Precise Dynamic Verification of Noninterference

Gurvan Le Guernic

Precise Dynamic Verification of Noninterference

Gurvan Le Guernic

INRIA-MSR - Parc Orsay Universit, 91893 Orsay - France
http://www.Le-Guernic.info
gleguern@gmail.com

Abstract

Confidentiality is maybe the most popular security property to be formally or informally verified. Noninterference is a baseline security policy to formalize confidentiality of secret information manipulated by a program. Many static analyses have been developed for the verification of noninterference. In contrast to those static analyses, this paper considers the run-time verification of the respect of confidentiality by a single execution of a program. It proposes a dynamic noninterference analysis for sequential programs based on a combination of dynamic and static analyses. The static analysis is used to analyze some unexecuted pieces of code in order to take into account all types of flows. The static analysis is sensitive to the current program state. This sensitivity allows the overall dynamic analysis to be more precise than previous work. The soundness of the overall dynamic noninterference analysis with regard to confidentiality breaches detection and correction is proved.

1 Introduction

Language-based security is an active field of research. The majority of work on confidentiality in this field focuses on static analyses [13]. Recent years have seen a resurgence of dynamic analyses aiming at enforcing confidentiality at run time [5, 7, 14]. The first reason is that nowadays it is nearly impossible for consumers to prevent the execution of "bad" code on their devices — for example, in September 2007 Yahoo served 12 million times a flash ad installing a Trojan horse [4] and cybercriminals introduced malicious scripts into webpages of a US Consulate [8]. Moreover, there are two main potential advantages of dynamic analyses over static analyses [7]. The first one is the increased knowledge of the execution environment and behavior at run time, including the knowledge of the precise control flow followed by the current execution. This increased knowledge allows the dynamic analysis to be more precise than a static analysis in some cases; as, for example, with the program on page 14. The second advantage lies in the ability of sound information flow monitors to run some "safe" executions of an "unsafe" program while still guarantying the confidentiality of secret data. In order to take into account all indirect flows (flows originating in control statements) dynamic analyses relies on static analyses of some, but not all, unexecuted pieces of code.

This paper proposes to increase the precision of such dynamic information flow analyses. This is done by taking advantage, at the static analysis level, of the dynamic nature of the overall analysis. To do so, when statically analyzing an unexecuted piece of code, the current program state is taken into account in order to reduce the program space to analyze. The following piece of code is a motivating example for this work. It corresponds to the body of the main loop of an Instant Messaging (IM) program. This one has the appealing "movie-like" feature of displaying messages characters by characters as they are typed.

```
1 c := getCharFromKeyboard();
2 tmp := tmp + ((int) c);
3 if ( tmp > ((int) userSecretKey) ) {
4   tmp := 0;
5   if (to = "sexyPirate") {c := specialChar}
6 };
7 send(to, c)
```

This IM program is a malware developed by "sexyPirate". When someone uses this software to communicate with a user other than sexyPirate, everything goes as expected and no secret is revealed. However, if a user communicates with sexyPirate using this IM then information about the user's secret key is leaked to the pirate. When the integer value of the characters typed by the user since the last time *tmp* has been reset to 0 reaches the integer value of the user's secret key, a special character (that sexyPirate is able to distinguish) appears on the pirate's screen. Therefore, by iterating the process, sexyPirate is able to get an accurate approximation of the user's secret key. Any sound static

analysis would reject this program; and therefore, all its executions. One of the advantages of dynamic information flow analysis, if it is precise enough, is to allow use of this program for communicating with users other than sexyPirate, while still guarantying the confidentiality of the secret key in any case. However, none of the previous work are precise enough. When statically analyzing lines 4 and 5, no knowledge about the value of the variable *to* is taken into account. Therefore, the overall dynamic analysis will always consider that the value of *c* may be modified; which implies a flow from *userSecretKey* to *c*. With such dynamic analyses, line 7 must then be corrected in order to prevent any potential leakage of the value of the secret key to the outside world. In the work proposed in this paper, the static analysis used for lines 4 and 5 takes into account the run time value of the variable *to*. This allows the overall dynamic analysis proposed to detect that there is no flow from *userSecretKey* to *c* whenever *to* is different from `sexyPirate`. Therefore, it allows use of this IM program for communicating with any user different from sexyPirate; while still preserving the confidentiality of the user's secret key even when trying to use this program to communicate with sexyPirate.

The next section defines various notions used in this paper and introduces the principles of the dynamic analysis proposed in this paper. Section 3 formalizes the dynamic information flow analysis. The main properties of this analysis are exposed in Sect. 4 before concluding in Sect. 6.

2 Definitions and Principles

The current section starts by introducing some terminology. It then gives some formal definitions which are used to formalize the execution property that the dynamic analysis has to verify. Subsequently, it introduces succinctly the principles of the dynamic analysis.

A *direct flow* is a flow from the right side of an assignment to the left side. Executing "x **:=** y" creates a direct flow from *y* to *x*. An *explicit indirect flow* is a flow from the test of a conditional to the left side of an assignment in the branch executed. Executing "**if** c **then** x **:=** y **else skip end**" when *c* is `true` creates an explicit indirect flow from *y* to *x*. An *implicit indirect flow* is a flow from the test of a conditional to the left side of an assignment in the branch which is not executed. Executing "**if** c **then** x **:=** y **else skip end**" when *c* is `false` creates an implicit indirect flow from *y* to *x*.

At any execution step, a variable or expression is said to *carry variety* [2, Sect.1] if its values is not completely constrained by the public inputs of the program. In other words, a variable or expression *carries variety* if its value is influenced by the private inputs; therefore, if it may have a different value at this given execution step if the values of the private inputs were different.

A "safe" execution is a noninterfering execution. In this article, as commonly done, noninterference is defined as the absence of strong dependencies between the secret inputs of an execution and the final values of some variables

which are considered to be publicly observable at the end of the execution. For every execution of a given program P, two sets of variable identifiers are defined. The set of variables corresponding to the secret inputs of the program is designated by $\mathcal{S}(\mathtt{P})$. The set of variables whose final value are publicly observable at the end of the execution is designated by $\mathcal{O}(\mathtt{P})$. No requirements are put on $\mathcal{S}(\mathtt{P})$ and $\mathcal{O}(\mathtt{P})$ other than requiring them to be subsets of $\mathbb{X}$ (the domain of variables). A variable x is even allowed to belong to both sets. In such a case, in order to be noninterfering, the program P would be required to, at least, reset the value of x. In the following definitions, we consider that a program state may contain more than just a value store. This is the reason why a distinction is done between program states (X) and value stores (σ).

Definition 2.1 (V-Equivalent States)**.**
Let V be a set of variables. Two program states X_1, respectively X_2, containing the value stores σ_1, respectively σ_2, are V-equivalent *with regards to a set of variables V, written $X_1 \overset{V}{=} X_2$, if and only if the value of any variable belonging to V is the same in σ_1 and σ_2:*

$$X_1 \overset{V}{=} X_2 \quad \Longleftrightarrow \quad \forall x \in V : \sigma_1(x) = \sigma_2(x)$$

Definition 2.1 states a formal relation among program states. This relation defines equivalence classes of program states with regard to a given set of variables. If two program states are V-equivalent, it means that it is impossible to distinguish them solely by looking at the value of the variables belonging to the set V. This relation is used to define the confidentiality property which is verified by the dynamic analysis presented in this paper.

Definition 2.2 (Noninterfering Execution)**.**
Let $\Downarrow_s$ denote a big-step semantics. Let $\overline{\mathcal{S}(P)}$ be the complement of $\mathcal{S}(P)$ in the set $\mathbb{X}$. For all programs P, program states X_1 and X_1', an execution with the semantics $\Downarrow_s$ of the program P in the initial state X_1 and yielding the final state X_1' is noninterfering, written $ni(P, s, X_1)$, if and only if, for every program states X_2 and X_2' such that the execution with the semantics $\Downarrow_s$ of the program P in the initial state X_2 yields the final state X_2':

$$X_1 \overset{\overline{\mathcal{S}(P)}}{=} X_2 \;\Rightarrow\; X_1' \overset{\mathcal{O}(P)}{=} X_2'$$

Definition 2.2 states that an execution is safe — i.e. it has the desired confidentiality property — if any other execution started with the same public (non-secret) values yields a final program state which is $\mathcal{O}(\mathtt{P})$-equivalent to the final program state of the execution analyzed. It means that, by looking only at the final values of the variables observable at the end of the execution, it is impossible to distinguish this execution from any other execution whose initial program state differs only in the values of the secret inputs. Therefore, for such an execution, it is impossible to deduce information about the secret inputs of the program by looking solely at the values of the publicly observable outputs.

The dynamic analysis is based on a flow and state sensitive static analysis. During the execution, every variable is associated a tag which reflects the fact that the variable *may* or *may not* carry variety — i.e. *may* or *may not* be influenced by the secret inputs of the program. A tag store in the program state keeps track of those associations. The dynamic analysis treats directly the direct and explicit indirect flows. For implicit indirect flows, a static analysis is run on the unexecuted branch of every conditional whose test carries variety.

The static analysis is context sensitive. An unexecuted branch `P` is analyzed in the context of the program state at the time the test of the conditional, to which `P` belongs, has been evaluated. The static analysis is then aware of the exact value of the variables which do not carry variety. During the analysis, the context (value store and tag store used for the analysis) is modified to reflect loss of knowledge (in fact, only the tag store is modified). The static analysis does not compute the values of variables. Therefore, when analyzing an assignment to a variable x, the context of the static analysis is modified to reflect the fact that the static analysis does not anymore have knowledge of the precise value of the variable x. When analyzing a conditional whose test value can be computed in the current context (using only the values of the variables whose tag is $\bot$), only the branch designated by the test is analyzed. As the value of any variable which does not carry variety depends only on the public inputs, branches which are not designated by the test value would never be executed by any execution started with the same public inputs as the monitored execution. Implicit indirect flows and explicit indirect flows must be treated with the same precision in order to prevent the creation of a new covert channel [6]. This particular point is discussed in Sect. 4. As the static analysis detects implicit indirect flows more accurately than context insensitive analyses, explicit indirect flows can also be treated more accurately.

The next section formalizes the mechanisms presented above. It presents a monitoring semantics incorporating a dynamic noninterference analysis.

3 The Monitoring Semantics

The dynamic information flow analysis and the monitoring semantics are defined together in Fig. 2. Information flows are tracked using tags. At any execution step, every variable has a tag which reflects whether this variable may carry variety or not. The static analysis used for the analysis of some unexecuted branches is characterized in Fig. 3.

The language studied is an imperative language for sequential programs. Its syntax is given in Fig. 1. In this grammar, $\langle ident \rangle$ stands for a variable identifier. $\langle expr \rangle$ is an expression of values and variable identifiers. Expressions in this language are deterministic — their evaluation in a given program state always results in the same value — and are free of side effects — their evaluation has no influence on the program state.

A program expressed with this language is either a skip statement (**skip**)

$$
\begin{array}{rcl}
\langle prog \rangle & ::= & \textbf{skip} \\
& | & \langle ident \rangle \textbf{ := } \langle expr \rangle \\
& | & \langle prog \rangle \text{ ; } \langle prog \rangle \\
& | & \textbf{if } \langle expr \rangle \textbf{ then } \langle prog \rangle \textbf{ else } \langle prog \rangle \textbf{ end} \\
& | & \textbf{while } \langle expr \rangle \textbf{ do } \langle prog \rangle \textbf{ done}
\end{array}
$$

Figure 1: Grammar of the language

which has no effect, an assignment of the value of an expression to a variable, a sequence of programs ($\langle prog \rangle$; $\langle prog \rangle$), a conditional executing one program — out of two — depending on the value of a given expression (`if` statements), or a loop executing repetitively a given program as long as a given expression is true (`while` statements).

3.1 A Semantics Making Use of Static Analysis Results

Let $\mathbb{X}$ be the domain of variable identifiers, $\mathbb{D}$ be the semantics domain of values, and $\mathbb{T}$ be the domain of tags. In the remainder of this article, $\mathbb{T}$ is equal to $\{\top, \bot\}$. Those tags form a lattice such that $\bot \sqsubset \top$. $\top$ is the tag associated to variables that *may* carry variety — i.e. whose value may be influenced by the secret inputs.

The monitoring semantics described in Fig. 2 is presented as standard inference rules for sequents written in the format:

$$\zeta,\ t^{\mathrm{pc}} \ \vdash\ \mathtt{P}\ \Downarrow_{\mathcal{M}}\ \zeta'$$

This reads as follows: in the monitoring execution state ζ, with a program counter tag equal to t^{pc}, program `P` yields the monitoring execution state ζ'. The program counter tag (t^{pc}) is a tag which reflects the security level of the information carried by the control flow. A monitoring execution state ζ is a pair $(\sigma,\ \rho)$ composed of a value store σ and a tag store ρ. A value store ($\mathbb{X} \to \mathbb{D}$) maps variable identifiers to values. A tag store ($\mathbb{X} \to \mathbb{T}$) maps variable identifiers to tags. The definitions of value store and tag store are extended to expressions. $\sigma(e)$ is the value of the expression e in a program state whose value store is σ. Similarly, $\rho(e)$ is the tag of the expression e in a program state whose tag store is ρ. $\rho(e)$ is formally defined as follows, with $FV(e)$ being the set of free variables appearing in the expression e:

$$\rho(e) = \bigsqcup_{x \in FV(e)} \rho(x)$$

The semantics rules make use of static analyses results. In Fig. 2, application of a static information flow analysis to the piece of code P in the context ζ is written: $[\![\zeta \vdash \mathtt{P}]\!]^{\sharp_{IF}}$. The analysis of a program P in a monitoring execution state ζ must return a subset of $\mathbb{X}$. This set, usually written $\mathfrak{X}$, is an over-approximation of the set of variables which are potentially defined in an execution of P in the context ζ. This static information flow analysis can be any such analysis that satisfies a set of formal constraints which are stated in Sect. 3.2.

