H, \sigma\rangle$, since P_H can only modify high level variables, each time an action $\langle P_H^i, \sigma_i\rangle \rightarrow \langle P_H^{i+1}, \theta_i\rangle$ of P_H is performed, we have that $\theta_i =_L \sigma_i$. On the other hand, when an action $\langle P_i, \sigma_i\rangle \rightarrow \langle P_{i+1}, \sigma_{i+1}\rangle$ of P is performed then we can define $\theta_i = \sigma_i$. Hence, $\exists \sigma_1, \ldots, \sigma_n, \theta_0, \ldots, \theta_{n-1}$ such that $\sigma_i =_L \theta_i$ and $\langle P_i, \theta_i\rangle \rightarrow \langle P_{i+1}, \sigma_{i+1}\rangle$ for $i \in [0\,..\,n-1]$ where $\langle P_0, \sigma_0\rangle \equiv \langle P, \sigma\rangle$ and $\langle P_n, \theta_n\rangle \equiv \langle F, \psi\rangle$.

$\Rightarrow$) In each step of the computation P_H can only change the value of high level variables, hence we immediately get the thesis. □

Definition 6 ($\mathbf{SIMP}_{hpar}$). *A program P is in $\mathbf{SIMP}_{hpar}$ if for each state σ, $\langle P, \sigma\rangle \in \mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{hpar})$.*

It is clear that the class $\mathbf{SIMP}_{hpar}$ is more restrictive than $\mathbf{SIMP}_{lts}$.

Lemma 5. $\mathbf{SIMP}_{hpar} \subseteq \mathbf{SIMP}_{lts}$

Example 10. Consider the program $P \equiv H := 1; \texttt{if}\ (H = 1)\ \texttt{then}\ \{\texttt{skip}; L := 1\}\ \texttt{else}\ \{H := 1; L := H\}$, where H is a high level location and L is a low level location. The program P belongs to the class $\mathbf{SIMP}_{lts}$ but it does not belong to the class $\mathbf{SIMP}_{hpar}$. In fact, given an initial state σ there exists a state ψ such that the pair $\langle L := H, \psi\rangle$ belongs to $\mathcal{R}_{hpar}(\langle P, \sigma\rangle)$. Moreover $\langle L := H, \psi\rangle \stackrel{\texttt{high}}{\rightarrow} \langle \texttt{end}, \varphi\rangle$ but clearly it does not hold that for each π such that $\pi =_L \psi$ there exist R and ρ such that $\langle L := H, \pi\rangle \rightarrow \langle R, \rho\rangle$ and $\langle R, \rho\rangle \sim_l \langle \texttt{end}, \varphi\rangle$.

Notice that if we consider the program $Q \equiv H := 3$ then $P \| Q$ is not in $\mathbf{SIMP}_{lts}$ although both P and Q are in $\mathbf{SIMP}_{lts}$. □

It is easy to prove that the reachability function $\mathcal{R}_{hpar}$ is transitive. Hence by Lemma 3 the class $\mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{hpar})$ is persistent. Indeed, we have that if a program P starting in a state σ is in $\mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{hpar})$ then also each pair $\langle P', \sigma'\rangle \in \mathcal{R}_{hpar}(\langle P, \sigma\rangle)$ is in $\mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{hpar})$. However, as for $\mathbf{SIMP}_{lts}$, in general it does not hold that if a program P is in $\mathbf{SIMP}_{hpar}$ then also each program P' reachable from P is in $\mathbf{SIMP}_{hpar}$. In order to see this, it is sufficient to consider again the program of Example 9.

Finally, we introduce the class of secure imperative programs $\mathbf{SIMP}_{par}$ by using the reachability function $\mathcal{R}_{par}$ defined below.

