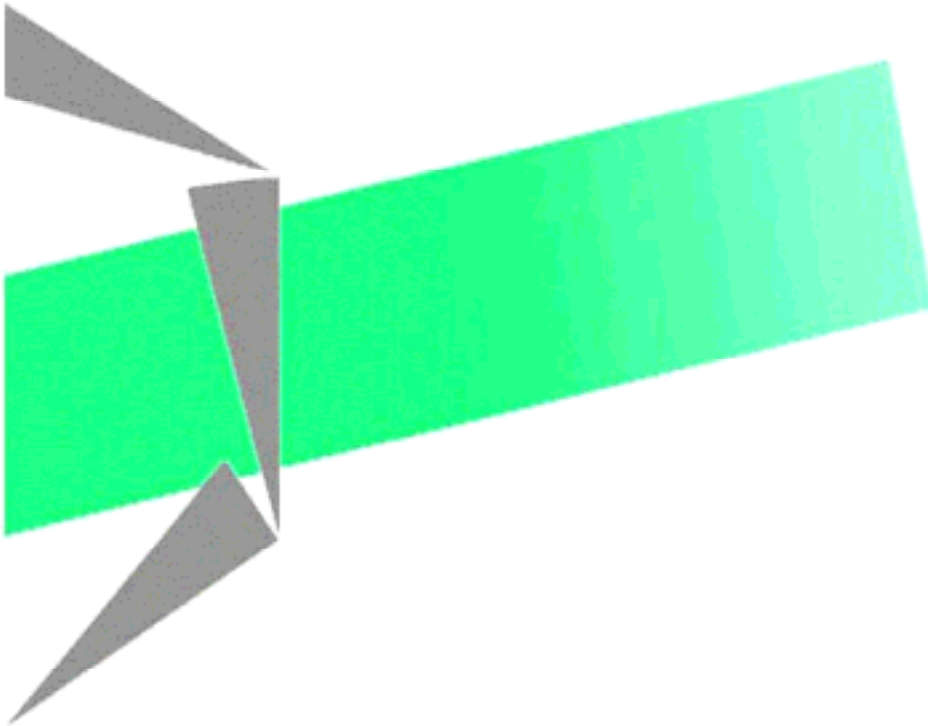


# Les cahiers du laboratoire Leibniz



## Assuring Secrecy for Concurrent Declarative Programs

Rachid Echahed, Frédéric Prost, Wendelin Serwe

Laboratoire Leibniz-IMAG, 46 av. Félix Viallet, 38000 GRENOBLE, France -  
ISSN : 1298-020X

n° 40  
Janv. 2002

Site internet : <http://www-leibniz.imag.fr/LesCahiers/>



# Assuring Secrecy for Concurrent Declarative Programs

Rachid Echahed    Frédéric Prost    Wendelin Serwe  
Laboratoire LEIBNIZ – Institut IMAG, CNRS  
46, avenue Félix Viallet, F-38031 Grenoble, FRANCE

E-mail: {echahed,prost,serwe}@imag.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 towards 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 which includes strictly those considered by Boudol and Castellani or Smith and Volpano.*

## 1. Introduction

### 1.1. Secrecy in Computer Science

The need for secrecy increases as more and more (sensitive) private data (credit cards numbers, personal medical files ...) migrates through the Internet. One needs to ensure that sensitive data remains in some restricted, controlled area. One way to define secrecy from a theoretical point of view is to assign *privacy levels* to data used by a program, high levels denoting highly (sensitive) private data while low levels representing public ones. The aim of secrecy analysis is to show how high and low level data interact with each other. Secrecy is achieved when information may only flow from low levels to higher ones. A program is told *safe* if its execution does not harm secrecy. 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.

This type of approach to secrecy has been first studied in the context of imperative programming in [17]. The extension to concurrent programming is not straightforward as [16] shows. Indeed control-flow may be turned into

information-flow. The idea is that the following program:

```
while (PIN <> 0) skip; end while;  
spy := 0;
```

verifies secrecy because at the end of every execution, the value of variable `spy` is the same, and thus is not apparently influenced by the value of variable `PIN`. Nevertheless, it is shown in [16, 3] that a smart combination of many of such processes allows to gather information about the value of `PIN`. To avoid such cases, it is forbidden in [16] to guard while loops with tests on high level data. This condition is a bit drastic, and has been relaxed by the introduction of a subtler type system in [3]. The idea is that if a guard has a high level of privacy then all following assignments must be performed at a higher or equal level. Therefore the maximal levels of loop guards have to be taken into account for sequential compositions. To implement this idea Boudol and Castellani define in [3] program types as couples where the first component is the upper bound of privacy level of guards, while the second component records the lower bound of privacy level of assigned variables.

### 1.2. Paper Aims

Secrecy has been well investigated in many programming paradigms: imperative programming [17], functional programming or more precisely lambda-calculi e.g. [2, 15, 10], concurrent programming [1, 11, 12]. Nevertheless the line of approach of these works is rather theoretical: very basic computation models are considered, e.g.  $\lambda$ -calculus,  $\pi$ -calculus and other basic languages. In this paper we consider a programming system closer to reality [6]. It is a combination of declarative programming (term normalization, constraint solving) and concurrency (parallel processes, message passing, etc.). Moreover this framework strictly generalizes those considered in [3] and [16]: e.g., it is possible to dynamically create parallel processes or even to consider terms of the form  $(P \parallel Q); R$ , which are neither possible in [3] nor in [16].