The monitoring semantics rules are straightforward. As can be expected, the execution of a **skip** statement with the semantics given in Fig. 2 yields a final state equal to the initial state. The monitored execution of the assignment of the value of the expression e to the variable x yields a monitored execution state $(\sigma',\ \rho')$. The final value store (σ') is equal to the initial value store (σ) except for the variable x. The final value store maps the variable x to the value of the expression e evaluated with the initial value store $(\sigma(e))$. Similarly, the final tag store (ρ') is equal to the initial tag store (ρ) except for the variable x. The tag of x after the execution of the assignment is the least upper bound of the program counter tag (t^{pc}) and the tag of the expression computed using the initial tag store $(\rho(e))$. $\rho(e)$ corresponds to the level of the information flowing into x through direct flows. t^{pc} corresponds to the level of the information flowing into x through explicit indirect flows.

The monitored execution of a conditional whose test expression does not carry variety $(\rho(e) = \bot)$ follows the same scheme as with a standard semantics. For a conditional whose test expression e carries variety, the branch (P^v) designated by the value of e (v) is executed and the other one $(P^{\neg v})$ is analyzed. The final value store is the one returned by the execution of P^v. The final tag store (ρ') is the least upper bound of the tag store returned by the execution of P^v and a new tag store (ρ^e) generated from the result of the analysis of $P^{\neg v}$ $(\mathfrak{X})$. By definition, $\rho \sqcup \rho'$ is equal to $\lambda x.\rho(x) \sqcup \rho'(x)$. The new tag store (ρ^e) reflects the implicit indirect flows between the value of the test of the conditional and the variables $(\mathfrak{X})$ which may be defined in an execution of $P^{\neg v}$. In ρ^e, the tag of a variable x is equal to the initial tag of the test expression of the conditional $(\rho(e))$ if and only if x belongs to $\mathfrak{X}$; otherwise, its tag is $\bot$.

3.2 The Static Analysis Used

Fig. 3 defines some constraints characterizing a set of static analyses which can be used by the dynamic noninterference analysis. The result $\mathfrak{X}$ of a static analysis of a given program (P) in a given context (ζ) is *acceptable* for the dynamic analysis only if the result satisfies those rules. This is written in the format: $\mathfrak{X} \models (\zeta \vdash \mathtt{P})$. In the definitions of those rules, $\{\![S^{\mathtt{true}}, S^{\mathtt{false}}]\!\}^t_v$ returns either the set $S^{\mathtt{true}}$, the set $S^{\mathtt{false}}$ or the union of both depending on the tag t and the boolean v. Its formal definition follows.

$$\{\![S^{\mathtt{true}}, S^{\mathtt{false}}]\!\}^t_v = \begin{cases} S^{\mathtt{true}} \cup S^{\mathtt{false}} & \text{iff } t = \top \\ S^v & \text{iff } t = \bot \end{cases}$$

$$\frac{}{\zeta,\ t^{\mathrm{pc}}\ \vdash\ \mathbf{skip}\ \Downarrow_{\mathcal{M}}\ \zeta}$$

$$\frac{}{(\sigma,\ \rho),\ t^{\mathrm{pc}}\ \vdash\ x := e\ \Downarrow_{\mathcal{M}}\ (\sigma[x \mapsto \sigma(e)],\ \rho[x \mapsto \rho(e) \sqcup t^{\mathrm{pc}}])}$$

$$\frac{\zeta,\ t^{\mathrm{pc}}\ \vdash\ P^{\mathrm{h}}\ \Downarrow_{\mathcal{M}}\ \zeta^{\mathrm{h}} \qquad \zeta^{\mathrm{h}},\ t^{\mathrm{pc}}\ \vdash\ P^{\mathrm{t}}\ \Downarrow_{\mathcal{M}}\ \zeta'}{\zeta,\ t^{\mathrm{pc}}\ \vdash\ P^{\mathrm{h}}\ ;\ P^{\mathrm{t}}\ \Downarrow_{\mathcal{M}}\ \zeta'}$$

$$\frac{\rho(e) = \bot \qquad \sigma(e) = v \qquad (\sigma,\ \rho),\ t^{\mathrm{pc}} \sqcup \bot\ \vdash\ P^{v}\ \Downarrow_{\mathcal{M}}\ \zeta'}{(\sigma,\ \rho),\ t^{\mathrm{pc}}\ \vdash\ \mathbf{if}\ e\ \mathbf{then}\ P^{\mathtt{true}}\ \mathbf{else}\ P^{\mathtt{false}}\ \mathbf{end}\ \Downarrow_{\mathcal{M}}\ \zeta'}$$

$$\frac{\begin{array}{c}\rho(e) = \top \qquad \sigma(e) = v \qquad (\sigma,\ \rho),\ t^{\mathrm{pc}} \sqcup \top\ \vdash\ P^{v}\ \Downarrow_{\mathcal{M}}\ (\sigma^{v},\ \rho^{v}) \\ [\![(\sigma,\ \rho) \vdash P^{\neg v}]\!]^{\sharp_{\mathcal{G}}} = \mathfrak{X} \qquad \rho^{\mathrm{e}} = (\mathfrak{X} \times \{\top\}) \cup (\overline{\mathfrak{X}} \times \{\bot\})\end{array}}{(\sigma,\ \rho),\ t^{\mathrm{pc}}\ \vdash\ \mathbf{if}\ e\ \mathbf{then}\ P^{\mathtt{true}}\ \mathbf{else}\ P^{\mathtt{false}}\ \mathbf{end}\ \Downarrow_{\mathcal{M}}\ (\sigma^{v},\ \rho^{v} \sqcup \rho^{\mathrm{e}})}$$

$$\frac{\rho(e) = \bot \qquad \sigma(e) = \mathtt{false}}{(\sigma,\ \rho),\ t^{\mathrm{pc}}\ \vdash\ \mathbf{while}\ e\ \mathbf{do}\ P^{\mathrm{l}}\ \mathbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma,\ \rho)}$$

$$\frac{\rho(e) = \bot \qquad \sigma(e) = \mathtt{true} \qquad (\sigma,\ \rho),\ t^{\mathrm{pc}} \sqcup \bot\ \vdash\ P^{\mathrm{l}}\ ;\ \mathbf{while}\ e\ \mathbf{do}\ P^{\mathrm{l}}\ \mathbf{done}\ \Downarrow_{\mathcal{M}}\ \zeta'}{(\sigma,\ \rho),\ t^{\mathrm{pc}}\ \vdash\ \mathbf{while}\ e\ \mathbf{do}\ P^{\mathrm{l}}\ \mathbf{done}\ \Downarrow_{\mathcal{M}}\ \zeta'}$$

$$\frac{\rho(e) = \top \qquad \sigma(e) = \mathtt{true} \qquad (\sigma,\ \rho),\ t^{\mathrm{pc}} \sqcup \top\ \vdash\ P^{\mathrm{l}}\ ;\ \mathbf{while}\ e\ \mathbf{do}\ P^{\mathrm{l}}\ \mathbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma',\ \rho^{\mathrm{e}})}{(\sigma,\ \rho),\ t^{\mathrm{pc}}\ \vdash\ \mathbf{while}\ e\ \mathbf{do}\ P^{\mathrm{l}}\ \mathbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma',\ \rho \sqcup \rho^{\mathrm{e}})}$$

$$\frac{\begin{array}{c}\rho(e) = \top \qquad \sigma(e) = \mathtt{false} \\ [\![(\sigma,\ \rho) \vdash P^{\mathrm{l}}\ ;\ \mathbf{while}\ e\ \mathbf{do}\ P^{\mathrm{l}}\ \mathbf{done}]\!]^{\sharp_{\mathcal{G}}} = \mathfrak{X} \\ \rho^{\mathrm{e}} = (\mathfrak{X} \times \{\top\}) \cup (\overline{\mathfrak{X}} \times \{\bot\})\end{array}}{(\sigma,\ \rho),\ t^{\mathrm{pc}}\ \vdash\ \mathbf{while}\ e\ \mathbf{do}\ P^{\mathrm{l}}\ \mathbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma,\ \rho \sqcup \rho^{\mathrm{e}})}$$

Figure 2: Rules of the monitoring semantics

4 Properties of the Monitoring Semantics

Section 3 formally defines the dynamic information flow analysis proposed in this article. In the current section, this dynamic noninterference analysis is proved to be sound with regard to the enforcement of the notion of noninterfering execution given in Definition 2.2. This means that, any monitor enforcing

$$\emptyset \models (\ (\sigma,\ \rho) \vdash \textbf{skip}\ )$$

$$\{x\} \models (\ (\sigma,\ \rho) \vdash x := e\ )$$

$$\mathfrak{X} \models (\ (\sigma,\ \rho) \vdash \mathtt{P}^{\mathrm{h}}\ ;\ \mathtt{P}^{\mathrm{t}}\ )$$
iff there exist $\mathfrak{X}^{\mathrm{h}}$ and $\mathfrak{X}^{\mathrm{t}}$ such that:
$\mathfrak{X}^{\mathrm{h}} \models ((\sigma,\ \rho) \vdash \mathtt{P}^{\mathrm{h}})$
let $\rho' = \rho \sqcup ((\mathfrak{X}^{\mathrm{h}} \times \top) \cup (\overline{\mathfrak{X}^{\mathrm{h}}} \times \bot))$ in $\mathfrak{X}^{\mathrm{t}} \models ((\sigma, \rho') \vdash \mathtt{P}^{\mathrm{t}})$
$\mathfrak{X} = \mathfrak{X}^{\mathrm{h}} \cup \mathfrak{X}^{\mathrm{t}}$

$$\mathfrak{X} \models (\ (\sigma,\ \rho) \vdash \textbf{if}\ e\ \textbf{then}\ \mathtt{P}^{\mathtt{true}}\ \textbf{else}\ \mathtt{P}^{\mathtt{false}}\ \textbf{end}\ )$$
iff there exist $\mathfrak{X}^{\mathtt{true}}$ and $\mathfrak{X}^{\mathtt{false}}$ such that:
$\mathfrak{X}^{\mathtt{true}} \models ((\sigma,\ \rho) \vdash \mathtt{P}^{\mathtt{true}})$
$\mathfrak{X}^{\mathtt{false}} \models ((\sigma,\ \rho) \vdash \mathtt{P}^{\mathtt{false}})$
$\mathfrak{X} = \{\!|\mathfrak{X}^{\mathtt{true}}, \mathfrak{X}^{\mathtt{false}}|\!\}^{\rho(e)}_{\sigma(e)}$

$$\mathfrak{X} \models (\ (\sigma,\ \rho) \vdash \textbf{while}\ e\ \textbf{do}\ \mathtt{P}^{\mathrm{l}}\ \textbf{done}\ )$$
iff there exists $\mathfrak{X}^{\mathrm{l}}$ such that:
let $\rho' = \rho \sqcup ((\mathfrak{X}^{\mathrm{l}} \times \top) \cup (\overline{\mathfrak{X}^{\mathrm{l}}} \times \bot))$ in $\mathfrak{X}^{\mathrm{l}} \models ((\sigma, \rho') \vdash \mathtt{P}^{\mathrm{l}})$
$\mathfrak{X} = \{\!|\mathfrak{X}^{\mathrm{l}}, \emptyset|\!\}^{\rho(e)}_{\sigma(e)}$

Figure 3: Constraints on the static analysis results

noninterference using this dynamic analysis would be able to ensure that: for any two monitored executions of a given program P started with the same public inputs (variables which do not belong to $\mathcal{S}(\mathtt{P})$), the final values of observable outputs (variables which belong to $\mathcal{O}(\mathtt{P})$) are the same for both executions.

Theorem 4.1 proves that the dynamic analysis is sound with regard to information flow *detection*. Any variable, whose tag at the end of the execution is $\bot$, has the same final value for any executions started with the same public inputs.

Theorem 4.1 (Detection Soundness)**.**
For all programs P, monitoring execution states $(\sigma_1,\ \rho_1)$, $(\sigma_1',\ \rho_1')$, $(\sigma_2,\ \rho_2)$ *and* $(\sigma_2',\ \rho_2')$ *such that:*

- $(\sigma_1,\ \rho_1),\ \bot\ \vdash\ P\ \Downarrow_{\mathcal{M}}\ (\sigma_1',\ \rho_1')$,
- $(\sigma_2,\ \rho_2),\ \bot\ \vdash\ P\ \Downarrow_{\mathcal{M}}\ (\sigma_2',\ \rho_2')$,

- $\forall x \notin \mathcal{S}(P).\ \sigma_1(x) = \sigma_2(x)$,
- $\forall x \in \mathcal{S}(P).\ \rho_1(x) = \top$

the following holds:

$$\forall x. \quad (\rho_1'(x) = \bot) \ \Rightarrow \ (\sigma_1'(x) = \sigma_2'(x))$$

Proof summary. The detailed formal proof can be found in the appendices (Lemma A.6). This proof goes by induction on the derivation tree of " $(\sigma_1,\ \rho_1),\ \bot \vdash \mathtt{P} \Downarrow_{\mathcal{M}} (\sigma_1',\ \rho_1')$ " and by cases on the last evaluation rule used. The proof aims at showing that the *invariant* which relates the fact, for every variable, of having a $\bot$ tag and not carrying variety — i.e. not being influenced by the secret inputs — is preserved during the execution.

The invariant preservation is obvious for **skip** statements. If the tag of a variable x after an assignment of e to x is $\bot$, then it means, first, that the control flow does not carry variety ($t^{\mathrm{pc}} = \bot$) and therefore that the assignment will always be executed with similar public inputs. It also means that the tag of every variable appearing in e is $\bot$, therefore the expression does not carry variety and the invariant property is preserved by the execution of assignments. For sequence statements, if the invariant is preserved by the first and second statements then it is preserved by the sequential execution of both statements.