Definition 7. *The function $\mathcal{R}_{par}$ from $\mathbf{Prog} \times \Sigma$ to $\wp(\mathbf{Prog} \times \Sigma)$ is defined as follows: $\mathcal{R}_{par}(\langle P_0, \sigma_0\rangle) = \{\langle P_n, \theta_n\rangle \mid n \geq 0,\ \theta_n \in \Sigma,\ \exists P_1, \ldots, P_{n-1},\ \exists \sigma_1, \ldots, \sigma_n,\ \exists \theta_0, \ldots, \theta_{n-1}$ such that $\langle P_i, \theta_i\rangle \rightarrow \langle P_{i+1}, \sigma_{i+1}\rangle$ for $i \in [0\,..\,n-1]\}$.*

Intuitively, a pair $\langle F, \psi\rangle$ is in $\mathcal{R}_{par}(\langle P, \sigma\rangle)$ if $\langle F, \psi\rangle$ is reachable from $\langle P \| Q, \sigma\rangle$ for some program Q. The following lemma is similar to Lemma 4.

Lemma 6. *Let P be a program and σ be a state. $\langle F, \psi\rangle \in \mathcal{R}_{par}(\langle P, \sigma\rangle)$ if and only if F is a subprogram of P and $\langle F, \psi\rangle \in Reach(\langle P \| Q, \sigma\rangle)$ for a program Q.*

Definition 8 ($\mathbf{SIMP}_{par}$). *A program P is in $\mathbf{SIMP}_{par}$ if for each state σ, $\langle P, \sigma\rangle \in \mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{par})$.*

The class $\mathbf{SIMP}_{par}$ is more restrictive than $\mathbf{SIMP}_{hpar}$.

Lemma 7. $\mathbf{SIMP}_{par} \subseteq \mathbf{SIMP}_{hpar} \subseteq \mathbf{SIMP}_{lts}$.

Example 11. Consider the program $P \equiv H := 4; L := 1; \texttt{if } (L = 1) \texttt{ then skip else } L := H$. It belongs to $\mathbf{SIMP}_{lts}$ and $\mathbf{SIMP}_{hpar}$ but it does not belong to $\mathbf{SIMP}_{par}$. In fact given an initial state σ there exists a state ψ such that the pair $\langle L := H, \psi\rangle$ belongs to $\mathcal{R}_{par}(\langle P, \sigma\rangle)$. Moreover $\langle L := H, \psi\rangle \overset{\texttt{high}}{\rightarrow} \langle \texttt{end}, \varphi\rangle$ but clearly it does not hold that for each π such that $\pi =_L \psi$ there exist R and ρ such that $\langle L := H, \pi\rangle \rightarrow \langle R, \rho\rangle$ and $\langle R, \rho\rangle \sim_l \langle \texttt{end}, \varphi\rangle$. □

The reachability function $\mathcal{R}_{par}$ is transitive and then, by Lemma 3, the class $\mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{par})$ is persistent in the sense that if $\langle P, \sigma\rangle$ is in $\mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{par})$ then also each pair $\langle P', \sigma'\rangle \in \mathcal{R}_{par}(\langle P, \sigma\rangle)$ is in $\mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{par})$. Moreover, differently from $\mathbf{SIMP}_{lts}$ and $\mathbf{SIMP}_{hpar}$, if a program P is in $\mathbf{SIMP}_{par}$ then also each program P' reachable from P is in $\mathbf{SIMP}_{par}$.

Lemma 8. *Let P be a program. If $P \in \mathbf{SIMP}_{par}$ then for all $P' \in Reach(P)$, $P' \in \mathbf{SIMP}_{par}$.*

Proof. Let $P' \in Reach(P)$, i.e., $\langle P', \sigma'\rangle \in Reach(\langle P, \sigma\rangle)$ for some σ and σ'. By definition of $\mathcal{R}_{par}$, $\langle P', \theta\rangle \in \mathcal{R}_{par}(\langle P, \sigma\rangle)$ for all state θ. Hence, by persistence of $\mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{par})$, $\langle P', \theta\rangle \in \mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{par})$, i.e., $P' \in \mathbf{SIMP}_{par}$. □

The three instances of our generalized unwinding condition introduced above allow us to express timing-sensitive notions of secuirity for imperative programs. This is a consequence of the fact that $\sim_l$ equates programs which exhibit the same timing behavior (see Lemma 2).

