Statically Assuring Secrecy for Dynamic Concurrent Processes

Rachid Echahed
LEIBNIZ – IMAG – CNRS
F-38031 Grenoble, France
Rachid.Echahed@imag.fr

Frédéric Prost
LEIBNIZ – IMAG – CNRS
F-38031 Grenoble, France
Frederic.Prost@imag.fr

Wendelin Serwe
IRISA
F-35042 Rennes, France
serwe@irisa.fr

ABSTRACT

We propose a new algorithm of secrecy analysis in a framework integrating declarative programming and concurrency. The analysis of a program ensures that information can only flow from less sensitive levels toward more sensitive ones. Our algorithm uses a terminating abstract operational semantics which reduces the problem of secrecy to constraint solving within finite lattices. It departs in that from the previous works essentially based on type systems. Furthermore, our proposal is general and tackles a very large class of programs, featuring dynamic process creation, general sequential composition, recursive process calls and high level synchronization.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification

General Terms

Algorithms, Security, Verification.

Keywords

Safety, noninterference, abstract interpretation

1. INTRODUCTION

The need for secrecy increases as more and more (secret or sensitive) private data (e.g., credit card numbers, personal medical files, etc.) migrate through the Internet. Hence the protection of sensitive data becomes increasingly important. We follow the popular approach of defining secrecy as absence of *information flow* from sensitive to less sensitive data [20]. In this setting, a *privacy level* is assigned to each datum used by a program (similar to the model of [4]), where high levels denote highly sensitive data, and low levels represent public ones. A program respects secrecy if the observation of the data with low privacy levels does not allow an attacker to gather information about the secret data. In other words, public data may influence private data whereas the converse is forbidden, i.e., any modification of private data should not be observable at the public level. Hence the aim of secrecy analysis is to show the absence of information flow from high to low levels of privacy. This approach to secrecy has been introduced by [7] and studied for some time now; see [20] for a recent survey.

In this paper we present a static secrecy analysis for concurrent declarative programs [11]. Our computation model integrates term rewrite systems with process rewrite systems la process algebra. Thus it exhibits a rich set of features, including dynamic creation of parallel processes, general sequential composition (including synchronization with the termination of a parallel composition of an unbounded number of processes), recursive process calls and high level synchronization via a (constraint) store. Another point of this work is that in contrary to most of recent works [3, 5, 18, 19, 21, 23, 24, 25, 26, 27] which are based on *type systems*, we use an *abstract execution* to compute a set of constraints (on the assignment of privacy levels) the satisfaction of which is sufficient to guarantee secrecy. It has to be noted, that using this approach we can easily extend our analysis to obtain a compositional one, although this is generally a strong point of type based systems not available with abstract interpretations. In summary, our work is a step toward extending previous work in the dimension of language expressiveness and analysis precision, using a new style of approach (abstract interpretation versus type inference).

The rest of the paper is organized as follows. In Sect. 2 we briefly present the computation model considered in this paper. We give a precise definition of (the respect of) secrecy in Sect. 3. Roughly speaking, we consider a program to respect secrecy if information cannot flow from any privacy level toward a lower one. In other words, we consider secrecy from a non-interference point of view: a program respects secrecy if there are no interferences from higher levels toward lower ones. We present an abstract operational semantics in Sect. 4, which always terminates and allows the computation of a constraint set. We also show that a program respects secrecy, whenever these constraints are satisfied. Sect. 5 gives a brief comparison with some related work, and Sect. 6 concludes. To help the reviewers verify the accuracy of our results, we give some proofs and technical lemmas in App. A.

2. A FRAMEWORK FOR CONCURRENT DECLARATIVE PROGRAMMING

The computational model used in this paper is a simplified version of the one proposed in [11, 12, 22]; we refer to these works for a more detailed presentation of the complete model and its implementation. Roughly speaking, a program or a component in our framework consists of two parts $\mathcal{S} = (F, \mathrm{I\!R})$. $\mathrm{I\!R}$ is a set of process definitions and F is a declarative program or theory presentation, i.e., a set of formulæ (here, we consider Term Rewriting Systems (TRS)), which we call a *store*. We assume some familiarity of the reader with TRS (see, e.g., [8]).

The execution model of a component can be schematized as follows. Processes communicate by modifying a common store F, i.e., by altering, in a non-monotonic way, the current theory described by F, for example by simply redefining constants (e.g., adding a message in a queue) or by adding or deleting formulæ in F. Hence, the execution of processes will cause the transformation of the store F. Every change of the store is the result of the execution of an *action*.

We now define the different parts of our computational model. The first step is to formalize the notion of *store*, where constants and functions are defined.

Definition 1. *A* store *is a many sorted conditional* TRS $F = \langle \Sigma, \mathcal{R} \rangle$*, composed of a* signature Σ *and a set of* rules *(or* formulæ *)* $\mathcal{R}$*. A* signature $\Sigma = \langle S, \Omega \rangle$ *is a pair of a set* S *of* sorts *and a (S-indexed) family* Ω *of* operator *or* function *symbols, such that* Σ *contains at least the sort* `Truth` *with its constructor* True*. We note the (S-indexed) family of sets of* terms *for a signature* Σ *and variables* X *as* $T(\Sigma, X)$*. Rules (elements of* $\mathcal{R}$*) are of the form* $l \rightarrow r \mid c$ *which has to be read: "l rewrites to r if c holds", where l and r are terms of the same sort, and c is a term of sort* `Truth`*.*

Furthermore, we use the function $eval(F, t)$, which evaluates the term t to its normal form with respect to the store $F = \langle \Sigma, \mathcal{R} \rangle$. The actual (rewriting) strategy used by the operational semantics is not important in this paper.

We write $F \cup (l \rightarrow r \mid c)$ (respectively $F \setminus (l \rightarrow r \mid c)$) the store F to (from) which the rule $l \rightarrow r \mid c$ has been added (removed). Furthermore, we write $F \bullet (l \rightarrow r \mid c)$ the store equivalent to store F where all rules of the form $l \rightarrow r' \mid c'$ have been erased and rule $l \rightarrow r \mid c$ has been added.

Actions allow the modification of stores. We distinguish two kinds of actions: *elementary actions* like assignment, addition or removal of rules, and *guarded actions* that are executed atomically only if their guard evaluates to True, providing high level synchronization.

Definition 2. *An* action α *is a pair consisting of a guard* g *and a sequence of elementary actions* a_i*, written:* $[g \Rightarrow \mathsf{a}_1; \ldots; \mathsf{a}_n]$*. A* guard *is a term of sort* `Truth` *whose validity in the store is decidable (i.e., normalization to* True *is decidable). The* elementary actions a *we consider in this paper are assignment* (:=)*, addition of a rule to the store* (tell)*, removal of a rule from the store* (del) *and* (skip) *the (elementary) action that does nothing.*

Basic processes in our model are $\mathsf{success}$ (the process which terminates successfully), guarded actions α, or process calls $\mathrm{q}(t_1, \ldots, t_n)$. As usual in process algebra (see, e.g., [14]), we provide some operators for combining processes: parallel ($\|$) and sequential (;) composition as well as nondeterministic choice (+). Hence we have the following definition.

Definition 3. *A* process term p *is a (well-sorted) expression defined by the following grammar:* $p ::= \mathsf{success} \mid [g \Rightarrow \mathsf{a}] \mid \mathrm{q}(t_1, \ldots, t_n) \mid p;p \mid p \parallel p \mid p + p$.
A process q *is defined by a sentence of the form:*
$\mathrm{q}(x_1, \ldots, x_n) \Leftarrow \sum_{i=1}^{m} \alpha_i; p_i$ *where (for each i)* α_i *is an action and* p_i *is a process term, such that the free variables of* α_i *and* p_i *are included in the parameter set* $\{x_1; \ldots; x_n\}$*.*

Definition 4. *A* system $\mathcal{S}$ *is a tuple* $\langle F, \mathrm{I\!R} \rangle$*, where* F *is a store and* $\mathrm{I\!R}$ *a set of process definitions. A* program p *of system* $\mathcal{S}$ *is a closed process* $\mathrm{q}(t_1, \ldots, t_n)$ *where* $\mathrm{q}(x_1, \ldots, x_n)$ *is in* $\mathrm{I\!R}$*.*

The operational semantics of our execution model is defined by a transition system, defined by the rules shown in Fig. 1. The transition relation $\hookrightarrow$ describes the modification of the store by the execution of sequences of elementary actions. The execution of processes is described by the transition relation $\longrightarrow$. Transitions are of the form $\langle F, p \rangle \longrightarrow \langle F', p' \rangle$ where F is a store and p is a process term. The relation $\longrightarrow$ is defined modulo the (classical) equivalence relation $\equiv_p$ which states that the operators $\|$ and + are commutative and $\mathsf{success}$ may vanish. Notice that the execution of an action is atomic (see rule (P_{guard})).

Example 1. *As a first example, consider a direct translation of an example given in [5], showing how control flow can lead to information flow. It shows the limitations of the analysis of [26] in a concurrent context, and it is a simplified version of the example given in [25].*

$$\boxed{\begin{array}{lll}
\alpha & \Leftarrow & [c_\alpha = \text{TT} \Rightarrow SPY := \text{FF};\ c_\beta := \text{TT}];\ \mathsf{success} \\
\beta & \Leftarrow & [c_\beta = \text{TT} \Rightarrow SPY := \text{TT};\ c_\alpha := \text{TT}];\ \mathsf{success} \\
\gamma & \Leftarrow & \big([PIN = \text{TT} \Rightarrow c_\alpha := \text{TT}]; \mathsf{success}\big) \\
 & & +\big([PIN = \text{FF} \Rightarrow c_\beta := \text{TT}]; \mathsf{success}\big)
\end{array}}$$

PIN, SPY, c_α and c_β are constants of sort bool *defined by the two constructors* TT *and* FF*. c_α and c_β are initiated to* FF*. Notice that execution of $\alpha \parallel \beta \parallel \gamma$ copies the (secret) value PIN into the (public) constant SPY.*

Example 2. *For the second example we present a toy program, illustrating several points in a small amount of code. Notice that the process* s *has a non-terminating execution.*

$$\boxed{\begin{array}{l}
\mathrm{s} \Leftarrow [\mathrm{test}_1(g_1) \Rightarrow c := c+1];\ \mathrm{s} \parallel \mathrm{p}(g_1) \\
\quad + [\mathrm{test}_2(g_2) \Rightarrow c := c+1];\ \mathrm{s} \parallel \mathrm{p}(g_2) \\
\quad + [\mathrm{test}_3(c) \ \ \Rightarrow g_1 := 0 \quad];\ \mathrm{s}
\end{array}}$$