For branching statements (`if` statements), things get a little bit more complex. If the tag of the condition of the branching statement is $\bot$, then the condition does not carry variety and any execution started with the same public inputs evaluates the same branch as the execution monitored. As, by induction, the execution of the branch preserves the invariant, the invariant is also preserved by the execution of a branching statement whose condition's tag is $\bot$. If the condition's tag is $\top$, then all the executions started with the same public inputs do not evaluate the same branch. However, as the tag of the control flow (t^{pc}) is updated to $\top$ for the evaluation of the designated branch, all the variables assigned to in the branch have a tag of $\top$ after the execution of the branching statement (follows from the rule for assignments and is detailed in Lemma A.5 in the appendices). Additionally, any variable which may have been assigned to by the execution of the other branch are in the set returned by the static analysis (follows from the rules in Fig. 3 and is detailed in Lemma A.1 in the appendices), therefore, at the end of the evaluation of the branching statement, their tag is also $\top$. Hence, the invariant relating $\bot$ and not carrying variety is preserved by the execution of a branching statement whose condition carries variety.

For loops (`while` statements), the proof goes by cases, first, on the tag of the condition and then on its value. If the condition of the loop does not carry variety (its tag is $\bot$) then, if its value is false the loop is equivalent to a **skip** and the invariant is preserved. If the condition does not carry variety (its tag is $\bot$) and is true, the proof of preservation of the invariant follows by induction from the sub-derivation tree corresponding to the evaluation of the body of the loop at least once. If the condition carries variety (its tag is $\top$), then all the

executions started with the same public inputs do not execute the body of the loop the same number of times. However, for the same reasons as the case for branching statements ($t^{\mathrm{pc}} = \top$ and the static analysis of the body of the loop returns all the variables which may be assigned by an evaluation of the loop), the tag of all the variables whose value may be modified by the loop is $\top$ after the evaluation of the loop. Therefore, the invariant property is also preserved by loops whose condition carries variety. □

Variables whose final tags are not $\bot$ may have different final values for two executions started with the same public inputs. Therefore, a monitor *enforcing* noninterference must reset to a default value any variable belonging to $\mathcal{O}(\mathtt{P})$ whose final tag is not $\bot$. However, as shown by Le Guernic and Jensen [6], if the correction of "bad" information flows is done without enough care, the correction mechanism itself can become a new covert channel carrying secret information. Theorem 4.2 proves that the final tag of a variable does not depend on the secret inputs of the program. Therefore, any variable belonging to $\mathcal{O}(\mathtt{P})$ whose final tag is not $\bot$ can safely be reset to a default value without creating a new covert channel.

Theorem 4.2 (Correction Soundness)**.**
For all programs P*, monitoring execution states* $(\sigma_1,\ \rho_1)$*,* $(\sigma_1',\ \rho_1')$*,* $(\sigma_2,\ \rho_2)$ *and* $(\sigma_2',\ \rho_2')$ *such that:*

- $(\sigma_1,\ \rho_1),\ \bot\ \vdash\ P\ \Downarrow_{\mathcal{M}}\ (\sigma_1',\ \rho_1')$,
- $(\sigma_2,\ \rho_2),\ \bot\ \vdash\ P\ \Downarrow_{\mathcal{M}}\ (\sigma_2',\ \rho_2')$,
- $\forall x \notin \mathcal{S}(P).\ \sigma_1(x) = \sigma_2(x)$,
- $\forall x \in \mathcal{S}(P).\ \rho_1(x) = \top$
- $\rho_1 = \rho_2$

the following holds: $\rho_1' = \rho_2'$.

Proof summary. The detailed formal proof can be found in the appendices (Lemma A.8). This proof goes by induction on the derivation tree of " $(\sigma_1,\ \rho_1),\ \bot\ \vdash\ \mathtt{P}\ \Downarrow_{\mathcal{M}}\ (\sigma_1',\ \rho_1')$ " and by cases on the last evaluation rule used. In order to be able to use induction, Lemma A.8 proves a generalization of the theorem stated above with two differences in its statement. The first one concerns the tag of the program counter (t^{pc}) which is not constrained to be $\bot$, but must be the same for the two executions compared. The second difference is the substitution of the 2 hypotheses concerning $\mathcal{S}(\mathtt{P})$ by a single hypothesis stating that the variables containing data which is considered non secret ($\rho_1(x) = \bot$) must have the same value in both executions ($\sigma_1(x) = \sigma_2(x)$).

The case for **skip** is direct. For assignments, the only tag modified is the one of the variable assigned. It is set to $\rho(e) \sqcup t^{\mathrm{pc}}$. As the two tag stores are initially equal, $\rho_1(e)$ is equal to $\rho_2(e)$. As explained above, the proof is for a generalization of the theorem where t^{pc_1} is equal to t^{pc_2}. Therefore, both tag

stores (ρ'_1 and ρ'_2) are equal after the execution of the assignment. For the sequential execution of S_1 ; S_2, the inductive hypothesis implies that both tag stores are equal before the execution of S_2 and the generalization of Theorem 4.1 (Lemma A.6) implies that the hypothesis $\rho_1(x) = \bot \Rightarrow \rho_1(e) = \rho_2(e)$ also holds before the execution of S_2. Therefore the inductive hypothesis can be applied and implies that both tag stores are equal after the execution of the sequential statement.

For `if` statements, if the tag of the condition is $\bot$ then the condition evaluates to the same value for both executions. Therefore, the same branch is executed and the inductive hypothesis implies that both tag stores are equal at the end of the execution of the branching statement. If the tag of the condition is $\top$ but both executions evaluate the same branch, then for similar reasons both tag stores are equal at the end of the execution of the branching statement. If two different branches are executed then the tag of the program counter is $\top$. Therefore, as exposed in the proof of Theorem 4.1 and detailed in Lemma A.5, the tag of every assigned variable is $\top$ (as long as the execution of the branch is not over new tags are always $\top$). Additionally, during the execution of this branch, for every branching statement whose condition's tag is $\top$ the unexecuted branch is analyzed and the tag of every variable which may have been assigned to by an execution of this branch is set to $\top$. Fig. 3 constrains the analysis to make the same choices with regard to which subbranches to ignore and which ones to analyze. Therefore, the set of variables returned by the analysis of the unexecuted branch is exactly the set of variables whose tag would have been set to $\top$ by an execution of this branch (this particular point is detailed in Lemma A.3). Therefore, whatever branch is executed or analyzed, the same set of variables have their tag set to $\top$. Hence, the final tag stores are equal after the execution of the `if` statement.

For `while` statements, if the condition evaluates to the same value for both executions then the inductive hypothesis implies that the tag stores at the end of both executions are equal. If the condition evaluates to two different values then it means that its tag is $\top$. Therefore the final tag store is the least upper bound of the initial tag store (equal for both executions) and a new tag store ρ^e. This new tag store is either the tag store returned by the execution of the `while` statement (executing the body at least once) with program counter tag (t^{pc}) equal to $\top$ or the analysis of the same statement. As exposed in the case of `if` statements (and detailed in Lemma A.3), in both cases the tags of the exact same set of variables are set to $\top$ and they are the only tag modified. Hence, the final tag stores are equal after the execution of the `while` statement. □

5 Related Work

The vast majority of research on noninterference concerns static analyses and involves type systems [11, 13]. Some "real size" languages together with security type system have been developed (for example, JFlow/JIF [10] and FlowCaml [12]).

Dynamic information flow analyses [1, 3, 17, 18] are not as popular as static analyses for information flow, but there has been interesting research. For example, RIFLE [15] is a complete run-time information flow security system based on an architectural framework and a binary translator. Masri et al. [9] present a dynamic information flow analysis for structured or unstructured languages. Venkatakrishnan et al. [16] propose a program transformation for a simple deterministic procedural language that ensures a sound detection of information flows. However, none of those three later works prove that the correction mechanisms of "bad" flows proposed do not create a new covert channel that can reveal secret information — see, e.g., [6]. In fact, those correction mechanisms do create a new covert channel. Theorem 4.2 proves that a correction mechanism of "bad" flows can be based on the dynamic analysis proposed in this paper as the results of the dynamic analysis are the same for every executions started with the same public inputs. More recently, Shroff, Smith, and Thober [14] proposed a dynamic information flow analysis which tracks direct flows and collects indirect flows dynamically. The information collected about indirect flows is transferred from one execution to another using a cache mechanism. After an undetermined number of executions, the analysis will know about all indirect flows in the program and thus will then be sound with regard to the detection of all information flows. This information about indirect flows can be precomputed using a static analysis at the cost of a decrease of precision. Using this approach they are able to handle a language including alias and method calls.

Contrary to common assumption, none of the related works on dynamic information flow analysis known to the author take enough context information into account to detect that the program on page 2 is noninterfering when using it to communicate with users other than sexyPirate. For example, the transformation of Venkatakrishnan et al. [16] updates the security label of c with the security label of the condition on line 3 before executing line 7. Therefore, at line 7, c is always considered as secret even if the user is not communicating with sexyPirate. When executing the assignment of line 5, the program counter of Shroff et al.'s work [14] contains a reference to the program point of line 3 and therefore is added to the set of source of implicit flows to c. Consequently, any complete implicit dependency cache contains a reference to the implicit flow from line 3 to line 7. As the test of line 3 is always executed, Shroff et al.'s work always considers line 7 as displaying secret information. The dynamic analysis proposed in this paper *is* able to detect the noninterfering behavior of the program on page 2 when communicating with someone other than sexyPirate.

6 Conclusion

This article addresses the problem of information flow verification and correction at run time in order to enforce the confidentiality of secret data. The confidentiality property to monitor is expressed using the property of noninterference between secret inputs of the execution and its public outputs. The

language taken into consideration is a sequential language with assignments and conditionals (including loops). The main difference between the monitoring mechanism proposed in this article and the ones of related works lies in the static analysis used to detect implicit indirect flows. The static information flow analyses used by the dynamic analysis proposed in this article are sensitive to the current program state. This allows to increase the precision of the overall dynamic information flow analysis for the detection of implicit and explicit indirect flows. In Sect. 4, the proposed noninterference monitor is proved to be sound both with regard to the detection of information flows and with regard to their correction when necessary.

Benefits of monitoring compared to static analyses. Monitoring an execution has a cost. So, what are the main benefits of noninterference monitoring compared to static analyses? The first concerns the possibility that a monitoring mechanism can be used to change the security policy for each execution. In the majority of cases, running a static analysis before every execution would be more costly than using a monitor. The second reason is that noninterference is a rather strong property. Many programs are rejected by static analyses of noninterference. In such cases it is still possible to use a monitoring mechanism with the possibility that some executions will be altered by the monitoring mechanism. However behavior alteration is an intrinsic feature of any monitoring mechanism. Monitoring noninterference ensures confidentiality while still allowing testing with regard to other specifications using unmonitored executions as perfect oracle — at least as perfect as the original program.

There are two main reasons why it is interesting to use a noninterference monitor on a program rejected by a static analysis. The first one is that a monitoring mechanism may be more precise than static analyses because during execution the monitoring mechanism gets some accurate information about the "path behavior" of the program. As an example, let us consider the following program where h is the only secret input and l the only other input (a public one).

```
1 if ( test1(l) ) then tmp := h else skip end;
2 if ( test2(l) ) then x := tmp else skip end;
3 output x
```

Without information on *test1* and *test2* (and often, even with), a static analysis would conclude that this program is unsafe because the secret input information could be carried to x through *tmp* and then to the output. However, if *test1* and *test2* are such that no value of l makes both predicates true, then any execution of the program is perfectly safe. In that case, the monitoring mechanism would allow any execution of this program. The reason is that, l being a public input, only executions following the same path as the current execution are taken care of by the monitoring mechanism. So, for such configurations where the branching conditions are not influenced by the secret inputs, a monitoring mechanism is at least as precise as any static analysis — and often more precise.

The second reason lies in the granularity of the noninterference property. Static analyses have to take into consideration all possible executions of the program analyzed. This implies that if a single execution is unsafe then the program (thus all its executions) is rejected. Whereas, even if some executions of a program are unsafe, a monitor still allows this program to be used. The unsafe executions, which are not useful, are altered to enforce confidentiality while the safe executions are still usable. For example, the program on page 2 being interfering, any static noninterfering analysis rejects this program. Therefore, users would be advised not to use this program at all. However, using a noninterference monitor, it is possible to safely use this program. When communicating with any user other than sexyPirate, monitored executions of this program have their normal behavior. When communicating with sexyPirate, monitored executions are safely detected as potentially interfering and can therefore be corrected to prevent any secret leakage. Of course, when attempting to communicate with sexyPirate, executions of this program are altered and it is therefore not possible to communicate with sexyPirate. However, this is the desired behavior of a noninterference monitor when confidentiality is more important than the service provided by the program.

Acknowledgments: The author is grateful to Anindya Banerjee, Gérard Boudol, Thomas Jensen, Andreï Sabelfeld and David Schmidt for their helpful feedback on an earlier version of this work.

References

[1] J. Brown and T. F. Knight, Jr. A minimal trusted computing base for dynamically ensuring secure information flow. Technical Report ARIES-TM-015, MIT, Nov. 2001.

[2] E. S. Cohen. Information transmission in computational systems. *ACM SIGOPS Operating Systems Review*, 11(5):133–139, 1977.

[3] J. S. Fenton. Memoryless subsystems. *The Computer Journal*, 17(2): 143–147, 1974. doi: 10.1093/comjnl/17.2.143. URL http://comjnl.oxfordjournals.org/cgi/content/abstract/17/2/143.

[4] D. Goodin. Yahoo feeds Trojan-laced ads to MySpace and PhotoBucket users. The Register, Sept. 2007. http://www.theregister.co.uk/2007/09/11/yahoo_serves_12million_malware_ads/.