Example 12. Let $P \equiv \texttt{if } (H = 0) \texttt{ then } \{H := H + 1; \texttt{skip}\} \texttt{ else } H := 2$. The program P does not belong to any class $\mathbf{SIMP}_*$ with $* \in \{lts, hpar, par\}$. This is due to the fact that if $\langle P, \sigma\rangle \overset{\texttt{high}}{\rightarrow} \langle \{H := H+1; \texttt{skip}\}, \sigma\rangle$ for some state σ then it does not hold that for each θ such that $\sigma =_L \theta$ there exist R and θ' such that $\langle P, \theta\rangle \rightarrow \langle R, \theta'\rangle$ and $\langle \{H := H + 1; \texttt{skip}\}, \sigma\rangle \sim_l \langle R, \theta'\rangle$. In fact, if $\theta(H) \neq 0$, $\langle P, \theta\rangle \rightarrow \langle H := 2, \theta\rangle$ but $\langle \{H := H+1; \texttt{skip}\}, \sigma\rangle \not\sim_l \langle H := 2, \theta\rangle$ because of their different timing behaviour. □

In the previous section we observed that the relation $\simeq_l$ is not reflexive. However, $\simeq_l$ is reflexive over the set of programs belonging to $\mathbf{SIMP}_{lts}$ (and then, by Lemma 7, to $\mathbf{SIMP}_{hpar}$ and $\mathbf{SIMP}_{par}$).

Lemma 9. *Let P be a program. If $P \in \mathbf{SIMP}_{lts}$ then $P \simeq_l P$.*

Proof. First, the following claim follows by structural induction on programs.

Claim. For each ψ and π such that $\psi_L = \pi_L$, if $\langle F, \psi\rangle \stackrel{\texttt{low}}{\rightarrow} \langle F', \psi'\rangle$ then $\langle F, \pi\rangle \stackrel{\texttt{low}}{\rightarrow} \langle F', \pi'\rangle$ with $\pi'_L = \psi'_L$.

Now assume that $P \in \mathbf{SIMP}_{lts}$. Then for all states σ and θ, $\langle P, \sigma\rangle, \langle P, \theta\rangle \in \mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{lts})$. Hence, in order to prove that $P \simeq_l P$, it is sufficient to show that for all σ and θ such that $\langle P, \sigma\rangle, \langle P, \theta\rangle \in \mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{lts})$ and $\sigma_L = \theta_L$, it holds $\langle P, \sigma\rangle \sim_l \langle P, \theta\rangle$. Consider the binary relation

$$\mathcal{S} = \{(\langle P, \sigma\rangle, \langle P, \theta\rangle) \mid \langle P, \sigma\rangle, \langle P, \theta\rangle \in \mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{lts}),\ \sigma_L = \theta_L\}$$

$$\cup\ \{(\langle P, \sigma\rangle, \langle Q, \theta\rangle) |\ \langle P, \sigma\rangle \sim_l \langle Q, \theta\rangle\}.$$

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

If $\langle P, \sigma\rangle \stackrel{\texttt{high}}{\rightarrow} \langle P', \sigma'\rangle$, then since $\langle P, \sigma\rangle \in \mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{lts})$, by Proposition 1, we have that $\langle P, \theta\rangle \rightarrow \langle P'', \theta'\rangle$ with $\langle P', \sigma'\rangle \sim_l \langle P'', \theta'\rangle$. Hence, by definition of $\mathcal{S}$, $(\langle P', \sigma'\rangle, \langle P'', \theta'\rangle) \in \mathcal{S}$.