This process roughly represents a server awaiting requests on g_1 and g_2. If some conditions are met (these situations are represented by test_i*) then, the server modifies (this might be viewed as an acknowledgment) the value of a constant c (a counter of the number of requests) and executes recursively itself in parallel with a process p parameterized by the request. One may think that g_i represents a bank account and p produces a service and decreases the account by some amount of money (the price of the service). When the number of requests satisfies some condition (*test_3*), then a message is sent to g_1 (that can be seen as the administrator of the server). We will show (cf. Ex. 3 and 5) that, depending on the circumstances, this program respects secrecy or not.*

$$\langle F, \mathsf{tell}(R); a\rangle \hookrightarrow \langle F \cup R, a\rangle \quad (ea_{\mathsf{tell}}) \qquad \langle F, f := t; a\rangle \hookrightarrow \langle F \bullet (f \to eval(F,t)), a\rangle \quad (ea_{:=})$$
$$\langle F, \mathsf{del}(R); a\rangle \hookrightarrow \langle F \setminus R, a\rangle \quad (ea_{\mathsf{del}}) \qquad \langle F, \mathsf{skip}; a\rangle \hookrightarrow \langle F, a\rangle \quad (ea_{\mathsf{skip};})$$
$$\mathsf{success}; p \equiv_p p \quad (Eq_{\mathsf{success};}) \qquad p_1 + p_2 \equiv_p p_2 + p_1 \quad (Eq_{+\ com})$$
$$\mathsf{success} \parallel p \equiv_p p \quad (Eq_{\mathsf{success}\parallel}) \qquad p_1 \parallel p_2 \equiv_p p_2 \parallel p_1 \quad (Eq_{\parallel\ com})$$

$$\frac{p_1 \equiv_p p_2 \quad \langle F, p_2\rangle \longrightarrow \langle F', p_3\rangle \quad p_3 \equiv_p p_4}{\langle F, p_1\rangle \longrightarrow \langle F', p_4\rangle} \quad (P_{\equiv_p})$$

$$\frac{\langle F, a_1; \dots; a_n; \mathsf{skip}\rangle \hookrightarrow^* \langle F', \mathsf{skip}\rangle \quad eval(F, g) = \mathsf{True}}{\langle F, [g \Rightarrow a_1; \dots; a_n]\rangle \longrightarrow \langle F', \mathsf{success}\rangle} \quad (P_{guard})$$

$$\frac{(\mathrm{q}(x_1,\dots,x_n) \Leftarrow \Sigma_{j=1}^m \alpha_j; p_j) \in \Pi\mathcal{R} \qquad \langle F, (\Sigma_{j=1}^m \alpha_j; p_j)[x_i/t_i]\rangle \longrightarrow \langle F', p'\rangle}{\langle F, \mathrm{q}(t_1,\dots,t_n)\rangle \longrightarrow \langle F', p'\rangle} \quad (P_{abs})$$

$$\frac{\langle F, p_1\rangle \longrightarrow \langle F', p_1'\rangle}{\langle F, p_1 + p_2\rangle \longrightarrow \langle F', p_1'\rangle} \quad (P_+) \qquad \frac{\langle F, p_1\rangle \longrightarrow \langle F', p_1'\rangle}{\langle F, p_1 \ \mathsf{op}\ p_2\rangle \longrightarrow \langle F', p_1' \ \mathsf{op}\ p_2\rangle} \quad \mathsf{op} \in \{\parallel, ;\} \quad (P_{\mathsf{op}})$$

Figure 1: Inference Rules Defining the Operational Semantics

3. FORMALIZATION OF SECRECY

In this section we precisely define what we intend by secrecy in our setting. In line with [4], we assign to each symbol in a store F a *privacy level* indicating its "status". The higher the privacy level of a value, the more private (or secret) is the status of this value. Thus we assign to every symbol of the signature, i.e., all elements of Ω, an element of a lattice $\mathfrak{L}$. We note $\sqsubseteq$ the order defined on $\mathfrak{L}$ and make no difference between the lattice and the set of its elements, i.e., $\mathfrak{L}$ denotes at the same time the lattice and its carrier. If π_1, π_2 are two elements of $\mathfrak{L}$ and $\pi_1 \sqsubseteq \pi_2$, we say that π_2 is *more private* than π_1. We write $\sqcup$ for the join operation (least upper bound) and $\sqcap$ for the meet operation (greatest lower bound). The upper bound of $\mathfrak{L}$ is $\top$ and its lower bound $\bot$.

Definition 5. *Let F be a store. We call any map ℓ from symbols of Ω toward $\mathfrak{L}$ a* privacy map. *We extend naturally any privacy map ℓ to all terms in $T(\Sigma, X)$. The privacy level of a term is recursively defined as the least upper bound of the privacy level of its subterms*[1]*:*

$$\ell(f(t_1,\dots,t_n)) = \left(\bigsqcup_{i=1}^n \ell(t_i)\right) \sqcup \ell(f)$$
$$\ell(x) = \bot \qquad (x \textit{ is a variable})$$

In the following we suppose that we are given a privacy map ℓ. When we talk of the privacy level of a term t we intend $\ell(t)$. We also define the notion of privacy level for rewrite rules: for a rule $l \to r \mid c$, $\ell(l \to r \mid c)$ is equal to $\ell(l)$. Finally we extend the notion of privacy level to actions. For an action we use the greatest lower bound, i.e., we extend ℓ with the following equations:

$$\ell(\mathsf{skip}) = {}^2\top \qquad \ell(f := t) = \ell(f)$$
$$\ell(\mathsf{tell}(l \to r \mid c)) = \ell(\mathsf{del}(l \to r \mid c)) = \ell(l)$$
$$\ell([g \Rightarrow a_1; \dots; a_n]) = \bigsqcap_{i=1}^n \ell(a_i)$$

We also define a notion of safe rewrite rules. Informally, a rewrite rule is safe whenever no information may flow from a higher level toward a lower one. Consider two constants PIN and SPY such that $\ell(PIN) = \top$ and $\ell(SPY) = \bot$. Then rewrite rules such as $SPY \to PIN$ or $SPY \to v \mid PIN = v'$ transmit a high level information to a lower one, by evaluating SPY. Hence we consider rewrite rules safe when the result of a rewrite step is a term of a lower or equal privacy than what it depends upon.

[1]By assigning $\bot$, the neutral element for $\sqcup$, to variables, we ensure that they do not interfere with the privacy level of a term.

Definition 6. *A rewrite rule $l \to r \mid c$ is* safe *whenever the following condition holds: $\big(\ell(r) \sqsubseteq \ell(l)\big) \wedge \big(\ell(c) \sqsubseteq \ell(l)\big)$.*

A store $F = \langle \Sigma, \mathcal{R}\rangle$ is safe, *if all rewrite rules $l \to r \mid c \in \mathcal{R}$ are safe.*

In order to define processes respecting secrecy, we have first to define a notion of equivalence between stores up to a given privacy level. Informally, two stores are $\pi\ell$-equivalent if they agree on all information the privacy level of which is less than π.

Definition 7. *Let F_0, F_1 be two stores, ℓ a privacy map and $\pi \in \mathfrak{L}$. We say that F_0 and F_1 are* $\pi\ell$-equivalent *and write $F_0 \cong_\pi^\ell F_1$ iff $\forall i \in \{0,1\}$ we have $\forall \rho_i \in \mathcal{R}_i$ and $\ell(\rho_i) \sqsubseteq \pi$, $\exists \rho_{1-i} \in \mathcal{R}_{1-i}$, such that $\rho_i = \rho_{1-i}$, up to variable renaming.*

A property of safe $\pi\ell$-equivalent stores is that the evaluation of a term is independent from the rules of a higher privacy than π. This property is expressed formally and proved in App. A.1.

We now define a notion of bisimulation of processes up to a privacy level π and with respect to a privacy map ℓ. Informally, two processes p_1, p_2 are $\pi\ell$-bisimilar when executed on $\pi\ell$-equivalent stores, they remain $\pi\ell$-bisimilar. In other words, either one can execute p_1 and then for each execution there is an execution of p_2 such that stores modified by these executions remain $\pi\ell$-equivalent, or p_1 can't be executed. In this last case, we ensure that every execution of p_2 only affects parts of the store with a higher privacy than π. Hence $\pi\ell$-bisimulation formalizes the idea that information may flow from low privacy levels to high privacy levels but not vice-versa. More formally:

Definition 8. *Let p_1, p_2 be two $\Pi\mathcal{R}$-process terms, $\pi \in \mathfrak{L}$, and F_1, F_2 two stores, and ℓ a privacy map such that $F_1 \cong_\pi^\ell$*

F_2. A relation $\mathcal{B}^{\ell}_{\pi}$ (on pairs of stores and process terms) is a $\pi\ell$-bisimulation *if it is symmetric and if $\langle F_1, p_1\rangle \ \mathcal{B}^{\ell}_{\pi} \ \langle F_2, p_2\rangle$ such that $\langle F_1, p_1\rangle \longrightarrow \langle F_1', p_1'\rangle$ then*

- *either $\exists F_2', p_2'$ such that $\langle F_2, p_2\rangle \longrightarrow \langle F_2', p_2'\rangle$ and $\langle F_1', p_1'\rangle \ \mathcal{B}^{\ell}_{\pi} \ \langle F_2', p_2'\rangle$*
- *or $\langle F_2, p_2\rangle \not\longrightarrow$ and $\forall F_1^{\sharp}, p_1^{\sharp}$ s.t. $\langle F_1', p_1'\rangle \longrightarrow^* \langle F_1^{\sharp}, p_1^{\sharp}\rangle$ we have $F_1^{\sharp} \cong^{\ell}_{\pi} F_2$.*

In the sequel, we consider only the largest $\pi\ell$-bisimulation, which we note as $\approx^{\ell}_{\pi}$.

Now we can define when a process respects secrecy, namely when it is bisimilar to itself for every privacy level.

Definition 9. *A program* p *of a system $\langle F, \Pi\!\mathcal{R}\rangle$* respects secrecy *for a privacy map ℓ, iff for all π and F' such that $F' \cong^{\ell}_{\pi} F$, we have $\langle F', \mathsf{p}\rangle \approx^{\ell}_{\pi} \langle F, \mathsf{p}\rangle$.*

Example 3. *We now examine examples 1 and 2 from a secrecy point of view.*

Regarding Ex. 1, suppose that we start with a store F in which PIN is TT *and that we have $\ell(PIN) = \top$ and $\ell(SPY) = \ell(c_\alpha) = \ell(c_\beta) = \bot$ with $\bot \sqsubset \top$. $\alpha \parallel \beta \parallel \gamma$ is not safe with respect to ℓ and F. Indeed, take a store F_1 in which PIN equals* FF *and SPY, c_α, c_β have the same values as in F. We have $F_1 \cong^{\ell}_{\bot} F$, because all operators with a privacy level less or equal to $\bot$ are equally defined in both F and F_1. Now, if one executes $\alpha \parallel \beta \parallel \gamma$ with F, one will obtain a store where the value of SPY is* TT *whereas with F_1 the value of SPY is* FF. *Thus we get two stores that are no longer $\pi\ell$-equivalent which is in contradiction with Def. 9.*

For Ex. 2, let $\ell(c) = \pi_c$, $\ell(g_1) = \pi_1$ and $\ell(g_2) = \pi_2$. We may show that if $\pi_c \sqsubset \pi_1$, then s *does not respect secrecy. Indeed, it is easy to see that g_1 depends on c and vice versa. Hence a modification of g_1 may influence the value of c, a constant with a lower privacy. On the other hand, if $\pi_2 \sqsubseteq \pi_c = \pi_1$ then* s *respects secrecy because the value of g_2 does depend neither on c nor g_1.*

4. ABSTRACTION AND CONSTRAINT GENERATION

We develop an abstract operational semantics aiming at the generation of a set of constraints, i.e., inequations on privacy levels, representing the constraints satisfied by a program respecting secrecy. If the constraint set cannot be satisfied, then the analyzed program *may* not respect secrecy.