[5] G. Le Guernic. Automaton-based Confidentiality Monitoring of Concurrent Programs. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium (CSFS20)*. IEEE Computer Society, July 6–8 2007. ISBN 0-7695-2819-8.

[6] G. Le Guernic and T. Jensen. Monitoring Information Flow. In A. Sabelfeld, editor, *Proceedings of the Workshop on Foundations of Computer Security*, pages 19–30. DePaul University, June 2005.

[7] G. Le Guernic, A. Banerjee, T. Jensen, and D. Schmidt. Automata-based Confidentiality Monitoring. In *Proceedings of the Annual Asian Computing Science Conference*, volume 4435 of *Lecture Notes in Computer Science*, Dec. 6–8 2006.

[8] J. Leyden. Trojan planted on US Consulate website. The Register, Sept. 2007. http://www.theregister.co.uk/2007/09/13/us_consulate_trojan/.

[9] W. Masri, A. Podgurski, and D. Leon. Detecting and debugging insecure information flows. In *Proceedings of the International Symposium on Software Reliability Engineering*, pages 198–209. IEEE, Nov. 2004. doi: http://doi.ieeecomputersociety.org/10.1109/ISSRE.2004.17.

[10] A. C. Myers. JFlow: Practical mostly-static information flow control. In *Proc. ACM Symp. Principles of Programming Languages*, pages 228–241, Jan. 1999.

[11] F. Pottier and S. Conchon. Information flow inference for free. In *Proc. ACM International Conf. on Functional Programming*, pages 46–57. ACM Press, Sept. 2000. ISBN 1-58113-202-6. doi: http://doi.acm.org/10.1145/351240.351245.

[12] F. Pottier and V. Simonet. Information flow inference for ML. *ACM Trans. on Programming Languages and Systems*, 25(1):117–158, 2003. ISSN 0164-0925. doi: http://doi.acm.org/10.1145/596980.596983.

[13] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE J. Selected Areas in Communications*, 21(1):5–19, Jan. 2003.

[14] P. Shroff, S. F. Smith, and M. Thober. Dynamic dependency monitoring to secure information flow. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium (CSFS20)*. IEEE Computer Society, July 6–8 2007. ISBN 0-7695-2819-8. (Best Paper Award).

[15] N. Vachharajani, M. J. Bridges, J. Chang, R. Rangan, G. Ottoni, J. A. Blome, G. A. Reis, M. Vachharajani, and D. I. August. Rifle: An architectural framework for user-centric information-flow security. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, Dec. 2004.

[16] V. N. Venkatakrishnan, W. Xu, D. C. DuVarney, and R. Sekar. Provably correct runtime enforcement of non-interference properties. In P. Ning, S. Qing, and N. Li, editors, *Proceedings of the International Conference on Information and Communications Security*, volume 4307 of *Lecture Notes in Computer Science*, pages 332–351. Springer-Verlag, Dec.

2006. ISBN 978-3-540-49496-6. doi: 10.1007/11935308_24. URL http://www.springerlink.com/content/lmn4768325081h8w.

[17] C. Weissman. Security controls in the adept-50 timesharing system. In *Proc. AFIPS Fall Joint Computer Conf.*, volume 35, pages 119–133, 1969.

[18] J. P. L. Woodward. Exploiting the dual nature of sensitivity labels. In *Proc. IEEE Symp. Security and Privacy*, pages 23–31, Oakland, CA, Apr. 1987.

A Appendix: Proofs

A.1 Static Analysis Properties

Lemma A.1 (Correctness of the static analysis for information flow detection)**.**

For all monitoring execution states $(\sigma,\ \rho)$*,* $(\sigma_i,\ \rho_i)$ *and* $(\sigma_o,\ \rho_o)$*, program counter tags* t^{pc}*, programs* $\mathtt{P}$*, and analysis results* $\mathfrak{X}$ *such that:*

1. $\forall x : (\rho(x) = \bot) \Rightarrow \sigma_i(x) = \sigma(x)$,
2. $(\sigma_i,\ \rho_i),\ t^{pc} \ \vdash\ \mathtt{P} \Downarrow_{\mathcal{M}}\ (\sigma_o,\ \rho_o)$,

the following holds:

$$[\![(\sigma,\ \rho) \vdash \mathtt{P}]\!]^{\sharp_{\mathcal{G}}} = \mathfrak{X} \quad \Rightarrow \quad \forall x \notin \mathfrak{X}.\ \sigma_o(x)\ =\ \sigma_i(x)$$

Proof. This lemma is obvious once we are convinced that the result of the static analysis contains all the variables whose value *may* be modified by any execution of the analyzed program in the context of the analysis (i.e. for any initial state where the value of the variables whose tag is $\bot$ in ρ have the same value has the one in σ). Going by induction on the structure of the analyzed program, it is obvious for skip statements and assignments. When analyzing the second part of a sequence "$\mathtt{P}_1$; $\mathtt{P}_2$", the tags of the variables which *may* have been modified by the execution of $\mathtt{P}_1$ are set to $\top$ to reflect the fact that the analysis does not know anymore the exact value of those variables at the beginning of $\mathtt{P}_2$. Therefore, it is easy to prove by induction that the property holds for sequences. The result of the analysis of an `if`-statement is the result of the analysis of the branch designated by the condition if and only if the tag of this condition is $\bot$ (i.e. any initial state compatible with the context of execution evaluates the condition to the same value as the analysis); otherwise, it is the union of the analysis of both branches. Once again, a simple induction on the structure of the analyzed program proves the desired property for `if`-statements. For `while`-statements, the result of the analysis is empty if and only if the analysis is able to determine that, any execution started in an initial state compatible with the context of the analysis do not execute the body of the loop. Otherwise, the result of the analysis of the `while`-statement relies on fix point computation. It corresponds to the result of the analysis of the body of the loop considering that the value of any variable whose value may be modified by an execution of the body of the loop is unknown at the beginning of the analysis (Thus taking into account any number of execution of the body of the loop). Therefore, an induction on the structure of the `while`-statements using the properties of fix points allow to prove the property for `while`-statements. □

Lemma A.2 (Static analysis is deterministic with regard to public data)**.**
For all monitoring execution states $(\sigma,\ \rho)$ *and* $(\sigma',\ \rho')$*, and programs* $\mathtt{P}$ *such that:*

1. $\rho = \rho'$,

2. $\forall x : (\rho(x) \sqsubseteq \bot) \Rightarrow \sigma(x) = \sigma'(x)$,

the following holds:

$$[\![(\sigma,\ \rho) \vdash P]\!]^{\sharp_{\mathcal{G}}} \ = \ [\![(\sigma',\ \rho') \vdash P]\!]^{\sharp_{\mathcal{G}}}$$

Proof. As the constraints stated in Fig. 3 use only the precise value of variables whose tag is $\bot$ and never reset a tag to $\bot$ for included constraints, this lemma is direct for any deterministic analysis satisfying the constraints of Fig. 3. □

Lemma A.3 (Correctness of the static analysis for information flow correction)**.**

For all monitoring execution states $(\sigma,\ \rho)$ *and* $(\sigma',\ \rho')$*, programs* P*, and analysis results* $\mathfrak{X}$ *such that:*

1. $(\sigma,\ \rho),\ \top\ \vdash\ P\ \Downarrow_{\mathcal{M}}\ (\sigma',\ \rho')$,
2. $[\![(\sigma,\ \rho) \vdash P]\!]^{\sharp_{\mathcal{G}}} = \mathfrak{X}$,

the following holds:

$$\rho' \quad = \quad \rho \quad \sqcup \quad \left(\ (\mathfrak{X} \times \{\top\}) \ \cup \ (\overline{\mathfrak{X}} \times \{\bot\}) \ \right)$$

Proof. This lemma is proved by induction on the structure of `P`. It is obvious for skip statements and assignments. For a sequence "$\mathtt{P}_1$; $\mathtt{P}_2$", the induction on $\mathtt{P}_1$ give that the tag store used for the evaluation of $\mathtt{P}_2$ is the same as the one used for the analysis of $\mathtt{P}_2$. And lemmas A.1 and A.2 give that the value store used for the evaluation of $\mathtt{P}_2$ is equivalent to the initial value store from the point of view of the analysis. Therefore, two simple inductions prove the lemma for sequences. For `if`-statements, if the tag of the condition is $\bot$ then the tag store after evaluation is the tag store after evaluation of the branch designated by the condition and the result of the analysis is the result of the analysis of the branch designated by the condition. Therefore, a simple induction proves the lemma whenever the tag of the condition is $\bot$. If the tag of the condition is $\top$ then the tag store after evaluation is the least upper bound of the tag store after the evaluation of the branch designated by the condition and a tag store equal to $(\mathfrak{X}_v \times \{\top\}) \cup (\overline{\mathfrak{X}_v} \times \{\bot\})$ with $\mathfrak{X}_v$ the result of the analysis of the unexecuted branch. The result of the analysis is the union of the result of the analysis of both branches. Therefore, a simple induction on the branch executed proves the lemma whenever the tag of the condition is $\top$. The evaluation of a `while`-statement always ends up evaluating this `while`-statement with the condition being `false` (at the end of the execution of the `while`-statement). Therefore, at the end of the evaluation, the tag of all the variables appearing in the result of the analysis of the `while`-statement are set to $\top$. For the same reasons as with sequences (lemmas A.1 and A.2) with the addition of the fix point property, the value stores and tag stores at the beginning of the evaluation and at the end of the evaluation of the loop are equivalent from the point of view of the analysis. The analysis of the `while`-statement returns the same result as the

static analysis run by the evaluation and the lemma holds for every variable appearing in the result of the static analysis. All the other variables *can not* be assigned to by the evaluation of the `while`-statement (cause of lemma A.1 which can be extracted from it by observing that the analysis does not compare the value assigned to a variable from its value before the analysis). Therefore, at the end of the evaluation , they have the same tag than at the beginning and the lemma also holds for those variables. □

A.2 Detection Soundness

Lemma A.4 (Public expressions are stable)**.**
For all tag stores ρ, value stores σ_1 and σ_2, and expression e such that for all variable x the following holds $(\rho(x) = \bot) \Rightarrow \sigma_1(x) = \sigma_2(x)$, if $\rho(e) = \bot$ then $\sigma_1(e) = \sigma_2(e)$.

Proof. The proof is straightforward. It follows directly from the facts that expression evaluation is deterministic, the tag of an expression is the least upper bound of the tags of its free variables and that public ($\bot$) variables have the same value in σ_1 and σ_2. □

Lemma A.5 (Tag of assigned variables contains t^{pc})**.**
For all value stores σ_i, tag stores ρ_i, and statement S such that:

- $(\sigma_i, \rho_i),\ t^{pc}\ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$

it is true that:

- $\forall x : t^{pc} \not\sqsubseteq \rho_o(x)\ \Rightarrow\ (\sigma_o(x) = \sigma_i(x) \wedge \rho_i(x) \sqsubseteq \rho_o(x))$

Proof. The proof goes by induction on the derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}}\ \vdash S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ ". Assume the lemma holds for any sub-derivation tree, if the last rule used is:

$(\mathbf{E}_{\mathcal{M}} - \mathbf{IF}_{\bot})$ then we can conclude that :

(1) S is "**if** e **then** S_{true} **else** S_{false} **end**" and " $(\sigma_i, \rho_i),\ t^{\text{pc}} \sqcup \bot\ \vdash\ S_v\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ " is a sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}}\ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ ".
It follows directly from the definition of the rule $(\mathbf{E}_{\mathcal{M}} - \mathbf{IF}_{\bot})$.

(•) $\forall x : t^{\text{pc}} \not\sqsubseteq \rho_o(x) \Rightarrow \sigma_o(x) = \sigma_i(x) \wedge \rho_i(x) \sqsubseteq \rho_o(x)$.
This is obtained by a simple induction because $t^{\text{pc}} \sqcup \bot = t^{\text{pc}}$.

$(\mathbf{E}_{\mathcal{M}} - \mathbf{IF}_{\top})$ then we can conclude that :

(1) S is "**if** e **then** S_1 **else** S_2 **end**" and there exist S_v and ρ_v such that:

- " $(\sigma_i, \rho_i),\ t^{\text{pc}} \sqcup \top\ \vdash\ S_v\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_v)$ " is a sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}}\ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ "
- $\forall x : \rho_v(x) \sqsubseteq \rho_o(x)$

It follows directly from the definition of the rule ($\mathbf{E}_{\mathcal{M}} - \mathbf{IF}_{\top}$).

(2) $\forall x : t^{\mathrm{pc}} \sqcup \top \not\sqsubseteq \rho_v(x) \Rightarrow \sigma_o(x) = \sigma_i(x) \wedge \rho_i(x) \sqsubseteq \rho_v(x)$.
It follows directly from the inductive hypothesis and the local conclusion (1).

(•) $\forall x : t^{\mathrm{pc}} \not\sqsubseteq \rho_o(x) \Rightarrow \sigma_o(x) = \sigma_i(x) \wedge \rho_i(x) \sqsubseteq \rho_o(x)$.
As $t^{\mathrm{pc}} \not\sqsubseteq \rho_o(x)$ implies $t^{\mathrm{pc}} \sqcup \top \sqsubseteq \rho_v(x)$ (from the local conclusion (1)), the above result follows directly from the local conclusions (1) and (2).

($\mathbf{E}_{\mathcal{M}} - \mathbf{WHILE}_{\mathbf{skip}}$) then we can conclude that :

(1) S is "**while** e **do** S_1 **done**" and $\sigma_o = \sigma_i \wedge \rho_i = \rho_o$.
It follows directly from the definition of the rule ($\mathbf{E}_{\mathcal{M}} - \mathbf{WHILE}_{\mathbf{skip}}$).