If $\langle P, \sigma\rangle \stackrel{\texttt{low}}{\rightarrow} \langle P', \sigma'\rangle$, then by Claim 3 we have that $\langle P, \theta\rangle \stackrel{\texttt{low}}{\rightarrow} \langle P', \theta'\rangle$ with $\sigma'_L = \theta'_L$. By Lemma 3, since $\mathcal{R}_{lts}$ is transitive, we have that $\mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{lts})$ is persistent, i.e., both $\langle P', \sigma'\rangle \in \mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{lts})$ and $\langle P', \theta'\rangle \in \mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{lts})$. Hence we have that $(\langle P', \sigma'\rangle, \langle P', \theta'\rangle) \in \mathcal{S}$, i.e., the thesis. □

The converse of Lemma 9 does not hold as illustrated below.

Example 13. Consider the program $P \equiv \texttt{if}\ (H = 1)\ \texttt{then}\ P_0\ \texttt{else}\ P_1$ where $P_0 \equiv \texttt{while}\ (H > 1)\ \texttt{do skip}$ and $P_1 \equiv \texttt{skip}$. In this case $P \simeq_l P$, i.e., for all states σ and θ such that $\sigma =_L \theta$, $\langle P, \sigma\rangle \sim_l \langle P, \theta\rangle$. Indeed, if σ and θ are such that both $\sigma(H) = 1$ and $\theta(H) = 1$, the LTS's of $\langle P, \sigma\rangle$ and $\langle P, \theta\rangle$ have the form

$$\begin{array}{cc} \langle P, \sigma\rangle & \langle P, \theta\rangle \\ \downarrow & \downarrow \\ \langle P_0, \sigma\rangle & \langle P_0, \theta\rangle \\ \downarrow & \downarrow \\ \langle \texttt{end}, \sigma\rangle & \langle \texttt{end}, \theta\rangle \end{array}$$

and thus $\langle P, \sigma\rangle \sim_l \langle P, \theta\rangle$. The case in which both $\sigma(H) \neq 1$ and $\theta(H) \neq 1$ is analogous. On the other hand, if $\sigma(H) = 1$ and $\theta(H) \neq 1$ the LTS's of $\langle P, \sigma\rangle$ and $\langle P, \theta\rangle$ have the form

$$\begin{array}{cc} \langle P, \sigma\rangle & \langle P, \theta\rangle \\ \downarrow & \downarrow \\ \langle P_0, \sigma\rangle & \langle P_1, \theta\rangle \\ \downarrow & \downarrow \\ \langle \texttt{end}, \sigma\rangle & \langle \texttt{end}, \theta\rangle \end{array}$$

and again $\langle P, \sigma\rangle \sim_l \langle P, \theta\rangle$.

However, the program $P \notin \mathbf{SIMP}_{lts}$. In fact $\langle P_0, \sigma\rangle \in \mathit{Reach}(\langle P, \sigma\rangle)$ and $\langle P_0, \sigma\rangle \stackrel{\texttt{high}}{\rightarrow} \langle \texttt{end}, \sigma\rangle$ but it does not hold that for all ρ such that $\sigma =_L \rho$ there

exist R and ρ' such that $\langle P_0, \rho\rangle \to \langle R, \rho'\rangle$ and $\langle \texttt{end}, \sigma\rangle \sim_l \langle R, \rho'\rangle$. Indeed, if $\rho(H) > 1$, $\langle P_0, \rho\rangle \to \langle \texttt{skip}; P_0, \rho\rangle$ and $\langle \texttt{end}, \sigma\rangle \not\sim_l \langle \texttt{skip}; P_0, \rho\rangle$. This is due to the fact that the subprogram P_0 of P is not in $\mathbf{SIMP}_{lts}$. □

Finally, we show that our security properties expressed in terms of unwinding conditions imply the standard non-interference principle which requires that high level values do not affect the low level observation.