Since our abstract operational semantics always terminates, it plays the same role as the type systems of [3, 5, 18, 19, 21, 23, 24, 25, 26, 27]. There, type inference is used to analyze non-interference properties of programs, either for imperative, functional or concurrent programs. Types inferred in these analyses are not similar to “standard types”, say `Int` $\rightarrow$ `Int`, they are rather like $\texttt{Int}^{\pi_1} \rightarrow \texttt{Int}^{\pi_2}$. That is to say, normal types are used as backbone where to put privacy annotations. This whole idea of using “standard types” as skeleton for the analysis has been studied in [17]. We follow this strategy and use the skeleton of an execution in order to collect constraints on privacy levels.

First we define the abstract operational semantics and show afterward how it can be used to analyze the respect of secrecy by a program.

4.1 Abstract Operational Semantics

Suppose that we are given a system $\mathcal{S} = \langle F, \Pi\!\mathcal{R}\rangle$. We define an abstract operational semantics for $\mathcal{S}$. Informally, the abstraction of the store is a set of inequations over privacy levels. For each symbol of the signature of F, we introduce a constant denoting its privacy level. The abstract operational semantics collects inequations through abstract executions. Since the number of these inequations is finite, it is possible to produce the whole set of inequations for a program. We prove in Theorem 1 that if a privacy map ℓ defined on $\mathcal{S}$ is such that all inequations hold, then the analyzed program respects secrecy for ℓ. This is a kind of abstract interpretation [6].

Abstract stores are sets of privacy inequations. We define *privacy formulæ* $\mathfrak{f}$ by the following grammar (where c denotes a constant): $\mathfrak{f} ::= \pi \mid c \mid \mathfrak{f} \sqcap \mathfrak{f} \mid \mathfrak{f} \sqcup \mathfrak{f}$. *Privacy inequations* are statements of the form $\mathfrak{f}_1 \sqsubseteq \mathfrak{f}_2$.

Definition 10. *An* abstract store *$F^{\mathcal{A}}$ is a couple $\langle \Sigma^{\mathcal{A}}, \mathcal{R}^{\mathcal{A}}\rangle$ where the signature $\Sigma^{\mathcal{A}} = \langle \{s^{\mathcal{A}}\}, \Omega^{\mathcal{A}}\rangle$ defines a set of constants $\Omega^{\mathcal{A}}$ and $\mathcal{R}^{\mathcal{A}}$ is a set of privacy inequations build with symbols belonging to $\Sigma^{\mathcal{A}}$. For a store $F = \langle \Sigma, \mathcal{R}\rangle$, we define $F^{\mathcal{A}} = \langle \Sigma^{\mathcal{A}}, \mathcal{R}^{\mathcal{A}}\rangle$, its* abstract store*, as follows: for all elements $f \in \Sigma$, there is a constant x_f (of sort $s^{\mathcal{A}}$) in $\Omega^{\mathcal{A}}$ and for all rules $l \rightarrow r \mid c$ in $\mathcal{R}$, there are inequations $\ell^{\mathcal{A}}(r) \sqsubseteq \ell^{\mathcal{A}}(l), \ell^{\mathcal{A}}(c) \sqsubseteq \ell^{\mathcal{A}}(l)$ in $\mathcal{R}^{\mathcal{A}}$, where $\ell^{\mathcal{A}}$ is the function defined by: $\ell^{\mathcal{A}}(f) = x_f$ ($\forall f \in \Sigma$) and $\ell^{\mathcal{A}}(f(t_1, \ldots, t_n)) = \ell^{\mathcal{A}}(f) \sqcup \ell^{\mathcal{A}}(t_1) \sqcup \ldots \sqcup \ell^{\mathcal{A}}(t_n)$. We call F the* original store *of $F^{\mathcal{A}}$ and write in the following $t^{\mathcal{A}}$ instead of $\ell^{\mathcal{A}}(t)$.*

We use an abstract execution to collect constraints that ensure secrecy. Informally, the abstract operational semantics is defined by a transition system, the states of which are tuples $\langle F^{\mathcal{A}}, \langle p^{\mathcal{A}}, \sigma\rangle\rangle$ consisting of an abstract store $F^{\mathcal{A}}$, an abstract process term $p^{\mathcal{A}}$ and a privacy level σ, corresponding to the highest level checked in a guard up to the current point in the execution. Abstract transitions generate constraints, depending on σ as well as on the privacy level of terms manipulated, and record them into the abstract store. Fig. 2 gives the rules defining the abstract transition relation. This transition system is obviously infinitely branching due to rules ($\mathsf{A}Eq_{\equiv_p}$), ($Eq_{\mathsf{success}_{\parallel}}$) and ($Eq_{\mathsf{success};}$). However, it is clear that we can get a finitely branching transition system by considering term processes modulo the equations ($Eq_{\mathsf{success};}$) and ($Eq_{\mathsf{success}_{\parallel}}$).

Abstract execution of sequences of elementary actions is described by the relation $\hookrightarrow^{\mathcal{A}}$. Abstract elementary actions modify an abstract store $F^{\mathcal{A}}$ with respect to a privacy level σ of $\mathfrak{L}$. Since in a parallel composition $p \parallel q$, p might check guards of a high level while q only works on low privacy levels, we need to duplicate the privacy level σ, in order to not reject such processes (as constraints generated by p can be too strong compared to q). Hence, we introduce *abstract operators* $+^{\mathcal{A}}$, $\parallel^{\mathcal{A}}$, $;^{\mathcal{A}}$and *abstract process terms* (or $\mathcal{M}$-terms), defined by the grammar: $\mathcal{M} ::= \langle p, \sigma\rangle \mid \mathcal{M} \parallel^{\mathcal{A}} \mathcal{M} \mid \mathcal{M} +^{\mathcal{A}} \mathcal{M} \mid \mathcal{M} \ ;^{\mathcal{A}} \mathcal{M}$. In some sense, the role of $\parallel^{\mathcal{A}}$ is similar to the one of the subtyping rule in type based analyses (see for instance [5]), namely to allow to associate different types for both parts of a parallel composition and then to use the subtyping rule in order to have the same type in both branches in the type derivation tree. The other two abstract operators are defined for notational coherence and do not influence the analysis. In order to translate between the

$$\langle F^{\mathcal{A}}, \langle f := t; a, \sigma\rangle\rangle \hookrightarrow^{\mathcal{A}} \langle F^{\mathcal{A}} \cup \{t^{\mathcal{A}} \sqsubseteq f^{\mathcal{A}}\} \cup \{\sigma \sqsubseteq f^{\mathcal{A}}\}, \langle a, \sigma\rangle\rangle \qquad (\mathsf{A}ea_{:=})$$

$$\langle F^{\mathcal{A}}, \langle \mathsf{tell}(l \to r \mid c); a, \sigma\rangle\rangle \hookrightarrow^{\mathcal{A}} \big\langle F^{\mathcal{A}} \cup \{r^{\mathcal{A}} \sqsubseteq l^{\mathcal{A}}\} \cup \{c^{\mathcal{A}} \sqsubseteq l^{\mathcal{A}}\} \cup \{\sigma \sqsubseteq l^{\mathcal{A}}\}, \langle a, \sigma\rangle\big\rangle \qquad (\mathsf{A}ea_{\mathsf{tell}})$$

$$\langle F^{\mathcal{A}}, \langle \mathsf{del}(l \to r \mid c); a, \sigma\rangle\rangle \hookrightarrow^{\mathcal{A}} \langle F^{\mathcal{A}} \cup \{\sigma \sqsubseteq l^{\mathcal{A}}\}, \langle a, \sigma\rangle\rangle \qquad (\mathsf{A}ea_{\mathsf{del}})$$

$$\langle F^{\mathcal{A}}, \langle \mathsf{skip}; a, \sigma\rangle\rangle \hookrightarrow^{\mathcal{A}} \langle F^{\mathcal{A}}, \langle a, \sigma\rangle\rangle \qquad (\mathsf{A}ea_{\mathsf{skip;}})$$

$$\frac{p_1 \equiv_p p_2}{\langle p_1, \sigma\rangle \equiv^{\mathcal{A}} \langle p_2, \sigma\rangle} \quad (\mathsf{A}Eq_{\equiv_p}) \qquad \mathcal{M}_1 \|^{\mathcal{A}} \mathcal{M}_2 \equiv^{\mathcal{A}} \mathcal{M}_2 \|^{\mathcal{A}} \mathcal{M}_1 \quad (\mathsf{A}Eq_{\|^{\mathcal{A}}})$$

$$\mathcal{M}_1 +^{\mathcal{A}} \mathcal{M}_2 \equiv^{\mathcal{A}} \mathcal{M}_2 +^{\mathcal{A}} \mathcal{M}_1 \quad (\mathsf{A}Eq_{+^{\mathcal{A}}})$$

$$\langle p_1 \ \mathsf{op}\ p_2, \sigma\rangle \longmapsto \langle p_1, \sigma\rangle\ \mathsf{op}^{\mathcal{A}}\ \langle p_2, \sigma\rangle \quad \mathsf{op} \in \{\|, ;, +\} \qquad (\mathsf{Aop}^{\mathcal{A}}\text{-}I)$$

$$\langle \mathsf{success}, \sigma_1\rangle \|^{\mathcal{A}} \langle \mathsf{success}, \sigma_2\rangle \longmapsto \langle \mathsf{success}, \sigma_1 \sqcup \sigma_2\rangle \qquad (\mathsf{A}\|^{\mathcal{A}}\text{-}E)$$

$$\langle \mathsf{success}, \sigma_1\rangle \ ;^{\mathcal{A}} \langle p_2, \sigma_2\rangle \longmapsto \langle p_2, \sigma_1 \sqcup \sigma_2\rangle \qquad (\mathsf{A}\,;^{\mathcal{A}}\text{-}E)$$

$$\frac{\overline{\mathcal{M}_1}^{\mathsf{nf}} \equiv^{\mathcal{A}} \mathcal{M}_2 \quad \langle F^{\mathcal{A}}, \mathcal{M}_2\rangle \longrightarrow^{\mathcal{A}} \langle F^{\mathcal{A}\prime}, \mathcal{M}_3\rangle \quad \overline{\mathcal{M}_3}^{\mathsf{nf}} \equiv^{\mathcal{A}} \mathcal{M}_4}{\langle F^{\mathcal{A}}, \mathcal{M}_1\rangle \longrightarrow^{\mathcal{A}} \langle F^{\mathcal{A}\prime}, \mathcal{M}_4\rangle} \qquad (\mathsf{A}P_{\equiv^{\mathcal{A}}})$$