(•) $\forall x : t^{\mathrm{pc}} \not\sqsubseteq \rho_o(x) \Rightarrow \sigma_o(x) = \sigma_i(x) \wedge \rho_i(x) \sqsubseteq \rho_o(x)$.
It follows directly from the local conclusion (1).

($\mathbf{E}_{\mathcal{M}} - \mathbf{WHILE}_{\mathbf{true}_{\bot}}$) then we can conclude that :

(1) S is "**while** e **do** S_1 **done**" and there exist S', t'_e and ρ' such that:

- " $(\sigma_i, \rho_i),\ t^{\mathrm{pc}} \sqcup \bot \ \vdash\ S' \ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ " is a sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\mathrm{pc}} \ \vdash\ S \ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ "

It follows directly from the definition of the rule ($\mathbf{E}_{\mathcal{M}} - \mathbf{WHILE}_{\mathbf{true}_{\bot}}$).

(•) $\forall x : t^{\mathrm{pc}} \not\sqsubseteq \rho_o(x) \Rightarrow \sigma_o(x) = \sigma_i(x) \wedge \rho_i(x) \sqsubseteq \rho_o(x)$.
It follows directly from the inductive hypothesis and the local conclusion (1).

($\mathbf{E}_{\mathcal{M}} - \mathbf{WHILE}_{\mathbf{true}_{\top}}$) then we can conclude that :

(1) S is "**while** e **do** S_1 **done**" and there exist S' and ρ' such that:

- " $(\sigma_i, \rho_i),\ t^{\mathrm{pc}} \sqcup \top \ \vdash\ S' \ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho')$ " is a sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\mathrm{pc}} \ \vdash\ S \ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ "
- $\forall x : \rho'(x) \sqsubseteq \rho_o(x)$

It follows directly from the definition of the rule ($\mathbf{E}_{\mathcal{M}} - \mathbf{WHILE}_{\mathbf{true}_{\top}}$).

(2) $\forall x : t^{\mathrm{pc}} \sqcup \top \not\sqsubseteq \rho'(x) \Rightarrow \sigma_o(x) = \sigma_i(x) \wedge \rho_i(x) \sqsubseteq \rho'(x)$.
It follows directly from the inductive hypothesis and the local conclusion (1).

(•) $\forall x : t^{\mathrm{pc}} \not\sqsubseteq \rho_o(x) \Rightarrow \sigma_o(x) = \sigma_i(x) \wedge \rho_i(x) \sqsubseteq \rho_o(x)$.
As $t^{\mathrm{pc}} \not\sqsubseteq \rho_o(x)$ and $\rho'(x) \sqsubseteq \rho_o(x)$ imply $t^{\mathrm{pc}} \sqcup t'_e \not\sqsubseteq \rho'(x)$, the above result follows from the local conclusions (1) and (2).

($\mathbf{E}_{\mathcal{M}} - \mathbf{WHILE}_{\mathbf{false}_{\top}}$) then we can conclude that :

(1) S is "**while** e **do** S_1 **done**", $\sigma_o = \sigma_i$ and $\forall x : \rho_i(x) \sqsubseteq \rho_o(x)$.
It follows directly from the definition of the rule ($\mathbf{E}_{\mathcal{M}} - \mathbf{WHILE}_{\mathbf{false}_{\top}}$).

(•) $\forall x : t^{\text{pc}} \not\sqsubseteq \rho_o(x) \Rightarrow \sigma_o(x) = \sigma_i(x) \wedge \rho_i(x) \sqsubseteq \rho_o(x)$.
It follows directly from the local conclusion (1).

($\mathbf{E}_{\mathcal{M}} - \mathbf{SEQUENCE}$) then we can conclude that :

(1) S is "S_1 ; S_2" and there exist σ_1 and ρ_1 such that:

- " $(\sigma_i, \rho_i),\ t^{\text{pc}} \vdash S_1 \Downarrow_{\mathcal{M}} (\sigma_1, \rho_1)$ " is a sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \vdash S \Downarrow_{\mathcal{M}} (\sigma_o, \rho_o)$ "
- " $(\sigma_1, \rho_1),\ t^{\text{pc}} \vdash S_2 \Downarrow_{\mathcal{M}} (\sigma_o, \rho_o)$ " is a sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \vdash S \Downarrow_{\mathcal{M}} (\sigma_o, \rho_o)$ "

It follows directly from the definition of the rule ($\mathbf{E}_{\mathcal{M}} - \mathbf{SEQUENCE}$).

(2) $\forall x : t^{\text{pc}} \not\sqsubseteq \rho_1(x) \Rightarrow \sigma_1(x) = \sigma_i(x) \wedge \rho_i(x) \sqsubseteq \rho_1(x)$ and $\forall x : t^{\text{pc}} \not\sqsubseteq \rho_o(x) \Rightarrow \sigma_o(x) = \sigma_1(x) \wedge \rho_1(x) \sqsubseteq \rho_o(x)$.
It follows directly from the inductive hypothesis and the local conclusion (1).

(•) $\forall x : t^{\text{pc}} \not\sqsubseteq \rho_o(x) \Rightarrow \sigma_o(x) = \sigma_i(x) \wedge \rho_i(x) \sqsubseteq \rho_o(x)$.
From the local conclusion (1), if $t^{\text{pc}} \not\sqsubseteq \rho_o(x)$ then $\rho_1(x) \sqsubseteq \rho_o(x)$. Hence, $t^{\text{pc}} \not\sqsubseteq \rho_1(x)$. Then, combining the results of the local conclusion (2), "$\sigma_o(x) = \sigma_1(x) = \sigma_i(x) \wedge \rho_i(x) \sqsubseteq \rho_1(x) \sqsubseteq \rho_o(x)$"

($\mathbf{E}_{\mathcal{M}} - \mathbf{ASSIGN}$) then we can conclude that :

(1) S is "id **:=** e" and there exist v_e, and t_e such that $\sigma_o = \sigma_i[id \mapsto v_e]$ and $\rho_o = \rho_i[id \mapsto t_e \sqcup t^{\text{pc}}]$.
It follows directly from the definition of the rule ($\mathbf{E}_{\mathcal{M}} - \mathbf{ASSIGN}$).

(•) $\forall x : t^{\text{pc}} \not\sqsubseteq \rho_o(x) \Rightarrow \sigma_o(x) = \sigma_i(x) \wedge \rho_i(x) \sqsubseteq \rho_o(x)$.
From (1), if $x = id$ then $t^{\text{pc}} \sqsubseteq \rho_o(x)$. Otherwise, $x \neq id$ and $\sigma_o(x) = \sigma_i(x) \wedge \rho_i(x) = \rho_o(x)$

($\mathbf{E}_{\mathcal{M}} - \mathbf{SKIP}$) then we can conclude that :

(1) S is "**skip**" and $\sigma_o = \sigma_i \wedge \rho_i = \rho_o$.
It follows directly from the definition of the rule ($\mathbf{E}_{\mathcal{M}} - \mathbf{SKIP}$).

(•) $\forall x : t^{\text{pc}} \not\sqsubseteq \rho_o(x) \Rightarrow \sigma_o(x) = \sigma_i(x) \wedge \rho_i(x) \sqsubseteq \rho_o(x)$.
It follows directly from the local conclusion (1).

□

Lemma A.6 (Correctness for information flow detection with semantics $\Downarrow_{\mathcal{M}}$)**.**

For all:

- *variable* x,
- *value stores* σ_i, σ_o, σ_i', *and* σ_o',

- *tag stores* ρ_i, ρ_o, ρ'_i, *and* ρ'_o,
- *tags* t^{pc} *and* $t^{pc'}$,
- *and statement* S

such that:

$\star_1$ $\forall y : (\rho_i(y) = \bot) \Rightarrow \sigma_i(y) = \sigma'_i(y)$,

$\star_2$ $(\sigma_i, \rho_i),\ t^{pc} \vdash S \Downarrow_{\mathcal{M}} (\sigma_o, \rho_o)$,

$\star_3$ $(\sigma'_i, \rho'_i),\ t^{pc'} \vdash S \Downarrow_{\mathcal{M}} (\sigma'_o, \rho'_o)$,

$\star_4$ $\rho_o(x) = \bot$

it is true that:

- $\sigma_o(x) = \sigma'_o(x)$.

Proof. The proof goes by induction on the derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \vdash S \Downarrow_{\mathcal{M}} (\sigma_o, \rho_o)$ ". Assume the lemma holds for any sub-derivation tree, if the last rule used is:

$(\mathbf{E-IF}_{\bot})$ then we can conclude that :

(1) S is "**if** e **then** S_{true} **else** S_{false} **end**" and there exists $v \in \{true, false\}$ such that:

- $\sigma_i(e) = v$ and $\rho_i(e) = \bot$
- " $(\sigma_i, \rho_i),\ t^{\text{pc}} \sqcup \bot \vdash S_v \Downarrow_{\mathcal{M}} (\sigma_o, \rho_o)$ " is a sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \vdash S \Downarrow_{\mathcal{M}} (\sigma_o, \rho_o)$ "

It follows directly from the definition of the rule $(\mathbf{E-IF}_{\bot})$.

(2) There exist a value v', a tag t', and a tag store ρ' such that:

- $\sigma'_i(e) = v'$ and $\rho'_i(e) = t'$
- " $(\sigma'_i, \rho'_i),\ t^{\text{pc}'} \sqcup t' \vdash S_{v'} \Downarrow_{\mathcal{M}} (\sigma'_o, \rho')$ "

It follows from the global hypothesis $\star_5$, the local conclusion (1), and the definitions of the rules applying to "**if** e **then** S_{true} **else** S_{false} **end**" ($(\mathbf{E-IF}_{\bot})$ and $(\mathbf{E-IF}_{\top})$).

(3) $v = v'$.
This result comes from lemma A.4, the global hypothesis $\star_1$ and local conclusions (1) and (2).

(•) $\sigma_o(x) = \sigma'_o(x)$.
The conclusion is obtained by applying the inductive hypothesis on " $(\sigma_i, \rho_i),\ t^{\text{pc}} \sqcup \bot \vdash S_v \Downarrow_{\mathcal{M}} (\sigma_o, \rho_o)$ " (sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \vdash S \Downarrow_{\mathcal{M}} (\sigma_o, \rho_o)$ ") and " $(\sigma'_i, \rho'_i),\ t^{\text{pc}'} \sqcup t' \vdash S_v \Downarrow_{\mathcal{M}} (\sigma'_o, \rho')$ " (because $v = v'$).

$(\mathbf{E-IF}_{\top})$ then we can conclude that :

(1) S is "**if** e **then** S_{true} **else** S_{false} **end**" and there exist $v \in \{true, false\}$, two tag store ρ_v and ρ_e, and an analysis result $\mathfrak{X}$ such that:

- " $(\sigma_i, \rho_i),\ t^{\text{pc}} \sqcup \top \ \vdash\ S_v\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_v)$ " is a sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ "
- $[\![\sigma_i; \rho_i \vdash S_{\neg v}]\!]^{\sharp_{\mathcal{G}}} = \mathfrak{X}$
- $\rho_v(x) \sqsubseteq \rho_o(x)$
- $\rho_e = (\mathfrak{X} \times \{\top\}) \cup (\overline{\mathfrak{X}} \times \{\bot\})$
- $\rho_e(x) \sqsubseteq \rho_o(x)$

It follows directly from the definition of the rule $(\mathbf{E-IF}_{\bot})$.

(2) There exist a value $v' \in \{true, false\}$, a tag store $\rho'_{v'}$, and a tag t'_e such that:

- " $(\sigma'_i, \rho'_i),\ t^{\text{pc}'} \sqcup t'_e \ \vdash\ S_{v'}\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_{v'})$ "

It follows from the global hypothesis $\star_5$, the local conclusion (1), and the definitions of the rules applying to "**if** e **then** S_{true} **else** S_{false} **end**" ($(\mathbf{E-IF}_{\bot})$ and $(\mathbf{E-IF}_{\top})$).

Case 1: $v = v'$

(a) $\rho_v(x) = \bot$.
It follows from the local conclusion (1) and the global hypothesis $\star_4$.

(•) $\sigma_o(x) = \sigma'_o(x)$.
The conclusion is obtained by applying the inductive hypothesis on " $(\sigma_i, \rho_i),\ t^{\text{pc}} \sqcup \top \ \vdash\ S_v\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_v)$ " (sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ ") and " $(\sigma'_i, \rho'_i),\ t^{\text{pc}'} \sqcup t'_e \ \vdash\ S_{v'}\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_{v'})$ ". It is possible to apply the inductive hypothesis because of the local conclusion (a) and the fact that, by the case hypothesis, $v = v'$).

Case 2: $v \neq v'$ (which implies that $v' = \neg v$)

(a) $\sigma_o(x) = \sigma_i(x) \wedge \rho_i(x) \sqsubseteq \rho_o(x)$.
From the global hypothesis $\star_4$, $\rho_o(x) = \bot$. Hence, from lemma A.5 applied to " $(\sigma_i, \rho_i),\ t^{\text{pc}} \sqcup \top \ \vdash\ S_v\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_v)$ ", $\sigma_o(x) = \sigma_i(x) \wedge \rho_i(x) \sqsubseteq \rho_o(x)$.

(b) $\sigma'_o(x) = \sigma'_i(x)$.
From the local conclusion (1), the global hypothesis $\star_4$, and the definition of ρ_e in the local conclusion (1), $x \notin \mathfrak{X}$. Hence, from the local conclusion (1) and the lemma A.1 applied to " $(\sigma'_i, \rho'_i),\ t^{\text{pc}'} \sqcup t'_e \ \vdash\ S_{v'}\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_{v'})$ ", $\sigma'_o(x) = \sigma'_i(x)$.

(c) $\sigma_i(x) = \sigma'_i(x)$.
From the local conclusion (a) and the global hypothesis $\star_4$, we get that $\rho_i(x) = \bot$. Hence, from the global hypothesis $\star_1$, we get that $\sigma_i(x) = \sigma'_i(x)$.