Theorem 1 (Soundness). *Let P be a program such that $P \in \mathbf{SIMP}_*$ with $* \in \{lts, hpar, par\}$. For each state σ and θ such that $\sigma =_L \theta$,*

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

Proof. By Lemma 9, since $\sigma =_L \theta$, we have that $\langle P, \sigma\rangle \sim_l \langle P, \theta\rangle$. Then, by Lemma 2, we get that $\langle P, \theta\rangle$ reaches a pair $\langle P', \theta'\rangle$ with $\langle P', \theta'\rangle \sim_l \langle \texttt{end}, \sigma'\rangle$. Hence we immediately have $\sigma' =_L \theta'$. Moreover, since `end` is not bisimilar to any program, it must be $P' \equiv \texttt{end}$. □

4 Compositionality

The classes $\mathbf{SIMP}_{lts}$, $\mathbf{SIMP}_{hpar}$ and $\mathbf{SIMP}_{par}$ introduced above are, in general, not compositional with respect to the language constructors. In particular, they are not compositional with respect to the parallel composition constructor as illustrated by the following example.

Example 14. Consider the program $P \equiv \texttt{if } (H = 1 \wedge L = 1) \texttt{ then } P_0 \texttt{ else } P_1$ where $P_0 \equiv \texttt{if } (L = 1) \texttt{ then skip else } L := 2$ while $P_1 \equiv \texttt{if } (L \neq 1) \texttt{ then } L := 3 \texttt{ else skip}$. The program P belongs to the class $\mathbf{SIMP}_{par}$ (and then also to the classes $\mathbf{SIMP}_{lts}$ and $\mathbf{SIMP}_{hpar}$). In fact, given an initial state σ, $\langle P, \sigma\rangle \stackrel{\texttt{high}}{\to} \langle P_i, \sigma\rangle$ for some $i \in \{0, 1\}$ and for each π such that $\pi =_L \sigma$ there always exist R and ρ such that $\langle P, \pi\rangle \to \langle R, \rho\rangle$ and $\langle R, \rho\rangle \sim_l \langle P_i, \sigma\rangle$. Now consider the program $Q \equiv L := 4$ which clearly belongs to $\mathbf{SIMP}_{par}$. We show that the program $P \| Q$ does not belong to $\mathbf{SIMP}_{lts}$ (and thus neither to $\mathbf{SIMP}_{hpar}$ and $\mathbf{SIMP}_{par}$). Indeed, let σ be a state such that $\sigma(H) = \sigma(L) = 1$. Then $\langle P \| Q, \sigma\rangle \stackrel{\texttt{high}}{\to} \langle P_0 \| Q, \sigma\rangle$. Now let π be a state such that $\pi =_L \sigma$ and in particular $\pi(L) = 1$ but $\pi(H) \neq 1$. Hence $\langle P \| Q, \pi\rangle \stackrel{\texttt{high}}{\to} \langle P_1 \| Q, \pi\rangle$. However, $\langle P_0 \| Q, \sigma\rangle \not\sim_L \langle P_1 \| Q, \pi\rangle$: in fact if the assigment $L := 4$ of Q is performed at the first step, then $\langle P_0 \| Q, \sigma\rangle$ ends in a state σ' such that $\sigma'(L) = 2$ while $\langle P_1 \| Q, \pi\rangle$ ends in a state π' such that $\pi'(L) = 3$. □

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.

In the next definition we introduce a class $\mathcal{C}$ of programs which is closed under composition and it is a subclass of $\mathbf{SIMP}_{par}$ (and then also of $\mathbf{SIMP}_{lts}$ and $\mathbf{SIMP}_{hpar}$).