$$\frac{\langle F^{\mathcal{A}}, \langle a_1; \ldots; a_n; \mathsf{skip}, \sigma \sqcup g^{\mathcal{A}}\rangle\rangle \hookrightarrow^{\mathcal{A}*} \langle F^{\mathcal{A}\prime}, \langle \mathsf{skip}, \sigma \sqcup g^{\mathcal{A}}\rangle\rangle}{\langle F^{\mathcal{A}}, \langle [g \Rightarrow a_1; \ldots; a_n], \sigma\rangle\rangle \longrightarrow^{\mathcal{A}} \langle F^{\mathcal{A}\prime}, \langle \mathsf{success}, \sigma \sqcup g^{\mathcal{A}}\rangle\rangle} \qquad (\mathsf{A}P_{guard})$$

$$\frac{(\mathrm{q}(x_1, \ldots, x_n) \Leftarrow \Sigma_{j=1}^{m} \alpha_j; p_j) \in \mathrm{I\!R} \quad \langle F^{\mathcal{A}}, \langle (\Sigma_{j=1}^{m} \alpha_j; p_j)[x_i/t_i], \sigma\rangle\rangle \longrightarrow^{\mathcal{A}} \langle F^{\mathcal{A}\prime}, \mathcal{M}'\rangle}{\langle F^{\mathcal{A}}, \langle \mathrm{q}(t_1, \ldots, t_n), \sigma\rangle\rangle \longrightarrow^{\mathcal{A}} \langle F^{\mathcal{A}\prime}, \mathcal{M}'\rangle} \qquad (\mathsf{A}P_{abs})$$

$$\frac{\langle F^{\mathcal{A}}, \mathcal{M}_1\rangle \longrightarrow^{\mathcal{A}} \langle F^{\mathcal{A}\prime}, \mathcal{M}_1'\rangle}{\langle F^{\mathcal{A}}, \mathcal{M}_1 \mathsf{op}^{\mathcal{A}} \mathcal{M}_2\rangle \longrightarrow^{\mathcal{A}} \langle F^{\mathcal{A}\prime}, \mathcal{M}_1' \mathsf{op}^{\mathcal{A}} \mathcal{M}_2\rangle} \quad \mathsf{op} \in \{;, \|\} \qquad (\mathsf{A}P_{\mathsf{op}^{\mathcal{A}}})$$

$$\frac{\langle F^{\mathcal{A}}, \mathcal{M}_1\rangle \longrightarrow^{\mathcal{A}} \langle F^{\mathcal{A}\prime}, \mathcal{M}_1'\rangle}{\langle F^{\mathcal{A}}, \mathcal{M}_1 +^{\mathcal{A}} \mathcal{M}_2\rangle \longrightarrow^{\mathcal{A}} \langle F^{\mathcal{A}\prime}, \mathcal{M}_1'\rangle} \qquad (\mathsf{A}P_{+^{\mathcal{A}}})$$

Figure 2: Abstract Operational Semantics

concrete and abstract operators, we define a *transformation relation* $\longmapsto$. It is clear that $\longmapsto$ is confluent and strongly normalizing. We write $\overline{\mathcal{M}}^{\mathsf{nf}}$ the normal form of $\mathcal{M}$ for $\longmapsto$.

By inspection of the similarities of the inference rules in Figs. 1 and 2 we prove in App. A.3 that to each concrete transition step corresponds an abstract reduction step.

4.2 Program Analysis

The idea of our analysis is to collect the constraints computed by *all* possible abstract executions. We claim that if there exists a substitution of variables in the privacy formulæ to $\mathfrak{L}$ elements satisfying the computed constraints, then the program respects secrecy for any privacy map assigning the same privacy to f as the substitution associates to x_f (see Def. 10). Crucial for the termination of our analysis is that the abstract store becomes stable during an abstract execution, i.e., after a certain point no more new privacy inequations are created. This is due to the fact that abstract elementary actions only increase the number of privacy inequations in the store. Furthermore, abstract executions are purely symbolic, and since the number of symbols appearing in a program is finite so is the number of inequations that can be generated by this program. Therefore it is possible to cut infinite branches, when no more information can be collected.

Definition 11. *The* analysis reduction $\rightsquigarrow$ *is the relation between triples of the form* $\langle F^{\mathcal{A}}, \mathcal{M}, \mathfrak{H}\rangle$*, where* $\mathfrak{H}$ *(denoting the* $\mathfrak{H}$*istory of executed process calls) is a set of pairs of the form* $\langle \mathrm{q}, [\ell^{\mathcal{A}}(t_1); \ldots; \ell^{\mathcal{A}}(t_n)]\rangle$*.* $\rightsquigarrow$ *is defined as follows:*

- *if* $\langle F^{\mathcal{A}}, \mathcal{M}\rangle \longrightarrow^{\mathcal{A}} \langle F^{\mathcal{A}\prime}, \mathcal{M}'\rangle$ *using a reduction rule different from* $(\mathsf{A}P_{abs})$*, then* $\langle F^{\mathcal{A}}, \mathcal{M}, \mathfrak{H}\rangle \rightsquigarrow \langle F^{\mathcal{A}\prime}, \mathcal{M}, \mathfrak{H}\rangle$ *and*
- *if* $\mathcal{M} = \langle \mathrm{q}(t_1, \ldots, t_n), \sigma\rangle$ *and* $\langle F^{\mathcal{A}}, \langle (\Sigma_{j=1}^{m} \alpha_j; p_j)[x_i/t_i], \sigma\rangle\rangle \longrightarrow^{\mathcal{A}} \langle F^{\mathcal{A}\prime}, \mathcal{M}'\rangle$*, where the process* q *is defined by* $\big(\mathrm{q}(x_1, \ldots, x_n) \Leftarrow \Sigma_{j=1}^{m} \alpha_j; p_j\big) \in \mathrm{I\!R}$*, then*

$$\langle F^{\mathcal{A}}, \mathcal{M}, \mathfrak{H}\rangle \rightsquigarrow \begin{cases} \langle F^{\mathcal{A}}, \langle \mathsf{success}, \sigma\rangle, \mathfrak{H}\rangle \\ \qquad \textit{if } \langle \mathrm{q}, [t_1{}^{\mathcal{A}}; \ldots; t_n{}^{\mathcal{A}}]\rangle \in \mathfrak{H} \\ \langle F^{\mathcal{A}\prime}, \mathcal{M}', \mathfrak{H} \cup \langle \mathrm{q}, [t_1{}^{\mathcal{A}}; \ldots; t_n{}^{\mathcal{A}}]\rangle\rangle \\ \qquad \textit{otherwise} \end{cases}$$

Using that the number of non equivalent abstract process calls is finite (since $\sqcup$ is idempotent and associative), we prove in App. A.5 that there are no infinite $\rightsquigarrow$ reduction sequences. Since furthermore the number of rules that are applicable is always finite, we can define the result returned by our analysis as the collection of all reachable abstract stores.

Definition 12. *Let* $\mathcal{S} = \langle F, \mathrm{I\!R}\rangle$ *be a system and* p *be a program of* $\mathcal{S}$*. We call* skeleton *of* p*, and write* $\mathsf{p}_F^{\natural}$*, the constraint set* $\bigcup_{i \in I} F_i^{\mathcal{A}}$ *where the index set* I *is defined such that* $\forall \langle F_i^{\mathcal{A}}, \mathsf{success}, \sigma_i\rangle$ *reachable from* $\langle F^{\mathcal{A}}, \langle \mathsf{p}, \bot\rangle\rangle$ *using* $\rightsquigarrow$*,* $i \in I$*.*

We now address the question of *how* this analysis can be used. The idea is that if we consider a program p, a privacy map ℓ, and if every constraint of $\mathsf{p}_F^{\natural}$ is compatible with ℓ then the program respects secrecy, otherwise there *might* be information flow from high privacy levels to lower ones. We say that a constraint set $F^{\mathcal{A}}$ is *satisfied* by a substitution $\mathfrak{S}$ from $\Sigma^{\mathcal{A}}$ to $\mathfrak{L}$, iff for every rule $l \sqsubseteq r$ of $\mathcal{R}^{\mathcal{A}}$ $\mathfrak{S}(l) \sqsubseteq \mathfrak{S}(r)$ is correct, where $\mathfrak{S}(l)$ is the natural extension of $\mathfrak{S}$ to terms of the form $x_1 \sqcup \ldots \sqcup x_n$. Informally, compatibility between an abstract store $F^{\mathcal{A}}$ and a privacy map ℓ expresses that the inequations of the abstract store are verified for the privacy map defined on the original store.

Definition 13. *A program skeleton* $\mathsf{p}^{\natural}_{F}$ *is told* compatible *with a privacy map* ℓ *iff* $\mathfrak{S}_\ell$ *satisfies* $\mathsf{p}^{\natural}_{F}$, *where* $\mathfrak{S}_\ell$ *is defined by:* $\forall f \in F \quad \mathfrak{S}_\ell(x_f) = \ell(f)$.

If the skeleton of a program is compatible with a privacy map ℓ, then we claim that this program is safe with respect to ℓ.

Example 4. *Consider again* $\alpha \parallel \beta \parallel \gamma$ *as defined in Example 1. As in Example 3, suppose* $\ell(PIN) = \top$ *and* $\ell(SPY) = \bot$. *The abstract execution of* $\alpha \parallel \beta \parallel \gamma$ *produces a collection of constraints separately generated by* α, β, γ. *Consider* γ. *From rules* $(\mathsf{A}P_{guard})$ *and* $(\mathsf{A}ea_{:=})$ *(condition* $\sigma \sqsubseteq c^{\mathcal{A}}$ *in definition of* $(\mathsf{A}ea_{:=})$*), we have* $\top \sqsubseteq {c_\alpha}^{\mathcal{A}}$ *and* $\top \sqsubseteq {c_\beta}^{\mathcal{A}}$. *On the other hand, considering* α, *the same rules lead to* ${c_\alpha}^{\mathcal{A}} \sqsubseteq \bot$. *It is clear that the constraints generated by* γ *and* α *are not compatible with* ℓ. *Indeed we have* $\top \sqsubseteq {c_\alpha}^{\mathcal{A}} \sqsubseteq \bot$.