(•) $\sigma_o(x) = \sigma'_o(x)$.
It follows directly from the local conclusions (a), (b), and (c).

($\mathbf{E-WHILE_{skip}}$) then we can conclude that :

(1) S is "**while** e **do** S_1 **done**" and:

- $\sigma_i(e) = \texttt{false}$ and $\rho_i(e) = \bot$
- $\sigma_i = \sigma_o$ and $\rho_i = \rho_o$

It follows directly from the definition of the rule ($\mathbf{E-IF_{\bot}}$).

(2) There exist a value v' and a tag t' such that:

- $\sigma'_i(e) = v'$ and $\rho'_i(e) = t'$

It follows from the global hypothesis $\star_5$, the local conclusion (1), and the definitions of the rules applying to "**while** e **do** S_1 **done**" (($\mathbf{E-WHILE_{skip}}$), ($\mathbf{E-WHILE_{true_{\bot}}}$), ($\mathbf{E-WHILE_{true_{\top}}}$) and ($\mathbf{E-WHILE_{false_{\top}}}$)).

(3) $\sigma'_i = \sigma'_o$.
From lemma A.4, the global hypothesis $\star_1$ and the local conclusions (1) and (2), we get $v' = \mathit{false}$. Hence, from the global hypothesis $\star_5$, the local conclusion (1), and the definitions of the rules applying to "**while** e **do** S_1 **done**" whenever e is $\texttt{false}$ (($\mathbf{E-WHILE_{skip}}$) and ($\mathbf{E-WHILE_{false_{\top}}}$)), $\sigma'_i = \sigma'_o$.

(4) $\sigma_i(x) = \sigma'_i(x)$.
From the local conclusion (1) and the global hypothesis $\star_4$, we get that $\rho_i(x) = \bot$. Hence, from the global hypothesis $\star_1$, we get that $\sigma_i(x) = \sigma'_i(x)$.

(•) $\sigma_o(x) = \sigma'_o(x)$.
It follows directly from the local conclusions (1), (3), and (4).

($\mathbf{E-WHILE_{true_{\bot}}}$) then we can conclude that :

(1) S is "**while** e **do** S_1 **done**" and:

- $\sigma_i(e) = \texttt{true}$ and $\rho_i(e) = \bot$
- " $(\sigma_i, \rho_i),\ t^{\mathrm{pc}} \sqcup \bot \ \vdash\ S_1\ ;\ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ " is a sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\mathrm{pc}}\ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ "

It follows directly from the definition of the rule ($\mathbf{E-WHILE_{true_{\bot}}}$).

(2) There exist a value v' and a tag t'_e such that:

- $\sigma'_i(e) = v'$ and $\rho'_i(e) = t'_e$

It follows from the global hypothesis $\star_5$, the local conclusion (1), and the definitions of the rules applying to "**while** e **do** S_1 **done**" (($\mathbf{E-WHILE_{skip}}$), ($\mathbf{E-WHILE_{true_{\bot}}}$), ($\mathbf{E-WHILE_{true_{\top}}}$) and ($\mathbf{E-WHILE_{false_{\top}}}$)).

(3) $v' = \texttt{true}$.
It follows directly from the local conclusions (1) and (2), lemma A.4 and global hypothesis $\star_1$.

(4) There exists a tag store ρ'_l such that:

- " $(\sigma'_i, \rho'_i),\ t^{\mathrm{pc}'} \sqcup t'_e \ \vdash\ S_1\ ;\ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_l)$ "

It follows from the global hypothesis $\star_5$, the local conclusions (1), (2) and (3), and the definitions of the rules applying to "**while** e **do** S_1 **done**" whenever e is `true` ($(\mathbf{E-WHILE_{true_\perp}})$ and $(\mathbf{E-WHILE_{true_\top}})$).

(•) $\sigma_o(x) = \sigma'_o(x)$.
By applying the inductive hypothesis on the derivations " $(\sigma_i, \rho_i),\ t^{\mathrm{pc}} \sqcup \bot\ \vdash\ S_1\ ;\ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ " and " $(\sigma'_i, \rho'_i),\ t^{\mathrm{pc}'} \sqcup t'_e\ \vdash\ S_1\ ;\ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_l)$ ", it is possible to deduce that $\sigma_o(x) = \sigma'_o(x)$.

$(\mathbf{E-WHILE_{true_\top}})$ then we can conclude that :

(1) S is "**while** e **do** S_1 **done**" and there exist a tag t_e and a tag store ρ_l such that:

- $\sigma_i(e) = \mathtt{true}$ and $\rho_i(e) = \top$
- " $(\sigma_i, \rho_i),\ t^{\mathrm{pc}} \sqcup \top\ \vdash\ S_1\ ;\ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_l)$ " is a sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\mathrm{pc}}\ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ "
- $\rho_l \sqsubseteq \rho_o$

It follows directly from the definition of the rule $(\mathbf{E-WHILE_{true_\top}})$.

(2) $\rho_l(x) = \bot$.
It follows from the local conclusion (1) and the global hypothesis $\star_4$.

(3) There exist a value v' and a tag t'_e such that:

- $\sigma'_i(e) = v'$ and $\rho'_i(e) = t'_e$

It follows from the global hypothesis $\star_5$, the local conclusion (1), and the definitions of the rules applying to "**while** e **do** S_1 **done**" ($(\mathbf{E-WHILE_{skip}})$, $(\mathbf{E-WHILE_{true_\bot}})$, $(\mathbf{E-WHILE_{true_\top}})$ and $(\mathbf{E-WHILE_{false_\top}})$).

Case 1: $v' = \mathtt{true}$

(a) There exists a tag store ρ'_l such that:

- " $(\sigma'_i, \rho'_i),\ t^{\mathrm{pc}'} \sqcup t'_e\ \vdash\ S_1\ ;\ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_l)$ "

It follows from the global hypothesis $\star_5$, the local conclusions (1) and (3), the case hypothesis, and the definitions of the rules applying to "**while** e **do** S_1 **done**" whenever e is `true` ($(\mathbf{E-WHILE_{true_\bot}})$ and $(\mathbf{E-WHILE_{true_\top}})$).

(•) $\sigma_o(x) = \sigma'_o(x)$.
By applying the inductive hypothesis on the derivations " $(\sigma_i, \rho_i),\ t^{\mathrm{pc}} \sqcup \top\ \vdash\ S_1\ ;\ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_l)$ " and " $(\sigma'_i, \rho'_i),\ t^{\mathrm{pc}'} \sqcup t'_e\ \vdash\ S_1\ ;\ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_l)$ ", it is possible to deduce, using the local conclusion (2), that $\sigma_o(x) = \sigma'_o(x)$.

Case 2: $v' = \mathtt{false}$

(a) $\sigma_o(x) = \sigma_i(x)$.
It follows directly from the local conclusions (1), the fact that "$t^{\text{pc}} \sqcup \top \not\sqsubseteq \rho_l(x)$" (from the local conclusion (2)), and the lemma A.5 applied to " $(\sigma_i, \rho_i),\ t^{\text{pc}} \sqcup \top \vdash S_1$; **while** e **do** S_1 **done** $\Downarrow_{\mathcal{M}}$ (σ_o, ρ_l) ".

(b) $\sigma'_o(x) = \sigma'_i(x)$.
From the global hypothesis $\star_5$, the local conclusions (1) and (3), the case hypothesis, and the definitions of the rules applying to "**while** e **do** S_1 **done**" whenever e is `false` ($(\mathbf{E-WHILE_{skip}})$ and $(\mathbf{E-WHILE_{false_\top}})$), $\sigma'_i = \sigma'_o$.

(c) $\sigma_i(x) = \sigma'_i(x)$.
From the local conclusion (2) and lemma A.5 applied to " $(\sigma_i, \rho_i),\ t^{\text{pc}} \sqcup \top \vdash S_1$; **while** e **do** S_1 **done** $\Downarrow_{\mathcal{M}}$ (σ_o, ρ_l) ", $\rho_i(x) = \bot$. Hence, from the global hypothesis $\star_1$, we get that $\sigma_i(x) = \sigma'_i(x)$.

(•) $\sigma_o(x) = \sigma'_o(x)$.
It follows directly from the local conclusions (a), (b), and (c).

$(\mathbf{E-WHILE_{false_\top}})$ then we can conclude that :

(1) S is "**while** e **do** S_1 **done**" and there exist two tag stores ρ_1 and ρ_e such that:

- $\sigma_i(e) = \texttt{false}$ and $\rho_i(e) = \top$
- $[\![\sigma_i; \rho_i \vdash S_1$; **while** e **do** S_1 **done**$]\!]^{\sharp_{\mathcal{G}}} = \mathfrak{X}$
- $\sigma_o = \sigma_i$
- $\rho_e = (\mathfrak{X} \times \{\top\}) \cup (\overline{\mathfrak{X}} \times \{\bot\})$
- $\rho_e(x) \sqsubseteq \rho_o(x)$ and $\rho_i(x) \sqsubseteq \rho_o(x)$

It follows directly from the definition of the rule $(\mathbf{E-WHILE_{true}})$.

(2) There exist a value $v' \in \{\texttt{true}, \texttt{false}\}$ and a tag t'_e such that:

- $\sigma'_i(e) = v'$ and $\rho'_i(e) = t'_e$

It follows from the global hypothesis $\star_5$, the local conclusion (1), and the definitions of the rules applying to "**while** e **do** S_1 **done**" ($(\mathbf{E-WHILE_{skip}})$, $(\mathbf{E-WHILE_{true}})$ and $(\mathbf{E-WHILE_{false_\top}})$).

(3) $\sigma_i(x) = \sigma'_i(x)$.
From the local conclusion (1) ($\rho_i(x) \sqsubseteq \rho_o(x)$) and the global hypothesis $\star_4$, $\rho_i(x) = \bot$. Hence, from the global hypothesis $\star_1$, we get that $\sigma_i(x) = \sigma'_i(x)$.

Case 1: $v' = \texttt{false}$

(a) $\sigma'_o = \sigma'_i$.
From the global hypothesis $\star_5$, the local conclusions (1) and (2), the case hypothesis, and the definitions of the rules applying to "**while** e **do** S_1 **done**" whenever e evaluates to `false` ($(\mathbf{E-WHILE_{skip}})$ and $(\mathbf{E-WHILE_{false_\top}})$), $\sigma'_i = \sigma'_o$.

(•) $\sigma_o(x) = \sigma'_o(x)$.
It follows directly from the local conclusions (1) ($\sigma_o = \sigma_i$), (3), and (a).

Case 2: $v' = \texttt{true}$

(a) $(\sigma'_i, \rho'_i),\ t^{\text{pc}'} \sqcup t'_e \ \vdash\ S_1\ ;\ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_o)$
It follows from the global hypothesis $\star_5$, the local conclusions (1) and (2), the case hypothesis, and the definition of the rule applying to "**while** e **do** S_1 **done**" whenever e evaluates to `true`.

(b) $x \notin \mathfrak{X}$.
From the local conclusion (1) ($\rho_e(x) \sqsubseteq \rho_o(x)$) and the global hypothesis $\star_4$, we get that $\rho_e(x) = \bot$. Hence, from the definition of ρ_e given in the local conclusion (1), $x \notin \mathfrak{X}$.

(c) $\sigma'_o(x) = \sigma'_i(x)$.
From the local conclusion (b), the lemma A.1 applied to the analysis from the local conclusion (1) and the derivation from the local conclusion (a), $\sigma'_o(x) = \sigma'_i(x)$.

(•) $\sigma_o(x) = \sigma'_o(x)$.
It follows directly from the local conclusions (1), (3), and (c).

(**E** − **SEQUENCE**) then we can conclude that :

(1) S is "$S_1\ ;\ S_2$" and there exist σ_1 and ρ_1 such that:

- " $(\sigma_i, \rho_i),\ t^{\text{pc}} \ \vdash\ S_1\ \Downarrow_{\mathcal{M}}\ (\sigma_1, \rho_1)$ " is a sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ "
- " $(\sigma_1, \rho_1),\ t^{\text{pc}} \ \vdash\ S_2\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ " is a sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ "

It follows directly from the definition of the rule ($\mathbf{E}_{\mathcal{M}}$ − **SEQUENCE**).

(2) There exist σ'_1 and ρ'_1 such that:

- " $(\sigma'_i, \rho'_i),\ t^{\text{pc}'} \ \vdash\ S_1\ \Downarrow_{\mathcal{M}}\ (\sigma'_1, \rho'_1)$ "
- " $(\sigma'_1, \rho'_1),\ t^{\text{pc}'} \ \vdash\ S_2\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_o)$ "

It follows directly from the global hypothesis $\star_5$, the local conclusion (1), and the definition of the only rule applying to "$S_1\ ;\ S_2$" (($\mathbf{E}_{\mathcal{M}}$ − **SEQUENCE**)).

(3) $\forall y : (\rho_1(y) = \bot) \Rightarrow \sigma_1(y) = \sigma'_1(y)$.
This result is obtained by applying the inductive hypothesis on " $(\sigma_i, \rho_i),\ t^{\text{pc}} \ \vdash\ S_1\ \Downarrow_{\mathcal{M}}\ (\sigma_1, \rho_1)$ " (sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ ") and " $(\sigma'_i, \rho'_i),\ t^{\text{pc}'} \ \vdash\ S_1\ \Downarrow_{\mathcal{M}}\ (\sigma'_1, \rho'_1)$ " for any variable y such that $\rho_1(y) = \bot$.

(•) $\sigma_o(x) = \sigma'_o(x)$.
The conclusion is obtained by applying the inductive hypothesis on " $(\sigma_1, \rho_1),\ t^{\text{pc}} \ \vdash\ S_2\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ " (sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ ") and " $(\sigma'_1, \rho'_1),\ t^{\text{pc}'} \ \vdash\ S_2\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_o)$ " using the local conclusion (3).