Definition 9. *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. The class of programs $\mathcal{C}$ is recursively defined as follows.*

1. `skip` *is in* $\mathcal{C}$;
2. $L := a_l$ *is in* $\mathcal{C}$;
3. $H := a_h$ *is in* $\mathcal{C}$;
4. $H := a_l$ *is in* $\mathcal{C}$;
5. $P_0; P_1$ *is in* $\mathcal{C}$ *if* P_0, P_1 *are in* $\mathcal{C}$;
6. `if` b_l `then` P_0 `else` P_1 *is in* $\mathcal{C}$, *if* P_0, P_1 *are in* $\mathcal{C}$;
7. `if` b_h `then` P_0 `else` P_1 *is in* $\mathcal{C}$ *if* P_0, P_1 *are in* $\mathcal{C}$ *and* $P_0 \simeq_l P_1$;
8. `while` b_l `do` P_0 *is in* $\mathcal{C}$, *if* P_0 *is in* $\mathcal{C}$;
9. $P_0 \| P_1$ *is in* $\mathcal{C}$, *if* P_0, P_1 *are in* $\mathcal{C}$.

Theorem 2. *The class of programs $\mathcal{C}$ of Definition 9 is included in* $\mathbf{SIMP}_{par}$.

Proof. We first prove the following claim.

Claim. Let $G, F, R \in \mathcal{C}$. If $\varphi =_L \rho$ then $\langle F, \varphi\rangle \sim_l \langle F, \rho\rangle$. Moreover, if $\langle G, \varphi\rangle \sim_l \langle R, \rho\rangle$, then $\langle G; F, \varphi\rangle \sim_l \langle R; F, \rho\rangle$ and $\langle G \| F, \varphi\rangle \sim_l \langle R \| F, \rho\rangle$.

Proof. It is sufficient to show that

$$\begin{array}{c}\mathcal{S} = \{(\langle G; F, \varphi\rangle, \langle R; F, \rho\rangle), (\langle G \| F, \varphi\rangle, \langle R \| F, \rho\rangle), \mid G, F, R \in \mathcal{C}, \langle G, \varphi\rangle \sim_l \langle R, \rho\rangle\} \\ \cup\{(\langle F, \varphi\rangle, \langle F, \rho\rangle) \mid F \in \mathcal{C}, \varphi =_L \rho\} \\ \cup\{\langle F_0, \varphi\rangle, \langle F_1, \rho\rangle \mid F_0, F_1 \in \mathcal{C}, \varphi =_L \rho, F_0 \sim_l F_1\} \\ \cup\{(\langle F_0, \varphi\rangle, \langle F_1, \rho\rangle) \mid \langle F_0, \varphi\rangle \sim_l \langle F_1, \rho\rangle\}\end{array}$$

is a low level bisimulation.

In order to prove Theorem 2 we show that if $P \in \mathcal{C}$, then for each $F \in \mathit{Reach}(P)$ and for each ψ it holds that if $\langle F, \psi\rangle \xrightarrow{h} \langle G, \varphi\rangle$, then for each π such that $\pi =_L \psi$ we have $\langle F, \pi\rangle \to \langle R, \rho\rangle$ with $\langle R, \rho\rangle \sim_l \langle G, \varphi\rangle$. Indeed, from the fact that $P \in \mathcal{C}$ and $F \in \mathit{Reach}(P)$ we get that $F \in \mathcal{C}$. We prove the thesis for a generic $F \in \mathcal{C}$ and a generic state ψ. We proceed by structural induction on F.

The only interesting cases are $F \equiv F_0; F_1$ and $F \equiv F_0 \| F_1$. We consider the case $F \equiv F_0; F_1$ since the other one is similar. If $\langle F, \psi\rangle \xrightarrow{h} \langle F_0'; F_1, \varphi\rangle$, then we have $\langle F_0, \psi\rangle \xrightarrow{h} \langle F_0', \varphi\rangle$. Hence by inductive hypothesis on F_0 we have $\langle F_0, \pi\rangle \xrightarrow{h} \langle F_0'', \rho\rangle$ with $\langle F_0', \varphi\rangle \sim_l \langle F_0'', \rho\rangle$. Then we get that $\langle F, \pi\rangle \xrightarrow{h} \langle F_0''; F_1, \rho\rangle$ and by Claim 4 $\langle F_0''; F_1, \rho\rangle \sim_l \langle F_0'; F_1, \varphi\rangle$. If $\langle F, \psi\rangle \xrightarrow{h} \langle F_1, \varphi\rangle$, then $\langle F_1, \psi\rangle \xrightarrow{h} \langle \mathtt{end}, \varphi\rangle$. Hence by Claim 4 we get that $\langle F_1, \pi\rangle \xrightarrow{h} \langle \mathtt{end}, \rho\rangle$ with $\rho =_L \varphi$. So, $\langle F, \pi\rangle \xrightarrow{h} \langle F_1, \rho\rangle$, and again by Claim 4 we have $\langle F_1, \rho\rangle \sim_l \langle F_1, \varphi\rangle$. □