Example 5. *Example 2 illustrates that the analysis terminates, even in presence of recursive processes. If we suppose that* $\mathrm{test}_1, \mathrm{test}_2$ *and p only generate trivial inequations (like* ${g_1}^{\mathcal{A}} \sqsubseteq {g_1}^{\mathcal{A}}$*), then we can see that the first line of the process will generate* ${g_1}^{\mathcal{A}} \sqsubseteq c^{\mathcal{A}}$ *since c is modified* after *a test on* g_1. *For the same reasons the second and third line will generate* ${g_2}^{\mathcal{A}} \sqsubseteq c^{\mathcal{A}}$, *and generates* $c^{\mathcal{A}} \sqsubseteq {g_1}^{\mathcal{A}}$. *Each of these inequations is generated by an application of the choice rule* $(\mathsf{A}P_{+^{\mathcal{A}}})$. *The recursive call to* s *after the choice has* the same arguments *(that is no argument). Therefore because of Def. 11 the execution stops since* $\langle \mathrm{s}, [] \rangle$ *has already been recorded in* $\mathfrak{H}$. *Now by Def. 12 we can merge the three inequations in order to obtain the skeleton of* s. *From* $c^{\mathcal{A}} \sqsubseteq {g_1}^{\mathcal{A}}$ *and* ${g_1}^{\mathcal{A}} \sqsubseteq c^{\mathcal{A}}$, *it follows that program* s *respects secrecy if* ℓ *fulfills the equality* ${g_1}^{\mathcal{A}} = c^{\mathcal{A}}$. *Otherwise, program* s *may not respect secrecy.*

Now we give the main result of the paper, namely that a program respects secrecy for a privacy map ℓ if its skeleton is compatible with ℓ. The converse is not true. For instance, the process

$$\begin{aligned} \mathrm{q}(x) \Leftarrow & \big([PIN = 12 \Rightarrow SPY := 0]; \mathsf{success}\big) \\ & + \big([PIN \neq 12 \Rightarrow SPY := 0]; \mathsf{success}\big) \end{aligned}$$

is rejected for a privacy map that assigns $\top$ to PIN and $\bot$ to SPY, whereas in fact the final value of SPY does not depend on the actual value of PIN: there is no information flow from high towards low levels.

Recall that a process respects secrecy if it is bisimilar to itself for every privacy level. Thus by contradiction, if secrecy is not respected, there is a privacy level π, such that for two stores F_0, F_1 with $F_0 \cong^{\ell}_{\pi} F \cong^{\ell}_{\pi} F_1$, there exists an execution path leading from $\langle F_0, \mathsf{p} \rangle$ to a store $F^{\#}$ such that there is no path leading from $\langle F_1, \mathsf{p} \rangle$ to a store $\pi\ell$-equivalent to $F^{\#}$. Intuitively, this is impossible if ℓ is compatible with $\mathsf{p}^{\natural}_{F}$, because of the following two cases: either all guards are lower than π, and thus both stores allow the process to proceed with the same actions, or a guard is higher than π but then elementary actions modify the store only in a region higher than π and the transformation is not visible with respect to $\pi\ell$-equivalence. We get the following theorem:

Theorem 1. *Let* $\mathcal{S} = \langle F, \mathbb{\Pi R} \rangle$ *be a system,* p *a program on* $\mathcal{S}$, ℓ *a privacy map for* F, $\mathsf{p}^{\natural}_{F}$ *the skeleton of program* p, *then if* $\mathsf{p}^{\natural}_{F}$ *is compatible with* ℓ *then* p *respects secrecy for* ℓ.

5. RELATED WORK

Due to lack of space we cannot compare our analysis to all proposals in the literature, but focus only on some particularly related works. We refer the reader to [20] for a more exhaustive survey of existing work.

In contrary to [2, 5, 23, 25], our framework allows the dynamic creation of parallel (possibly non-terminating) processes, a feature very common in real-life programming, e.g., servers handling requests by separate processes (or threads). Furthermore, we can deal with general sequential composition, including programs of the form $(P \parallel Q)$; R, which are not possible in [5, 21, 23, 25]. Notice that processes of the form $(P \parallel Q)$; R are difficult to deal with using type systems. Indeed, in order to have the subject reduction property, it is necessary to introduce a process $\mathsf{success}$, such that $(\mathsf{success} \parallel P) \simeq P$ where $\mathsf{success}$ may have any type (from a secrecy point of view). This implies the loss of type unicity for observationally equivalent processes, since one can always add $\mathsf{success}$ processes within any process, obtaining an observationally equivalent process, and by giving an arbitrary type to $\mathsf{success}$ modify the type of the whole process. This may lead to technical difficulties to prove safety. Our proposition, based on an abstract interpretation, avoids such problems. Nevertheless, our approach retains a key property of type based systems, namely compositionality. It is possible to analyze separately different programs yielding constraint sets that can be mixed in an appropriate way (depending on the composition used) in order to obtain the analysis of the composition. In [29], Zanotti uses also abstract interpretation for the systematic derivation of secrecy type systems, but does not consider concurrent programming.

The computation model of [18] uses semaphores for synchronization and, as most other work, `while`-loops instead of recursion as in our model. However, the loop-guards of [18, 21] are limited to low privacy levels. Hence our analysis is more precise in this point.

The approach to secrecy based on type systems has also been applied to object oriented languages [3, 16]. However, we are not aware of extensions of these works dealing with concurrency. [28] presents a new approach to secrecy analysis based on linear bisimulation in the context of the linear π-calculus. The comparison of their results with ours remains to be done.

In [9, 10], the authors consider a computation model which is similar to ours. However, their work departs from ours, since their computation model, as well as their analysis and even their notion of interference are probabilistic. This enables their analysis to measure the amount of information flow. On the other hand, [9, 10] consider only *finite* executions, whereas we can deal with non-terminating programs (cf. Ex. 2).

The analysis presented in this paper does not address information flow due to peculiar scheduling policies. Different approaches have been studied in this direction: *probabilistic* non-interference, *scheduler encoding* as meta-process and program *transformations*. The propositions for the latter we are aware of (i.e., [1, 18, 21]) do not allow high guards in loops, and extensions to more than two privacy levels seem not straightforward. However, the extension of the model integrating the specification of scheduling policies as in [5, 21] seems interesting[3]. Our computation model containing both, dynamic process creation and hierarchic process structures instead of a "flattened" set of concurrent processes, the techniques need to be adapted. The use of probabilistic noninterference [15] as in [23, 24, 27], allows to ensure that the probability distributions for the values of public data do not depend on secret data. This path also merits further investigation.

6. CONCLUSION

In this paper we have presented a new secrecy analysis, which ensures that no information may flow from a secret area towards a public one. This idea of "no information flow from high to low", as shown in [13], may be used for many security issues. With respect to existing work, our analysis considers a richer computation model and uses a different technique, namely abstract computations and constraint solving instead of type systems.

Notice that our algorithm is general in the sense that it can be adapted to cope with new actions, as for instance actions which create new symbols in the store [22]. It suffices to add the appropriate definitions of $\hookrightarrow^{\mathcal{A}}$. In addition, it is easy to see that we can tune our algorithm to handle programs written in several other programming languages, e.g., Constraint Logic Programming, Concurrent Constraint Programming, Rewriting-based languages, LOTOS, SDL etc.

7. REFERENCES

[1] J. Agat. Transforming out timing leaks. In *Proceedings of the 27th ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages (POPL 2000)*, pages 40 – 53, Boston, Jan. 2000. ACM.

[2] J.-P. Banâtre, C. Bryce, and D. L. Métayer. Compile-time detection of information flow in sequential programs. In D. Gollmann, editor, *Computer Security, Proceedingd of the 3rd European Symposium on Research in Computer Security (ESORICS 94)*, volume 875 of *Lecture Notes in Computer Science*, pages 55 – 73, Brighton, Nov. 1994. Springer Verlag.

[3] A. Banerjee and D. A. Naumann. Secure information flow and pointer confinement in a java-like language. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, Cape Breton, 2002.

[4] D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations. Technical Report ESD-TR-73-278 (Vol. I – III), MITRE Corporation, Bedford, Apr. 1974.

[5] G. Boudol and I. Castellani. Noninterference for concurrent programs and thread systems. *Theoretical Computer Science*, 281(1):109 – 130, 2002. Special issue: "Merci, Maurice, A mosaic in honour of Maurice Nivat" (P.-L. Curien, Ed.).

[6] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages (POPL '77)*, pages 238 – 252, Los Angeles, Jan. 1977. ACM.

[7] D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504 – 513, July 1977.

[8] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, chapter 6, pages 243 – 320. Elsevier, Amsterdam, 1990.

[9] A. Di Pierro, C. Hankin, and H. Wiklicky. Probabilistic confinement in a declarative framework. *Electronic Notes in Theoretical Computer Science*, 48, 2001.

[10] A. Di Pierro, C. Hankin, and H. Wiklicky. Approximate non-interference. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, Cape Breton, June 2002.

[11] R. Echahed and W. Serwe. Combining mobile processes and declarative programming. In J. Lloyd et al., editors, *Proceedings of the 1st International Conference on Computational Logic (CL 2000)*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 300 – 314, London, July 2000. Springer Verlag.

[12] R. Echahed and W. Serwe. Integrating action definitions into concurrent declarative programming. *Electronic Notes in Theoretical Computer Science*, 64, Sept. 2002. special issue: selected papers of the International Workshop on Functional and (Constraint) Logic Programming (WFLP 2001).

[13] R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In U. Montanari, J. D. P. Rolim, and E. Welzl, editors, *Proceeding of the 27th International Colloquium on Automata, Languages and Programming (ICALP 2000)*, volume 1853 of *Lecture Notes in Computer Science*, pages 354 – 372, Geneva, July 2000. Springer Verlag.

[14] W. Fokkink. *Introduction to Process Algebra*. Texts in Theoretical Computer Science. Springer Verlag, 2000.

[15] J. W. Gray, III. Probabilistic interference. In *Proceedings of the 1990 IEEE Symposium on Security and Privacy*, pages 170 – 179, Oakland, May 1990. IEEE Computer Society Press.

[16] A. C. Myers. JFlow: Practical mostly-static information flow control. In *Proceedings of the 26th ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages (POPL '99)*, pages 228 – 241, San Antonio, Jan. 1999.

[17] F. Prost. A static calculus of dependencies for the λ-cube. In *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science (LICS '2000)*, pages 267 – 276, Santa Barbara, 2000. IEEE Computer Society Press.

[18] A. Sabelfeld. The impact of synchronisation on secure information flow in concurrent programs. In D. Bjørner, M. Broy, and A. V. Zamulin, editors, *Perspectives of System Informatics, Proceedings of the 4th International Andrei Ershov Memorial Conference*

[3]The differences between our bisimulation and the strong low-bisimulation of [21] needs further study. Notably: does our bisimulation already imply *scheduler independent noninterference*?

(PSI 2001), volume 2244 of *Lecture Notes in Computer Science*, pages 225 – 239, Akademgorodok, July 2001. Springer Verlag.

[19] A. Sabelfeld and H. Mantel. Static confidentiality enforcement for distributed programs. In M. V. Hermenegildo and G. Puebla, editors, *Proceedings of the 9th International Symposium on Static Analysis (SAS 2002)*, volume 2477 of *Lecture Notes in Computer Science*, pages 376 – 394, Madrid, Sept. 2002. Springer Verlag.