($\mathbf{E-ASSIGN}$) then we can conclude that :

(1) S is "id **:=** e".

Case 1: $x \neq id$

(a) $\sigma_o(x) = \sigma_i(x)$ and $\rho_o(x) = \rho_i(x)$.
It follows directly from the case hypothesis and the definition of the rule ($\mathbf{E-ASSIGN}$).

(b) $\sigma'_o(x) = \sigma'_i(x)$.
It follows directly from the global hypothesis $\star_5$, the local conclusion (1), the definition of the only rule applying to "id **:=** e" (($\mathbf{E-ASSIGN}$)), and the case hypothesis.

(•) $\sigma_o(x) = \sigma'_o(x)$.
As $\rho_o(x) = \bot$, from the global hypothesis $\star_4$, the local conclusion (a) implies $\rho_i(x) = \bot$. Then, from the global hypothesis $\star_1$, $\sigma_i(x) = \sigma'_i(x)$. Hence, from the local conclusions (a) and (b), $\sigma_o(x) = \sigma'_o(x)$.

Case 2: $x = id$

(a) There exist v and t_e such that:
- $\sigma_i(e) = v$ and $\rho_i(e) = t_e$
- $\sigma_o(x) = v$
- $t_e \sqsubseteq \rho_o(x)$

It follows directly from the case hypothesis and the definition of the rule ($\mathbf{E-ASSIGN}$).

(b) There exist v' and t'_e such that:
- $\sigma'_i(e) = v'$ and $\rho'_i(e) = t'_e$
- $\sigma'_o(x) = v'$

It follows directly from the global hypothesis $\star_5$, the local conclusion (1), the definition of the only rule applying to "id **:=** e" (($\mathbf{E-ASSIGN}$)), and the case hypothesis.

(c) $v = v'$.
This result comes from lemma A.4 applied to the local conclusions (a) and (b). It is possible to apply it because of the global hypothesis $\star_1$ and the fact that, from the global hypothesis $\star_4$ and the local conclusion (a), $t_e = \bot$.

(•) $\sigma_o(x) = \sigma'_o(x)$.
It follows directly from the local conclusions (a), (b), and (c).

($\mathbf{E-SKIP}$) then we can conclude that :

(1) S is "**skip**", $\sigma_o = \sigma_i$ and $\rho_o = \rho_i$.
It follows directly from the definition of the rule ($\mathbf{E-SKIP}$).

(2) $\sigma'_o = \sigma'_i$.
It follows directly from the global hypothesis $\star_5$, the local conclusion (1), and the definition of the only rule applying to "**skip**" (($\mathbf{E-SKIP}$)).

(•) $\sigma_o(x) = \sigma'_o(x)$.
From the local conclusion (1) and the global hypothesis $\star_4$, $\rho_i(x) = \bot$. Then, from the global hypothesis $\star_1$, $\sigma_i(x) = \sigma'_i(x)$. Hence, from the local conclusions (1) and (2), $\sigma_o(x) = \sigma'_o(x)$.

□

A.3 Correction Soundness

Lemma A.7 (Expression tag is deterministic)**.**
For all expressions e and tag stores ρ and ρ', if $\rho = \rho'$ then $\rho(e) = \rho'(e)$.

Proof. The proof is straightforward. It follows directly from the facts that the evaluation of an expression's tag is deterministic. It is the least upper bound of the tags of its free variables. □

Lemma A.8 (Correctness for information flow correction with semantics $\Downarrow_{\mathcal{M}}$)**.**

For all:

- *variable x,*
- *value stores σ_i, σ_o, σ'_i, and σ'_o,*
- *tag stores ρ_i, ρ_o, ρ'_i, and ρ'_o,*
- *tags t^{pc} and $t^{pc'}$,*
- *and statement S*

such that:

$\star_1$ $\forall y : (\rho_i(y) = \bot) \Rightarrow \sigma_i(y) = \sigma'_i(y)$,

$\star_2$ $\rho_i = \rho'_i$,

$\star_3$ $t^{pc} = t^{pc'}$,

$\star_4$ $(\sigma_i, \rho_i),\ t^{pc} \vdash S \Downarrow_{\mathcal{M}} (\sigma_o, \rho_o)$,

$\star_5$ $(\sigma'_i, \rho'_i),\ t^{pc'} \vdash S \Downarrow_{\mathcal{M}} (\sigma'_o, \rho'_o)$

it is true that:

- $\rho_o(x) = \rho'_o(x)$.

Proof. The proof goes by induction on the derivation tree of " $(\sigma_i, \rho_i),\ t^{\mathrm{pc}} \vdash S \Downarrow_{\mathcal{M}} (\sigma_o, \rho_o)$ ". Assume the lemma holds for any sub-derivation tree, if the last rule used is:

$(\mathbf{E-IF}_{\bot})$ then we can conclude that :

(1) S is "**if** e **then** S_{true} **else** S_{false} **end**" and there exists $v \in \{true, false\}$ such that:

- $\sigma_i(e) = v$ and $\rho_i(e) = \bot$
- " $(\sigma_i, \rho_i),\ t^{\text{pc}} \sqcup \bot \ \vdash\ S_v\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ " is a sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ "

It follows directly from the definition of the rule $(\mathbf{E-IF}_{\bot})$.

(2) There exist a value v' and a tag t' such that:

- $\sigma'_i(e) = v'$ and $\rho'_i(e) = t'$

It follows from the global hypothesis $\star_5$, the local conclusion (1), and the definitions of the rules applying to "**if** e **then** S_{true} **else** S_{false} **end**" $((\mathbf{E-IF}_{\bot})$ and $(\mathbf{E-IF}_{\top}))$.

(3) $t' = \bot$ and $v = v'$.
It follows directly from the local conclusion (1) and (2), lemmas A.7 and A.4 and the global hypotheses $\star_1$ and $\star_2$.

(4) " $(\sigma'_i, \rho'_i),\ t^{\text{pc}'} \sqcup \bot \ \vdash\ S_v\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_o)$ " It follows from the global hypothesis $\star_5$, the local conclusions (1), (2), and (3), and the definition of the rule applying to "**if** e **then** S_{true} **else** S_{false} **end**" whenever the tag of e is $\bot$ $((\mathbf{E-IF}_{\bot}))$.

(•) $\rho_o(x) = \rho'_o(x)$.
The conclusion is obtained by applying the inductive hypothesis on " $(\sigma_i, \rho_i),\ t^{\text{pc}} \sqcup \bot \ \vdash\ S_v\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ " (sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ ") and " $(\sigma'_i, \rho'_i),\ t^{\text{pc}'} \sqcup \bot \ \vdash\ S_v\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_o)$ ".

$(\mathbf{E-IF}_{\top})$ then we can conclude that :

(1) S is "**if** e **then** S_{true} **else** S_{false} **end**" and there exist $v \in \{true, false\}$, two tag stores ρ_v and ρ_e, and an analysis result $\mathfrak{X}$ such that:

- $\sigma_i(e) = v$ and $\rho_i(e) = \top$
- " $(\sigma_i, \rho_i),\ t^{\text{pc}} \sqcup \top \ \vdash\ S_v\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_v)$ " is a sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ "
- $[\![(\sigma_i, \rho_i) \vdash S_{\neg v}]\!]^{\sharp_{\mathcal{G}}} = \mathfrak{X}$
- $\rho_e = (\mathfrak{X} \times \{\top\}) \cup (\overline{\mathfrak{X}} \times \{\bot\})$
- $\rho_o = \rho_v \sqcup \rho_e$

It follows directly from the definition of the rule $(\mathbf{E-IF}_{\top})$.

(2) There exist a value $v' \in \{true, false\}$ and a tag t'_e such that:

- $\sigma'_i(e) = v'$ and $\rho'_i(e) = t'_e$

It follows from the global hypothesis $\star_5$, the local conclusion (1), and the definitions of the rules applying to "**if** e **then** S_{true} **else** S_{false} **end**" $((\mathbf{E-IF}_{\bot})$ and $(\mathbf{E-IF}_{\top}))$.

(3) $t'_e = \top$.
It follows directly from the local conclusions (1) and (2), the lemma A.7 and the global hypothesis $\star_2$.

(4) There exists an analysis result $\mathfrak{X}'$ and two tag stores $\rho'_{v'}$ and ρ'_e such that:

- " $(\sigma'_i, \rho'_i),\ t^{\mathrm{pc}'} \sqcup t'_e \ \vdash\ S_{v'}\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_{v'})$ "
- $[\![(\sigma'_i, \rho'_i) \vdash S_{\neg v'}]\!]^{\sharp_{\mathcal{G}}} = \mathfrak{X}'$
- $\rho'_e = (\mathfrak{X}' \times \{\top\}) \cup (\overline{\mathfrak{X}'} \times \{\bot\})$
- $\rho'_o = \rho'_{v'} \sqcup \rho'_e$

It follows from the global hypothesis $\star_5$, the local conclusions (1), (2), and (3), and the definition of the rule applying to "**if** e **then** S_{true} **else** S_{false} **end**" whenever the tag of e is $\top$ ($(\mathbf{E-IF}_\top)$).

Case 1: $v = v'$

(a) $\rho_e(x) = \rho'_e(x)$.
It follows from the definitions of ρ_e and ρ'_e (given in the local conclusions (1) and (2)), the case hypothesis, the global hypothesis $\star_2$, and the lemma A.2.

(b) $\rho_v(x) = \rho'_{v'}(x)$.
This result is obtained by applying the inductive hypothesis on " $(\sigma_i, \rho_i),\ t^{\mathrm{pc}} \sqcup \top\ \vdash\ S_v\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_v)$ " (sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\mathrm{pc}}\ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ ") and " $(\sigma'_i, \rho'_i),\ t^{\mathrm{pc}'} \sqcup t'_e\ \vdash\ S_{v'}\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_{v'})$ " using the result of the local conclusion (3) and the case hypothesis.

(•) $\rho_o(x) = \rho'_o(x)$.
It follows directly from the definitions of ρ_o and ρ'_o (given in the local conclusions (1) and (4)) and the local conclusions (a) and (b).

Case 2: $v \neq v'$ (which implies that $v' = \neg v$)

(a) $\rho_v = \rho_i \sqcup \rho'_e$.
It follows from the definitions of ρ_v and ρ'_e (given in the local conclusions (1) and (4)), the case hypothesis, and the hypotheses A.2 and A.3.

(b) $\rho'_{v'} = \rho'_i \sqcup \rho_e$.
It follows from the definitions of $\rho'_{v'}$ and ρ_e (given in the local conclusions (1) and (4)), the case hypothesis, the local conclusion (3) and the hypotheses A.2 and A.3.

(•) $\rho_o(x) = \rho'_o(x)$.
It follows directly from the definitions of ρ_o and ρ'_o (given in the local conclusions (1) and (4)), the global hypothesis $\star_2$ and the local conclusions (a) and (b).

($\mathbf{E-WHILE_{skip}}$) then we can conclude that :

(1) S is "**while** e **do** S_1 **done**" and:

- $\sigma_i(e) = \texttt{false}$ and $\rho_i(e) = \bot$
- $\rho_i = \rho_o$

It follows directly from the definition of the rule $(\mathbf{E-IF}_{\bot})$.

(2) There exist a value v' and a tag t' such that:

- $\sigma'_i(e) = v'$ and $\rho'_i(e) = t'$

It follows from the global hypothesis $\star_5$, the local conclusion (1), and the definitions of the rules applying to "**while** e **do** S_1 **done**" ($(\mathbf{E-WHILE_{skip}})$, $(\mathbf{E-WHILE_{true_{\bot}}})$, $(\mathbf{E-WHILE_{true_{\top}}})$ and $(\mathbf{E-WHILE_{false_{\top}}})$).

(3) $v' = \texttt{false}$ and $t' = \bot$.
It follows directly from the local conclusions (1) and (2), lemmas A.4 and A.7, and the global hypotheses $\star_1$ and $\star_2$.

(4) $\rho'_o = \rho'_i$.
It follows from the global hypothesis $\star_5$, the local conclusion (1), and the definitions of the rules applying to "**while** e **do** S_1 **done**" whenever e evaluates to $\texttt{false}$ and its tag is $\bot$ $(\mathbf{E-WHILE_{skip}})$.

(5) $\rho_i(x) = \rho'_i(x)$.
It follows directly from the global hypothesis $\star_2$.

(•) $\rho_o(x) = \rho'_o(x)$.
It follows directly from the local conclusions (1), (4), and (5).

$(\mathbf{E-WHILE_{true_{\bot}}})$ then we can conclude that :

(1) S is "**while** e **do** S_1 **done**" and:

- $\sigma_i(e) = \texttt{true}$ and $\rho_i(e) = \bot$
- " $(\sigma_i, \rho_i),\ t^{\text{pc}} \sqcup \bot \ \vdash\ S_1\ ;\ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ " is a sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ "

It follows directly from the definition of the rule $(\mathbf{E-WHILE_{true_{\bot}}})$.

(2) There exist a value v' and a tag t'_e such that:

- $\sigma'_i(e) = v'$ and $\rho'_i(e) = t'_e$

It follows from the global hypothesis $\star_5$, the local conclusion (1), and the definitions of the rules applying to "**while** e **do** S_1 **done**" ($(\mathbf{E-WHILE_{skip}})$, $(\mathbf{E-WHILE_{true_{\bot}}})$, $(\mathbf{E-WHILE_{true_{\top}}})$ and $(\mathbf{E-WHILE_{false_{\top}}})$).

(3) $t'_e = \bot$ and $v' = \mathit{true}$.
It follows directly from the local conclusions (1) and (2), lemmas A.7 and A.4, and the global hypotheses $\star_1$ and $\star_2$.