We conclude this section by observing that membership to the class $\mathcal{C}$ is not decidable due to the presence of the low level observation equivalence $\simeq_l$ in point 7 of Definition 9. However, a sound but incomplete method could be find to compute $\simeq_l$ by applying a suitable abstraction which guarantees equivalence up to high level locations as discussed, e.g., in [1].

5 Conclusion and Related Work

In this paper we introduced a generalized unwinding schema for the definition of non-interference properties of programs of a simple imperative language, admitting parallel executions on a shared memory. We studied three different instances of our unwinding condition and defined a subclass of programs which is compositional with respect to the language constructors.

There is a widespread literature on secure information flow in imperative languages (see [15] for a recent survey). A common approach is based on types in such a way that well-typed programs do not leak secrets (see, e.g., [16, 17]). Other approaches consider logical formulations of non-interference, e.g., [2, 3, 10], and abstract interpretation-based formalizations, e.g., [5, 7].

As far as we know, this is the first attempt of defining security properties of imperative languages through unwinding conditions. As observed by many authors (see, e.g., [11, 13]) such conditions are easier to handle and more amenable to automated proof with respect to global conditions. Similarly to what we already did in [4] for systems written in a process algebra language, we plan to exploit unwinding conditions for defining proof systems both to verify whether a program is secure and to build programs which are secure by construction in an incremental way.

Finally, we observe that the properties we have defined in terms of unwinding conditions characterize the security of programs againts so-called *passive attacks*, i.e., a low level users which try to infer the values of the high level variables just by observing the values of the low level ones. On the contrary, in defining non-interference one usually explicitly characterize the class of *active attacks*, i.e., malicious users or programs which try to directly transmit confidential information to the low level observer. Some authors have proved that there is a connection between properties characterizing passive attacks and properties involving active attacks [20]. In our approach an active attacker can be seen as a high level program which intentionally manipulates high level variables. We can prove that if P is a secure program belonging to the class $\mathbf{SIMP}_{hpar}$ (and hence also to $\mathbf{SIMP}_{par}$) then a low level user cannot distiguish P running in parallel with different (malicious) high programs P_H and P_K exhibiting the same timing behaviour (i.e., $P_H \simeq_l P_K$).

Theorem 3. *If $P \in \mathbf{SIMP}_{hpar}$ then $P \| P_H \simeq_l P \| P_K$ for all $P_H, P_K \in \mathbf{Prog_H}$ such that $P_H \simeq_l P_K$.*

Proof. It follows from the fact that

$$\mathcal{S} = \{(\langle P \| P_H, \sigma\rangle, \langle Q \| P_K, \theta\rangle) |\ \langle P, \sigma\rangle \sim_l \langle Q, \theta\rangle,\ P_H \sim_l P_K,\ P_H, P_K \in \mathbf{Prog_H}\ \langle P, \sigma\rangle, \langle Q, \theta\rangle \in \mathcal{W}(\sim_l, \dashrightarrow, \mathcal{R}_{hpar})\} \cup \{(\langle P, \sigma\rangle, \langle Q, \theta\rangle) |\ \langle P, \sigma\rangle \sim_l \langle Q, \theta\rangle\}$$

is a low level bisimulation $\sim_l$. □

Intuitively, this theorem states that if a program P belongs to $\mathbf{SIMP}_{hpar}$ then even if the values of the high level variables are changed during the computation, a low level user will never observe any difference on the values of low level variables.