[20] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications, special issue on Design and Analysis Techniques for Security Assurance*, 2002. to appear.

[21] A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 200 – 214, Cambridge, July 2000.

[22] W. Serwe. *On Concurrent Functional-Logic Programming.* thèse de doctorat, Institut National Polytechnique de Grenoble, Mar. 2002.

[23] G. Smith. A new type system for secure information flow. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, pages 115 – 125, Cape Breton, June 2001. IEEE Computer Society Press.

[24] G. Smith. Weak probabilistic bisimulation for secure information flow. In *Proceedings of the Workshop on Issues in the Theory of Security (WITS '02)*, Portland, Jan. 2002.

[25] G. Smith and D. M. Volpano. Secure information flow in a multi-threaded imperative language. In *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '98)*, pages 355 – 364, San Diego, Jan. 1998.

[26] G. Smith, D. M. Volpano, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167 – 187, 1996.

[27] D. M. Volpano and G. Smith. Probabilistic noninterference in a concurrent language. *Journal of Computer Security*, 7(2, 3):231 – 253, Nov. 1999.

[28] N. Yoshida, K. Honda, and M. Berger. Linearity and bisimulation. In M. Nielsen and U. Engberg, editors, *Proceedings of the 5th International Conference on Foundations of Software Science and Computer Structures (FoSSaCs 2002)*, volume 2303 of *Lecture Notes in Computer Science*, pages 417 – 434. Springer Verlag, Apr. 2002.

[29] M. Zanotti. Security typings by abstract interpretation. In M. V. Hermenegildo and G. Puebla, editors, *Proceedings of the 9th International Symposium on Static Analysis (SAS 2002)*, volume 2477 of *Lecture Notes in Computer Science*, pages 360 – 375, Madrid, Sept. 2002. Springer Verlag.

APPENDIX

A. PROOFS OMITTED FROM THE PAPER (INTENDED FOR THE REVIEWERS)

In this appendix we give the proofs omitted in the paper and introduce further auxiliary lemmas, together with their proofs.

A.1 Independence from Higher Privacy Levels

The following lemma expresses that the evaluation of a term in a *safe* store does not depend from the rules of higher privacy level. We use the following notations. Let t be a term of $T(\Sigma, X)$, ρ a rewrite rule in $\mathcal{R}$ and $\mathbf{p}$ a position in t, then a rewrite step at position $\mathbf{p}$ using rule ρ is written $t \dashrightarrow_{\mathbf{p},\rho} t'$.

LEMMA 1. *Let π be a privacy level, ℓ be a privacy map, $F_1 = \langle \Sigma, \mathcal{R}_1 \rangle, F_2 = \langle \Sigma, \mathcal{R}_2 \rangle$ be two safe stores such that $F_1 \cong^{\ell}_{\pi} F_2$ and t be a term of $T(\Sigma, \varnothing)$ such that $\ell(t) \sqsubseteq \pi$. If there is a reduction step $t \dashrightarrow_{\mathbf{p},\rho_1} t'$ with $\mathbf{p}$ a position and $\rho_1 \in \mathcal{R}_1$ then there exists t'' such that $t \dashrightarrow_{\mathbf{p},\rho_2} t''$ with $\rho_2 \in \mathcal{R}_2$, t' is equal to t'' up to renaming and $\ell(t') = \ell(t'') \sqsubseteq \pi$.*

PROOF. The safety of the stores F_1 and F_2 implies the safety of their rewrite rules. The privacy level of a rewrite rule $l \to r \mid c$ is $\ell(l \to r \mid c) = \ell(l)$. Now since the privacy level of a term is the least upper bound of the privacy levels of its subterms, if rule ρ_1 can be used at position $\mathbf{p}$, then the subterm at this position has a privacy level lower than π, and the level of ρ_1 is also lower than π (indeed the privacy levels of variables that could occur in l are $\perp$). Hence the privacy levels of all rules (notice that due to conditions in the rewrite rules, more than one rule can be used in a reduction step) used in the reduction step $t \dashrightarrow_{\mathbf{p},\rho_1} t'$ are lower or equal to π. By definition of $\cong^{\ell}_{\pi}$ we have thus for each of these rules ($\in \mathcal{R}_1$) the existence of a rule ($\in \mathcal{R}_2$) which is equal up to renaming. Thus the reduction steps are with respect to the same stores (modulo renaming of the variables in the rewrite rules). $\square$

A.2 Characterization of Nonbisimilar Processes

The following lemma characterizes a process which is not bisimilar to itself on equivalent stores. Intuitively, the statement corresponds to the negation of Def. 8. We use this lemma to prove Theorem 1.

LEMMA 2. *Let ℓ be a privacy map, π a privacy level, p_0 a process term and F^1_0, F^2_0 two stores. $\langle F^1_0, p_0 \rangle \not\approx^{\ell}_{\pi} \langle F^2_0, p_0 \rangle$ implies that there exist an integer N, and two derivations:*

$$\begin{array}{l} \langle F^0_0, p_0 \rangle \longrightarrow \langle F^0_1, p_1 \rangle \longrightarrow \ldots \longrightarrow \langle F^0_N, p_N \rangle \\ \langle F^1_0, p_0 \rangle \longrightarrow \langle F^1_1, p_1 \rangle \longrightarrow \ldots \longrightarrow \langle F^1_N, p_N \rangle \end{array}$$

such that for all $j < N$, $F^0_j \cong^{\ell}_{\pi} F^1_j$ and

- *$F^0_N \not\cong^{\ell}_{\pi} F^1_N$,*
- *or there exist $F^{\#}$, $p^{\#}$ such that (for $j \in \{0, 1\}$)*

$$\langle F^j_N, p_N \rangle \longrightarrow \langle F^j_{N+1}, p_{N+1} \rangle \longrightarrow^* \langle F^{\#}, p^{\#} \rangle$$

but $\langle F^{1-j}_N, p_N \rangle \not\longrightarrow$ and $F^{1-j}_N \not\cong^{\ell}_{\pi} F^{\#}$.

PROOF. Intuitively, Lemma 2 is simply the negation of Def. 8. We prove Lemma 2 by considering the different cases with respect to the length of the transition sequences.

Without loss of generality, consider, in the situation of Lemma 2, a *maximal* transition sequence d_0 starting with F^0_0 and p_0, i.e., a sequence of transitions that can not be extended. We distinguish between the case of a finite and an infinite transition sequence.

1. The maximal transition sequence d_0 is *finite*, i.e., we have $n \geq 0$ such that
$\langle F_0^0, p_0\rangle \longrightarrow \langle F_1^0, p_1\rangle \longrightarrow \cdots \longrightarrow \langle F_n^0, p_n\rangle \not\longrightarrow$.
We prove Lemma 2 by induction on the length n of d_0.
Base Case. In the case that $n = 0$, if we have $F_0^0 \not\cong_\pi^\ell F_0^1$ the lemma is obviously true (for $N = 0$). Otherwise, by definition of $\approx_\pi^\ell$, there exist $F^\sharp$, $p^\sharp$ such that $\langle F_0^1, p_0\rangle \longrightarrow^+ \langle F^\sharp, p^\sharp\rangle$ with $F_0^0 \not\cong_\pi^\ell F^\sharp$. Thus the lemma holds for $N = 0$.
Induction Step. Suppose now, that Lemma 2 holds for maximal transition sequences for $\langle F_0^0, p_0\rangle$ of length shorter than n. Consider a maximal transition sequence of length n. If $F_0^0 \not\cong_\pi^\ell F_0^1$, the lemma is obviously true (for $N = 0$). Otherwise we distinguish two further cases:
   - $\langle F_0^1, p_0\rangle \not\longrightarrow$. Assume that $F_0^1 \cong_\pi^\ell F_i^0$ for all $i \in \{1, \ldots, n\}$. Thus we have by definition 8 that $\langle F_0^0, p_0\rangle \approx_\pi^\ell \langle F_0^1, p_0\rangle$ which is in contradiction to our assumption. Thus there exists i_0 such that $\langle F_0^0, p_0\rangle \longrightarrow \langle F_1^0, p_0\rangle \longrightarrow^* \langle F_{i_0}^0, p_{i_0}^0\rangle$ but $\langle F_0^1, p_0\rangle \not\longrightarrow$ and $F_{i_0}^0 \not\cong_\pi^\ell F_0^1$, i.e., we are in the second situation of Lemma 2 for $N = 0$.
   - There exists p_0' such that[4] $\langle F_0^1, p_0\rangle \longrightarrow \langle F_1^1, p_0'\rangle$. In this case, we have the following transition[5] $\langle F_0^1, p_0\rangle \longrightarrow \langle F_1^1, p_1\rangle$. Consequently we have also $\langle F_1^0, p_1\rangle \not\approx_\pi^\ell \langle F_1^1, p_1\rangle$ and can apply the hypothesis of the induction, since the considered maximal transition sequence for $\langle F_1^0, p_1\rangle$ has length $n - 1$.
2. The maximal transition sequence d_0 is *infinite*, i.e., we have
$\langle F_0^0, p_0\rangle \longrightarrow \langle F_1^0, p_1\rangle \longrightarrow \cdots \longrightarrow \langle F_i^0, p_i\rangle \longrightarrow \cdots$.
Consider the maximal transition sequence d_1 for F_0^1 and p_0 which corresponds to an execution of (a prefix of) the same sequence of actions. If d_1 is finite, the proof is symmetric to case 1. Thus we suppose we have an infinite transition sequence[6] $\langle F_0^1, p_0\rangle \longrightarrow \langle F_1^1, p_1\rangle \longrightarrow \cdots \longrightarrow \langle F_i^1, p_i\rangle \longrightarrow \cdots$.
Notice that we cannot have $F_i^0 \cong_\pi^\ell F_i^1$ for all $i \geq 0$ since this implies that $\langle F_0^0, p_0\rangle \approx_\pi^\ell \langle F_0^1, p_0\rangle$, in contrary to our assumption. Thus, we choose N as the least index such that $F_n^0 \not\cong_\pi^\ell F_n^1$.

□

A.3 Correspondence Between Concrete and Abstract Executions

The correspondence between concrete and abstract executions is established by the following two lemmas. The first lemma states that the execution of abstract action modifies the abstract store in a monotonic manner, that is to say by only adding constraints to the abstract store.

LEMMA 3. *We have that $\langle F, \mathsf{a}; a\rangle \hookrightarrow \langle F', a\rangle$ implies that for all σ and constraint sets C there exists $C' \supseteq C$ such that $\langle F^{\mathcal{A}} \cup C, \langle \mathsf{a}; a, \sigma\rangle\rangle \hookrightarrow^{\mathcal{A}} \langle F'^{\mathcal{A}} \cup C', \langle a, \sigma\rangle\rangle$.*

PROOF. We analyze our four elementary actions separately.

skip**:** According to $(ea_{\mathsf{skip};})$, $F' = F$. We choose $C' = C$ and Lemma 3 holds due to $(\mathsf{A}ea_{\mathsf{skip};})$.