(4) " $(\sigma'_i, \rho'_i),\ t^{\text{pc}'} \sqcup \bot \ \vdash\ S_1\ ;\ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_o)$ ".
It follows from the global hypothesis $\star_5$, the local conclusions (1), (2), and (3), and the definition of the rule applying to "**while** e **do** S_1 **done**" whenever e evaluates to $\texttt{true}$ and its tag is $\bot$ $(\mathbf{E-WHILE_{true_{\bot}}})$.

(•) $\rho_o(x) = \rho'_o(x)$.
The conclusion is obtained by applying the inductive hypothesis on " $(\sigma_i, \rho_i),\ t^{\text{pc}} \sqcup \bot \ \vdash\ S_1\ ;\ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ " (sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ ") and " $(\sigma'_i, \rho'_i),\ t^{\text{pc}'} \sqcup \bot \ \vdash\ S_1\ ;\ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_o)$ ".

($\mathbf{E - WHILE_{true_\top}}$) then we can conclude that :

(1) S is "**while** e **do** S_1 **done**" and:

- $\sigma_i(e) = \texttt{true}$ and $\rho_i(e) = \top$
- " $(\sigma_i, \rho_i),\ t^{\text{pc}} \sqcup \top \ \vdash\ S_1\ ;\ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_l)$ " is a sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ "
- $\rho_o = \rho_i \sqcup \rho_l$

It follows directly from the definition of the rule ($\mathbf{E - WHILE_{true_\top}}$).

(2) There exist a value v' and a tag t'_e such that:

- $\sigma'_i(e) = v'$ and $\rho'_i(e) = t'_e$

It follows from the global hypothesis $\star_5$, the local conclusion (1), and the definitions of the rules applying to "**while** e **do** S_1 **done**" (($\mathbf{E - WHILE_{skip}}$), ($\mathbf{E - WHILE_{true_\bot}}$), ($\mathbf{E - WHILE_{true_\top}}$) and ($\mathbf{E - WHILE_{false_\top}}$)).

(3) $t'_e = \top$.
It follows directly from the local conclusions (1) and (2), lemma A.7, and the global hypothesis $\star_2$.

Case 1: $v' = \texttt{true}$

(a) There exists a tag store ρ'_l such that:

- " $(\sigma'_i, \rho'_i),\ t^{\text{pc}'} \sqcup \top \ \vdash\ S_1\ ;\ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_l)$ "
- $\rho'_o = \rho'_i \sqcup \rho'_l$

It follows from the global hypothesis $\star_5$, the local conclusions (1), (2), and (3), the case hypothesis, and the definitions of the rules applying to "**while** e **do** S_1 **done**" whenever e evaluates to `true` and its tag is $\top$ ($\mathbf{E - WHILE_{true_\top}}$).

(b) $\rho_l = \rho'_l$.
This is obtained by applying the inductive hypothesis on " $(\sigma_i, \rho_i),\ t^{\text{pc}} \sqcup \top \ \vdash\ S_1\ ;\ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_l)$ " (sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ ") and " $(\sigma'_i, \rho'_i),\ t^{\text{pc}'} \sqcup \top \ \vdash\ S_1\ ;\ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_l)$ ".

(•) $\rho_o = \rho'_o$.
It follows directly from the local conclusions (1), (a), and (b), and the global hypothesis $\star_2$.

Case 2: $v' = \texttt{false}$

(a) There exists an analysis result $\mathfrak{X}'$ and a tag store ρ'_e such that:

- $[\![(\sigma'_i, \rho'_i) \vdash S_1 \ ; \ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}]\!]^{\sharp_{\mathcal{G}}} = \mathfrak{X}'$
- $\rho'_e = (\mathfrak{X}' \times \{\top\}) \cup (\overline{\mathfrak{X}'} \times \{\bot\})$
- $\rho'_o = \rho'_i \sqcup \rho'_e$

It follows from the global hypothesis $\star_5$, the local conclusions (1), (2), and (3), the case hypothesis, and the definition of the rule applying to "**while** e **do** S_1 **done**" whenever e evaluates to `false` and its tag is $\top$ $((\mathbf{E-WHILE_{false_\top}}))$.

(b) $\rho_l = \rho'_i \sqcup \rho'_e$.
It follows from the definitions of ρ_l and ρ'_e (given in the local conclusions (1) and (a)), and the hypotheses A.2 and A.3.

(•) $\rho_o = \rho'_o$.
It follows directly from the global hypothesis $\star_2$ and the local conclusions (1), (a), and (b).

$(\mathbf{E-WHILE_{false_\top}})$ then we can conclude that :

(1) S is "**while** e **do** S_1 **done**" and there exists a tag store ρ_e such that:

- $\sigma_i(e) = \texttt{false}$ and $\rho_i(e) = \top$
- $[\![(\sigma_i, \rho_i) \vdash S_1 \ ; \ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}]\!]^{\sharp_{\mathcal{G}}} = \mathfrak{X}$
- $\rho_e = (\mathfrak{X} \times \{\top\}) \cup (\overline{\mathfrak{X}} \times \{\bot\})$
- $\rho_o = \rho_i \sqcup \rho_e$

It follows directly from the definition of the rule $(\mathbf{E-WHILE_{false_\top}})$.

(2) There exist a value $v' \in \{\texttt{true}, \texttt{false}\}$ and a tag t'_e such that:

- $\sigma'_i(e) = v'$ and $\rho'_i(e) = t'_e$

It follows from the global hypothesis $\star_5$, the local conclusion (1), and the definitions of the rules applying to "**while** e **do** S_1 **done**" $((\mathbf{E-WHILE_{skip}})$, $(\mathbf{E-WHILE_{true_\bot}})$, $(\mathbf{E-WHILE_{true_\top}})$ and $(\mathbf{E-WHILE_{false_\top}}))$.

(3) $t'_e = \top$.
It follows directly from the local conclusions (1) and (2), lemma A.7, and the global hypothesis $\star_2$.

Case 1: $v' = \texttt{false}$

(a) there exists a tag stores ρ'_e such that:

- $[\![(\sigma'_i, \rho'_i) \vdash S_1 \ ; \ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}]\!]^{\sharp_{\mathcal{G}}} = \mathfrak{X}'$
- $\rho'_e = (\mathfrak{X}' \times \{\top\}) \cup (\overline{\mathfrak{X}'} \times \{\bot\})$
- $\rho'_o = \rho'_i \sqcup \rho'_e$

It follows from the global hypothesis $\star_5$, the local conclusions (1), (2), and (3), the case hypothesis, and the definitions of the rule applying to "**while** e **do** S_1 **done**" whenever e evaluates to `false` and its tag is $\top$ $((\mathbf{E-WHILE_{false_\top}}))$.

(b) $\rho_e = \rho'_e$.
It follows from the definitions of ρ_e, and ρ'_e (given in the local conclusions (1) and (a)), the global hypothesis $\star_2$, and the lemma A.2.

(•) $\rho_o = \rho'_o$.
It follows directly from the definitions of ρ_o and ρ'_o (given in the local conclusions (1) and (a)), the local conclusions (b), and the global hypothesis $\star_2$.

Case 2: $v' = \texttt{true}$

(a) There exists a tag store ρ'_l such that:

- " $(\sigma'_i, \rho'_i),\ t^{\text{pc}'} \sqcup \top \ \vdash\ S_1\ ;\ \textbf{while}\ e\ \textbf{do}\ S_1\ \textbf{done}\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_l)$ "
- $\rho'_o = \rho'_i \sqcup \rho'_l$

It follows from the global hypothesis $\star_5$, the local conclusions (1), (2), and (3), the case hypothesis, and the definitions of the rules applying to "**while** e **do** S_1 **done**" whenever e evaluates to `true` and its tag is $\top$ ($(\mathbf{E-WHILE}_{\mathbf{true}_\top})$).

(b) $\rho'_l = \rho'_i \sqcup \rho_e$.
It follows from the definitions of ρ'_l and ρ_e (given in the local conclusions (1) and (a)), and the lemmas A.2 and A.3.

(•) $\rho_o = \rho'_o$.
It follows directly from the local conclusions (1), (a), and (b), and the global hypothesis $\star_2$

($\mathbf{E-SEQUENCE}$) then we can conclude that :

(1) S is "$S_1\ ;\ S_2$" and there exist σ_1 and ρ_1 such that:

- " $(\sigma_i, \rho_i),\ t^{\text{pc}} \ \vdash\ S_1\ \Downarrow_{\mathcal{M}}\ (\sigma_1, \rho_1)$ " is a sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ "
- " $(\sigma_1, \rho_1),\ t^{\text{pc}} \ \vdash\ S_2\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ " is a sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \ \vdash\ S\ \Downarrow_{\mathcal{M}}\ (\sigma_o, \rho_o)$ "

It follows directly from the definition of the rule ($\mathbf{E}_{\mathcal{M}} - \mathbf{SEQUENCE}$).

(2) There exist σ'_1 and ρ'_1 such that:

- " $(\sigma'_i, \rho'_i),\ t^{\text{pc}'} \ \vdash\ S_1\ \Downarrow_{\mathcal{M}}\ (\sigma'_1, \rho'_1)$ "
- " $(\sigma'_1, \rho'_1),\ t^{\text{pc}'} \ \vdash\ S_2\ \Downarrow_{\mathcal{M}}\ (\sigma'_o, \rho'_o)$ "

It follows directly from the global hypothesis $\star_5$, the local conclusion (1), and the definition of the only rule applying to "$S_1\ ;\ S_2$" ($(\mathbf{E}_{\mathcal{M}} - \mathbf{SEQUENCE})$).

(3) $\forall y : (\rho_1(y) \sqsubseteq \bot) \Rightarrow \sigma_1(y) = \sigma'_1(y)$.
This result is obtained by applying lemma A.6 on " $(\sigma_i, \rho_i),\ t^{\text{pc}} \ \vdash\ S_1\ \Downarrow_{\mathcal{M}}\ (\sigma_1, \rho_1)$ " and " $(\sigma'_i, \rho'_i),\ t^{\text{pc}'} \ \vdash\ S_1\ \Downarrow_{\mathcal{M}}\ (\sigma'_1, \rho'_1)$ " for any variable y such that $\rho_1(y) \sqsubseteq \bot$.

(4) $\rho_1 = \rho'_1$.
This result is obtained by applying the inductive hypothesis on " $(\sigma_i, \rho_i),\ t^{\text{pc}} \vdash S_1 \Downarrow_{\mathcal{M}} (\sigma_1, \rho_1)$ " (sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \vdash S \Downarrow_{\mathcal{M}} (\sigma_o, \rho_o)$ ") and " $(\sigma'_i, \rho'_i),\ t^{\text{pc}'} \vdash S_1 \Downarrow_{\mathcal{M}} (\sigma'_1, \rho'_1)$ " for any variable.

(•) $\rho_o(x) = \rho'_o(x)$.
The conclusion is obtained by applying the inductive hypothesis on " $(\sigma_1, \rho_1),\ t^{\text{pc}} \vdash S_2 \Downarrow_{\mathcal{M}} (\sigma_o, \rho_o)$ " (sub-derivation tree of " $(\sigma_i, \rho_i),\ t^{\text{pc}} \vdash S \Downarrow_{\mathcal{M}} (\sigma_o, \rho_o)$ ") and " $(\sigma'_1, \rho'_1),\ t^{\text{pc}'} \vdash S_2 \Downarrow_{\mathcal{M}} (\sigma'_o, \rho'_o)$ " using the local conclusions (3) and (4).

($\mathbf{E-ASSIGN}$) then we can conclude that :

(1) S is "$id := e$".

Case 1: $x \neq id$

(a) $\rho_o(x) = \rho_i(x)$.
It follows directly from the case hypothesis and the definition of the rule ($\mathbf{E-ASSIGN}$).

(b) $\rho'_o(x) = \rho'_i(x)$.
It follows directly from the global hypothesis $\star_5$, the local conclusion (1), the definition of the only rule applying to "$id := e$" (($\mathbf{E-ASSIGN}$)), and the case hypothesis.

(•) $\rho_o(x) = \rho'_o(x)$.
From the global hypothesis $\star_2$, $\rho_i(x) = \rho'_i(x)$. Hence, from the local conclusions (a) and (b), $\rho_o(x) = \rho'_o(x)$.

Case 2: $x = id$

(a) There exist v and t_e such that:
- $\sigma_i(e) = v$ and $\rho_i(e) = t_e$
- $\rho_o(x) = t_e \sqcup t^{\text{pc}}$

It follows directly from the case hypothesis and the definition of the rule ($\mathbf{E-ASSIGN}$).

(b) There exist v' and t'_e such that:
- $\sigma'_i(e) = v'$ and $\rho'_i(e) = t'_e$
- $\rho'_o(x) = t'_e \sqcup t^{\text{pc}'}$

It follows directly from the global hypothesis $\star_5$, the local conclusion (1), the definition of the only rule applying to "$id := e$" (($\mathbf{E-ASSIGN}$)), and the case hypothesis.

(c) $t_e = t'_e$.
This result comes from lemma A.7 and the global hypothesis $\star_2$.

(•) $\rho_o(x) = \rho'_o(x)$.
From the global hypothesis $\star_3$, $t^{\text{pc}'} = t^{\text{pc}}$. Then, the above result follows directly from the local conclusions (a), (b), and (c).

($\mathbf{E-SKIP}$) then we can conclude that :

(1) S is "**skip**" and $\rho_o = \rho_i$.
It follows directly from the definition of the rule $(\mathbf{E-SKIP})$.

(2) $\rho'_o = \rho'_i$.
It follows directly from the global hypothesis $\star_5$, the local conclusion (1), and the definition of the only rule applying to "**skip**" $((\mathbf{E-SKIP}))$.

(•) $\rho_o(x) = \rho'_o(x)$.
From the global hypothesis $\star_2$, $\rho_i(x) = \rho'_i(x)$. Hence, from the local conclusions (1) and (2), $\rho_o(x) = \rho'_o(x)$.

□