References

1. J. Agat. Transforming out Timing Leaks. In *Proc. of ACM Symposium on Principles of Programming Languages (POPL'00)*, pages 40–53. ACM Press, 2000.
2. T. Amtoft and A. Banerjee. Information Flow Analysis in Logical Form. In *Proceedings of the 11th Static Analysis Symposium (SAS'04)*, volume 3148 of *LNCS*, pages 100–115. Springer-Verlag, 2004.
3. G. Barthe, P. D'Argenio, and T. Rezk. Secure Information Flow by Self Composition. In *Proc. of the 17th IEEE Computer Security Foundations Workshop (CSFW'04)*, pages 100–114. IEEE Computer Society Press, 2004.
4. A. Bossi, R. Focardi, C. Piazza, and S. Rossi. Verifying Persistent Security Properties. *Computer Languages, Systems and Structures*, 30(3-4):231–258, 2004.
5. A. Di Pierro, C. Hankin, and H.Wiklicky. Approximate Non-Interference. In *Proc. of the IEEE Computer Security Foundations Workshop (CSFW'02)*, pages 3–17. IEEE Computer Society Press, 2002.
6. R. Focardi and R. Gorrieri. Classification of Security Properties (Part I: Information Flow). In R. Focardi and R. Gorrieri, editors, *Proc. of Foundations of Security Analysis and Design (FOSAD'01)*, volume 2171 of *LNCS*, pages 331–396. Springer-Verlag, 2001.
7. R. Giacobazzi and I. Mastroeni. Abstract Non-Interference: Parameterizing Non-Interference by Abstract Interpretation. In *Proc. of ACM Symposium on Principles of Programming Languages (POPL'04)*, pages 186–197. ACM Press, 2004.
8. J. A. Goguen and J. Meseguer. Security Policies and Security Models. In *Proc. of the IEEE Symposium on Security and Privacy (SSP'82)*, pages 11–20. IEEE Computer Society Press, 1982.
9. J. A. Goguen and J. Meseguer. Unwinding and Inference Control. In *Proc. of the IEEE Symposium on Security and Privacy (SSP'84)*, pages 75–86. IEEE Computer Society Press, 1984.
10. R. Joshi and K. R. M. Leino. A Semantic Approach to Secure Information Flow. *Science of Computer Programming*, 37(1–3):113–138, 2000.
11. H. Mantel. Unwinding Possibilistic Security Properties. In *Proc. of the European Symposium on Research in Computer Security (ESoRiCS'00)*, volume 2895 of *LNCS*, pages 238–254. Springer-Verlag, 2000.
12. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
13. P. Y. A. Ryan. A CSP Formulation of Non-Interference and Unwinding. *Cipher*, pages 19–27, 1991.
14. P.Y.A. Ryan and S. Schneider. Process Algebra and Non-Interference. *Journal of Computer Security*, 9(1/2):75–103, 2001.
15. A. Sabelfeld and A. C. Myers. Language-Based Information-Flow Security. *IEEE Journal on Selected Areas in Communication*, 21(1):5–19, 2003.
16. A. Sabelfeld and D. Sands. A Per Model of Secure Information Flow in Sequential Programs. *Higher-Order and Symbolic Computation*, 14(1):59–91, 2001.
17. D. M. Volpano and G. Smith. A Type-Based Approach to Program Security. In *TAPSOFT*, pages 607–621, 1997.
18. D. M. Volpano and G. Smith. Probabilistic Noninterference in a Concurrent Language. *Journal of Computer Security*, 7(2-3):231 – 253, 1999.
19. G. Winskel. *The formal semantics of programming languages*. The MIT Press, 1993.
20. S. Zdancewic and A. C. Myers. Robust Declassification. In *Proc. of the IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 15–23. IEEE Computer Society Press, 2001.