$\mathsf{tell}(l \to r \mid c)$**:** Inspection of $(\mathsf{A}ea_{\mathsf{tell}})$ and (ea_{tell}) shows that Lemma 3 holds for $C' = C \cup \{\sigma \sqsubseteq l^{\mathcal{A}}\}$, since the further inequations added to $F^{\mathcal{A}}$ are exactly those corresponding to the rule added to F.

$\mathsf{del}(l \to r \mid c)$**:** We distinguish two situations:
On the one hand, if $l \to r \mid c \notin F$, then we have by (ea_{del}) that $F' = F$, and inspection of $(\mathsf{A}ea_{\mathsf{del}})$ shows Lemma 3 holds for $C' = C \cup \{\sigma \sqsubseteq l^{\mathcal{A}}\}$.
On the other hand, if $l \to r \mid c \in F$, then we have that $F'^{\mathcal{A}} = F^{\mathcal{A}} \cup \{r^{\mathcal{A}} \sqsubseteq l^{\mathcal{A}}; c^{\mathcal{A}} \sqsubseteq l^{\mathcal{A}}\}$. Hence Lemma 3 holds by choosing
$C' = C \cup \{r^{\mathcal{A}} \sqsubseteq l^{\mathcal{A}}; c^{\mathcal{A}} \sqsubseteq l^{\mathcal{A}}\} \cup \{\sigma \sqsubseteq l^{\mathcal{A}}\}$.

$f := t$**:** By a similar reasoning as for tell and del[7], inspection of $(\mathsf{A}ea_{:=})$ and $(ea_{:=})$ shows that Lemma 3 holds for $C' = C \cup D \cup \{\sigma \sqsubseteq f^{\mathcal{A}}\}$, where D is defined as the following set of privacy inequations
$D = \{r \sqsubseteq l; c \sqsubseteq l \text{ such that } \exists (l \to r \mid c) \in F \setminus F'\}$.

□

Before we can state the correspondence between concrete and abstract transitions, we introduce the function ϕ which associates to an abstract process term $\mathcal{M}$ a corresponding concrete process term by omitting all privacy levels. ϕ is defined by the following four equations:

$$\phi(\langle p, \sigma\rangle) = p \qquad \phi(\mathcal{M} \|^{\mathcal{A}} \mathcal{M}') = \phi(\mathcal{M}) \parallel \phi(\mathcal{M}')$$
$$\phi(\mathcal{M} +^{\mathcal{A}} \mathcal{M}') = \phi(\mathcal{M}) + \phi(\mathcal{M}')$$
$$\phi(\mathcal{M} \,;^{\mathcal{A}} \mathcal{M}') = \phi(\mathcal{M}); \phi(\mathcal{M}')$$

The following lemma states that for each concrete transition exists a corresponding abstract transition.

LEMMA 4. *Let $\mathcal{S} = \langle F, \Pi\!\mathrm{R}\rangle$ and p a process term of $\mathcal{S}$. If $\langle F, p\rangle \longrightarrow \langle F', p'\rangle$ then for all $\mathcal{M}$ such that $\phi(\mathcal{M}) = p$ and for all constraint sets C we have $\mathcal{M}'$, C' such that $\langle F^{\mathcal{A}} \cup C, \overline{\mathcal{M}}^{\mathsf{nf}}\rangle \longrightarrow^{\mathcal{A}} \langle F'^{\mathcal{A}} \cup C', \mathcal{M}'\rangle$, $\phi(\mathcal{M}') \equiv_p p'$ and $C \subseteq C'$.*

PROOF. We prove Lemma 4 by induction of the height of the inference tree used to infer the concrete transition $\langle F, p\rangle \longrightarrow \langle F', p'\rangle$. That is to say, we prove for all inference rules for $\longrightarrow$ that, if Lemma 4 holds for the premises, than it also holds for the conclusion of the inference rule.

Base Case. The only inference rule for $\longrightarrow$ without any occurrence of $\longrightarrow$ in the premise is rule $(\mathsf{A}P_{guard})$. Notice first that $\phi(\mathcal{M}) = [g \Rightarrow \mathsf{a}_1; \ldots; \mathsf{a}_n]$ implies that $\mathcal{M} = \overline{\mathcal{M}}^{\mathsf{nf}}$ and $\mathcal{M}$ is of the form $\langle [g \Rightarrow \mathsf{a}_1; \ldots; \mathsf{a}_n], \sigma\rangle$, where σ is a privacy level. Applying Lemma 3 n times, we have that $\langle F, \mathsf{a}_1; \ldots; \mathsf{a}_n; \mathsf{skip}\rangle \hookrightarrow \langle F', \mathsf{skip}\rangle$ implies that for all σ and constraint sets C there exists $C' \supseteq C$ such that

$$\langle F^{\mathcal{A}} \cup C, \langle \mathsf{a}_1; \ldots; \mathsf{a}_n; \mathsf{skip}, \sigma\rangle\rangle \hookrightarrow^{\mathcal{A}} \langle F'^{\mathcal{A}} \cup C', \langle \mathsf{skip}, \sigma\rangle\rangle$$

Defining $\mathcal{M}' = \langle \mathsf{success}, \sigma\rangle$, we have by rule $(\mathsf{A}P_{guard})$ that Lemma 4 holds.

Induction Step. We consider the remaining inference rules one by one, under the hypothesis that lemma 4 holds for the transitions occurring in the premise.

[4]The transition yields the store F_1^1 since the execution of actions is deterministic.
[5]Since the store and the process term are the same, we just have to use the same inference rules to infer this transition.
[6]We have the same process terms in the transition sequences d_0 and d_1 by a similar reasoning as above (see footnote 5).
[7]Notice that assignment is a combination of del and tell.

$(P_{\equiv_p})$: Suppose that the premises of rule $(P_{\equiv_p})$ hold, i.e., that $p_1 \equiv_p p_2$, $p_3 \equiv_p p_4$ and $\langle F, p_2\rangle \longrightarrow \langle F', p_3\rangle$. Using the hypothesis of the induction, we have thus that for all $\mathcal{M}_2$ and C such that $\phi(\mathcal{M}_2) = p_2$ there exist $\mathcal{M}_3$ and C' such that $\langle F^{\mathcal{A}}, \overline{\mathcal{M}_2}^{\mathsf{nf}}\rangle \longrightarrow^{\mathcal{A}} \langle F'^{\mathcal{A}}, \mathcal{M}_3\rangle$, $\phi(\mathcal{M}_3) \equiv_p p_3$ and $C \subseteq C'$. Since $p_1 \equiv_p p_2$, we can, for any $\mathcal{M}_1$ such that $\phi(\mathcal{M}_1) = p_1$, choose $\mathcal{M}_2$ such that $\overline{\mathcal{M}_2}^{\mathsf{nf}} \equiv^{\mathcal{A}} \overline{\mathcal{M}_1}^{\mathsf{nf}}$ and $\phi(\mathcal{M}_2) = p_2$. Applying the hypothesis of the induction, we have for all C that there exist $\mathcal{M}_3$ and C' such that $\langle F^{\mathcal{A}}, \overline{\mathcal{M}_2}^{\mathsf{nf}}\rangle \longrightarrow^{\mathcal{A}} \langle F'^{\mathcal{A}}, \mathcal{M}_3\rangle$, $\phi(\mathcal{M}_3) = p_3$ and $C \subseteq C'$. Defining $\mathcal{M}_4 = \overline{\mathcal{M}_3}^{\mathsf{nf}}$, we have obviously $\phi(\mathcal{M}_4) \equiv_p p_4$. Thus by application of rule $(\mathsf{A}P_{\equiv_{\mathcal{A}}})$, Lemma 4 holds.

$(P_{\|})$: The premise of $(P_{\|})$ is $\langle F, p_1\rangle \longrightarrow \langle F', p_1'\rangle$. Hence, according to the hypothesis of the induction, for all C and $\mathcal{M}_1$ such that $\phi(\mathcal{M}_1) = p_1$, we have that there exist $\mathcal{M}_1'$ and C' such that $\langle F^{\mathcal{A}}, \overline{\mathcal{M}_1}^{\mathsf{nf}}\rangle \longrightarrow^{\mathcal{A}} \langle F'^{\mathcal{A}}, \mathcal{M}_1'\rangle$, $\phi(\mathcal{M}_1') = p_1'$ and $C \subseteq C'$. Hence we have by rules $(\mathsf{A}P_{\|^{\mathcal{A}}})$ and $(\mathsf{A}P_{\equiv_{\mathcal{A}}})$ an abstract transition $\langle F^{\mathcal{A}}, \mathcal{M}_1 \|^{\mathcal{A}} \mathcal{M}_2\rangle \longrightarrow^{\mathcal{A}} \langle F'^{\mathcal{A}}, \mathcal{M}_1' \|^{\mathcal{A}} \mathcal{M}_2\rangle$ for all $\mathcal{M}_2$, and in particular, for all $\mathcal{M}_2$ such that $\phi(\mathcal{M}_1 \|^{\mathcal{A}} \mathcal{M}_2) = p_1 \| p_2$, which proves Lemma 4.

(P_{abs}), $(P_;)$, or (P_+): These cases are proven in a similar way as for rule $(P_{\|})$.

□

A.4 "No Low Actions after High Guards"

The following lemma relates the level of actions executed by a process to the level of guards. If the skeleton of a program is compatible with a privacy map then it implies that actions following a guard of privacy level π operate on data of a privacy level higher than π.

LEMMA 5. *Let $\mathcal{S} = \langle F_0, \Pi R\rangle$ be a system, $\mathsf{p} = p_0$ a program on $\mathcal{S}$, ℓ a privacy map for F, $\mathsf{p}^{\natural}_{F_0}$ the skeleton of program p_0 and $\mathsf{p}^{\natural}_{F_0}$ be compatible with ℓ. Consider a transition sequence $\langle F_0, p_0\rangle \xrightarrow{\alpha_1} \langle F_1, p_1\rangle \longrightarrow \cdots \longrightarrow \langle F_n, p_n\rangle$, where $\alpha_1 = [g^1 \Rightarrow \mathsf{a}^1, \ldots, \mathsf{a}^1_{n_1}]$, we have that $\ell(g^1) \sqsubseteq \ell(\alpha_i)$ for all $i \geq 0$.*

PROOF. We reason by contradiction. Using lemma 4, consider an abstract transition sequence corresponding to the concrete transition sequence above

$$\begin{aligned}\langle F_0{}^{\mathcal{A}} \cup C_0, \mathcal{M}\rangle \quad &\longrightarrow^{\mathcal{A}} \langle F_1{}^{\mathcal{A}} \cup C_1, \mathcal{M}_1\rangle\\ &\longrightarrow^{\mathcal{A}} \ldots\\ &\longrightarrow^{\mathcal{A}} \langle F_n{}^{\mathcal{A}} \cup C_n, \mathcal{M}_n\rangle\end{aligned}$$

where $\phi(\mathcal{M}_i) \equiv_p p_i$ for $i \in \{0, \ldots, n\}$ and C_0 is an arbitrary set of privacy inequations. Suppose that there is an $i_0 \in \{1, \ldots, n\}$ such that $\ell(\alpha_{i_0}) \sqsubset \ell(g^1)$.

By inspection of rule $(\mathsf{A}P_{guard})$ and because the skeleton $\mathsf{p}^{\natural}_{F_0}$ is compatible with ℓ, we have that $\ell(g^1) \sqsubseteq \sigma$ for all σ occurring in $\mathcal{M}_i$ for $i \in \{1, \ldots, n\}$. Let α_{i_0} be $[g^{i_0} \Rightarrow \mathsf{a}^{i_0}_1; \ldots; \mathsf{a}^{i_0}_{n_{i_0}}]$. Thus we have an abstract transition corresponding to the execution of α_{i_0}:
$\langle F^{\mathcal{A}}, \mathsf{a}^{i_0}_1; \ldots; \mathsf{a}^{i_0}_{n_{i_0}}; \mathsf{skip}, \sigma_{i_0}\rangle \hookrightarrow^{\mathcal{A}} {*} \langle F^{\mathcal{A}\prime}, \mathsf{skip}, \sigma_{i_0}\rangle$.

We conclude from $\ell(\alpha_{i_0}) \sqsubset \ell(g)$ that there exists $j \in \{1, \ldots, n_{i_0}\}$ such that $\ell(\mathsf{a}^{i_0}_j) \sqsubset \ell(g)$. We consider the different possibilities for the elementary action $\mathsf{a}^{i_0}_j$ one by one.

$\mathsf{tell}(l \rightarrow r \mid c)$: According to rule $(\mathsf{A}ea_{\mathsf{tell}})$, $\mathsf{p}^{\natural}_{F_0}$ implies $\sigma_{i_0} \sqsubseteq \ell(l)$, which is in contradiction to the hypothesis, since $\ell\big(\mathsf{tell}(l \rightarrow r \mid c)\big) = \ell(l)$.

$\mathsf{del}(l \rightarrow r \mid c)$: According to rule $(\mathsf{A}ea_{\mathsf{del}})$, $\mathsf{p}^{\natural}_{F_0}$ implies $\sigma_{i_0} \sqsubseteq \ell(l)$, which is in contradiction to the hypothesis, since $\ell\big(\mathsf{del}(l \rightarrow r \mid c)\big) = \ell(l)$.

$(f := t)$: According to rule $(\mathsf{A}ea_{:=})$, $\mathsf{p}^{\natural}_{F_0}$ implies $\sigma_{i_0} \sqsubseteq \ell(f)$, which is in contradiction to the hypothesis, since $\ell(f := t) = \ell(f)$.

skip: This is impossible, since by definition $\ell(\mathsf{skip}) = \top$.

□

A.5 Strong Normalization of $\rightsquigarrow$

THEOREM 2. *Relation $\rightsquigarrow$ is strongly normalizing.*

PROOF. The first thing to notice is that if there is an infinite $\longrightarrow^{\mathcal{A}}$ reduction sequence then there is an infinite number of reduction steps using rule $(\mathsf{A}P_{abs})$. If it were not the case then there would be an infinite reduction sequences where rule $(\mathsf{A}P_{abs})$ is not used, but it is not possible since for each $\longrightarrow^{\mathcal{A}}$ rule different from $(\mathsf{A}P_{abs})$ the size of $\mathcal{M}$ terms decreases.

Now suppose that there is an infinite number of reduction steps using rule $(\mathsf{A}P_{abs})$. Since the size of the store is finite, so is the number of terms like $\ell^{\mathcal{A}}(t)$. Indeed, the definition of $\ell^{\mathcal{A}}$ (see Def. 10) uses the idempotent operator $\sqcup$. Finally the number of process definition is also finite, therefore the number of couples of the form $\langle \mathrm{q}, [t_1{}^{\mathcal{A}}; \ldots; t_n{}^{\mathcal{A}}]\rangle$ is finite too (note that the arity of a process is fixed). Thus there exists a natural N such that after a reduction sequence of size N a couple of the form $\langle \mathrm{q}, [t_1{}^{\mathcal{A}}; \ldots; t_n{}^{\mathcal{A}}]\rangle$ has already been integrated in $\mathfrak{H}$, and by definition of $\rightsquigarrow$ it yields the process $\mathsf{success}$, hence no longer $\longrightarrow^{\mathcal{A}}$ reduction step can be executed. Thus there can be no infinite number of reduction steps using rule $(\mathsf{A}P_{abs})$. □

A.6 Main Theorem

THEOREM 1. *Let $\mathcal{S} = \langle F, \Pi R\rangle$ be a system, p a program on $\mathcal{S}$, ℓ a privacy map for F, $\mathsf{p}^{\natural}_F$ the skeleton of program p, then if $\mathsf{p}^{\natural}_F$ is compatible with ℓ then p respects secrecy for ℓ.*

PROOF. We reason by contradiction. Suppose that p does not respect secrecy for ℓ. Def. 9 implies that it exists a privacy level π, two stores F_0, F_1 such that $F_0 \cong^{\ell}_{\pi} F_1 \cong^{\ell}_{\pi} F$ and $\langle F_0, \mathsf{p}\rangle \not\approx^{\ell}_{\pi} \langle F_1, \mathsf{p}\rangle$. Now from Lemma 2 we have that there exist an integer N and two derivations (with $p_0 = \mathsf{p}$):

$$\begin{aligned}\langle F^0_0, p_0\rangle \longrightarrow \langle F^0_1, p_1\rangle \longrightarrow \ldots \longrightarrow \langle F^0_N, p_N\rangle\\ \langle F^1_0, p_0\rangle \longrightarrow \langle F^1_1, p_1\rangle \longrightarrow \ldots \longrightarrow \langle F^1_N, p_N\rangle\end{aligned}$$

such that for all $j < N$, $F^0_j \cong^{\ell}_{\pi} F^1_j$ and we have two cases:

1. Either $F^0_N \not\cong^{\ell}_{\pi} F^1_N$,
2. or there exists $F^{\#}, p^{\#}$ such that for $j \in \{0, 1\}$:
   - $\langle F^j_N, p_N\rangle \xrightarrow{\alpha} \langle F^j_{N+1}, p_{N+1}\rangle \longrightarrow^* \langle F^{\#}, p^{\#}\rangle$ and $\langle F^{1-j}_N, p_N\rangle \not\xrightarrow{\alpha}$,
   - $F^{1-j}_N \not\cong^{\ell}_{\pi} F^{\#}$.

The first case corresponds to the analysis of [25]. A contradiction can be derived using Lemma 4, and the fact that the program skeleton is satisfied. Indeed if $F^0_N \not\cong^{\ell}_{\pi} F^1_N$, then it is because the last transition is done on a process term of the form $[g \Rightarrow \mathsf{a}_1; \ldots ; \mathsf{a}_n]$. We have

$$\begin{array}{rcl}\langle F^j_{N,0}, \mathsf{a}_1; \ldots ; \mathsf{a}_n; \mathsf{skip}\rangle & \hookrightarrow & \langle F^j_{N,1}, \mathsf{a}_2; \ldots ; \mathsf{a}_n; \mathsf{skip}\rangle \\ & \hookrightarrow & \langle F^j_{N,2}, \mathsf{a}_3; \ldots ; \mathsf{a}_n; \mathsf{skip}\rangle \\ & \vdots & \\ & \hookrightarrow & \langle F^j_{N,n}, \mathsf{skip}\rangle\end{array}$$

where $F^j_{N,0} = F^j_N$ for $j \in \{0, 1\}$.

Thus there must exist an elementary action a_i (for $i \in \{1; \ldots ; n\}$) that transforms two different $\pi\ell$-equivalent stores into two non $\pi\ell$-equivalent stores. On the other hand, thanks to Lemmas 3 and 4 we can mimic these reductions on the abstract level.

$\mathsf{tell}(\rho)$: The abstract action $\mathsf{tell}(\rho^{\mathcal{A}})$ has been executed during the analysis (because of Lemma 4), i.e., the computation of the constraints $\mathsf{p}^{\natural}_F$. We distinguish the following two cases:

- **$\rho^{\mathcal{A}} \sqsubseteq \pi$:** The rule ρ added to the store is the same for both, $F^0_{N,i-1}$ and $F^1_{N,i-1}$. Thus, we have that $F^0_{N,i} \cong^{\ell}_{\pi} F^1_i$, in contradiction to the assumption.
- **$\pi \sqsubset \rho^{\mathcal{A}}$:** In this case, the rules of a lower or equal privacy level than π are not modified, and thus we cannot have $F^0_{N,i} \ncong^{\ell}_{\pi} F^1_{N,i}$.

$\mathsf{del}(\rho)$: We distinguish two cases:

- **$\rho^{\mathcal{A}} \sqsubseteq \pi$:** In this case, the rule ρ is present in $F^0_{N,i-1}$ if and only if ρ is present in $F^1_{N,i-1}$. Thus the removal of ρ has the same effect, and we have that $F^1_{N,i} \cong^{\ell}_{\pi} F^1_{N,i}$, in contradiction to the assumption.
- **$\pi \sqsubset \rho^{\mathcal{A}}$:** Since the rules of the store which have a lower or equal privacy level than π are not modified by the execution of this action, we have $F^0_{N,i} \cong^{\ell}_{\pi} F^1_{N,i}$, in contradiction to the assumption.

$c := v$: We distinguish the following two cases:

- **$c^{\mathcal{A}} \sqsubseteq \pi$:** Since $F^0_{N,i-1} \cong^{\ell}_{\pi} F^1_{N,i-1}$, we have (by Lemma 1) that $eval(F^0_{i-1}, v) = eval(F^1_{N,i-1}, v)$. Consequently $F^0_{N,i} \cong^{\ell}_{\pi} F^1_{N,i}$, in contradiction to the assumptions.
- **$\pi \sqsubset c^{\mathcal{A}}$:** In this case, the assignment does modify only rules of a higher privacy level than π, and thus we cannot have $F^0_{N,i} \ncong^{\ell}_{\pi} F^1_i$.

The second case corresponds to the extension of [25] done in [5]. In this case the contradiction comes from the following fact: if $F^{\#} \ncong^{\ell}_{\pi} F^{1-j}_N$, then it means that an information of a privacy level *lower or equal* to π has been modified, but on store $1 - j$ an action α cannot have been performed ($\langle F^{1-j}_N, p_N\rangle \not\xrightarrow{\alpha}$) while it is possible on store j. By Lemma 1, we know that this implies that this action is guarded by a guard of privacy level *strictly superior* than π (if it were not the case then the evaluation of the guard would give the same result for both stores). But by Lemma 5 we know that actions following a guard must be defined at a level higher than the guard, hence the contradiction: an action must have been performed on a level lower or equal than π and since the skeleton of the program is satisfied, we have that actions must be done on a strictly higher level than π. $\square$