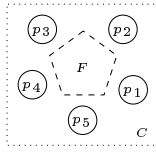


Figure 1. execution model

### 1.3. Paper Overview

In section 2 we briefly present the framework we consider for concurrent declarative programming and its operational semantics. For a description of the complete framework see [6]. We give a precise definition of what we intend by “safety” in section 3. Roughly speaking, we consider a program to be safe if information cannot flow from any privacy level towards a lower privacy level. In other words, we consider safety from a non-interference point of view: a program is safe if there are no interferences from higher levels towards lesser ones. Then we give an abstract operational semantics in section 4. This abstract operational semantics always terminates. It enables to compute a set of constraints. We give a short example to illustrate our analysis in section 5. We show that if the constraints computed by the analysis can be satisfied the program is safe. In section 6 we give the proof of this claim. Concluding remarks are given in section 7.

## 2. A Framework for Concurrent Declarative Programming

The computational model used in this paper is a simplified version of the one proposed in [6, 7]; we refer to these papers for a more detailed presentation of the model and its implementation. Here we limit ourselves to a single component with a fixed set of actions.

Roughly speaking, a program or a component in our framework will consist of two parts  $C = (F, \mathbb{R})$ .  $\mathbb{R}$  is a set of process definitions and  $F$  is a theory presented by a set of formulæ (here, we consider Term Rewriting Systems, TRS for short) which defines a declarative program, called a *store* in the sequel. We assume some familiarity of the reader with TRS (see e.g. [5]).

The execution model of a component can be schematised as in Fig. 1. Processes ( $p_i$ ) communicate by modifying the common store  $F$ , i.e., by altering, in a non-monotonic way, the current theory described by the store, 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 precisely define the different parts of this computational model. The first step is to formalize the notion of store, where constants and functions are defined.

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

Furthermore, the following operations are defined:

- a predicate  $eval(F, t)$ , which holds if the term  $t$  of sort `Truth`, i.e.,  $t \in T_{\text{Truth}}(\Sigma, X)$ , can be reduced to `True` using the rules (or formulæ) of the store  $F = \langle \Sigma, \mathcal{R} \rangle$ , and
- a function  $norm(F, t)$ , which returns the *normal form* of a term  $t \in T(\Sigma, X)$  with respect to the rules of  $F = \langle \Sigma, \mathcal{R} \rangle$ .

In practice a store  $F$  can be seen as a sequence of rules. 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, and conversely we write  $F \setminus (l \rightarrow r \mid c)$  the store  $F$  where rule  $l \rightarrow r \mid c$  has been erased. We write  $F \cup (l \rightarrow r \mid c)$  the store equivalent to store  $F$  with the concatenation of rule  $l \rightarrow r \mid c$ .

Let  $t$  be a term of  $T(\Sigma, X)$ ,  $\rho$  a rewrite rule in  $\mathcal{R}$  and  $\mathbf{p}$  a position in  $t$ , then a rewrite step at position  $\mathbf{p}$  using rule  $\rho$  is written  $t \xrightarrow{\mathbf{p}, \rho} t'$ .

We now define actions which allow to modify stores. We distinguish two kinds of actions: *elementary actions* like the assignment, the addition or removal of store information, and *guarded actions* that are executed only if their guard evaluates to `True`.

**Definition 2 (actions)** An action  $\alpha$  is a pair consisting of a guard  $g$  and a sequence of elementary actions  $a_i$ , written:  $[g \Rightarrow a_1; \dots; a_n]$ . 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 action that does nothing.

The operational semantics of the considered elementary actions is defined later on by the rules named  $(ea_{:=})$ ,  $(ea_{\text{tell}})$ ,  $(ea_{\text{del}})$  and  $(ea_{\text{skip}})$ .

We now define the notion of processes. Basic processes are success (the process which terminates successfully),

guarded actions  $\alpha$ , or process calls  $q(t_1, \dots, t_n)$ . As usual in process algebra (see, e.g., [9]), we provide some operators for combining processes: parallel ( $\parallel$ ) and sequential ( $;$ ) composition, and nondeterministic choice ( $+$ ). More formally we have the following definition.

**Definition 3 (processes)** A process term  $p$  is a well-typed expression defined by the following grammar:

$$p ::= \text{success} \mid [g \Rightarrow a] \mid q(t_1, \dots, t_n) \mid p; p \mid p \parallel p \mid p + p$$

A process  $q$  is defined by a sentence of the form

$$q(x_1, \dots, x_n) \Leftarrow \sum_{i=1}^m \alpha_i ; p_i$$

where (for each  $i$ )  $\alpha_i$  is an action and  $p_i$  is a process term. For a readable presentation, we omit here some formal technical conditions on the use of variables.

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

We now define the operational semantics of this execution model by a transition system. We start by the definition of a transition relation for (possibly empty) sequences of elementary actions. Elementary actions modify the store.

$$\begin{aligned} \langle F, f := t; a \rangle &\hookrightarrow \langle F \bullet (f \rightarrow t), a \rangle & (ea_{:=}) \\ \langle F, \text{tell}(l \rightarrow r \mid c); a \rangle &\hookrightarrow \langle F \cup (l \rightarrow r \mid c), a \rangle & (ea_{\text{tell}}) \\ \langle F, \text{del}(l \rightarrow r \mid c); a \rangle &\hookrightarrow \langle F \setminus (l \rightarrow r \mid c), a \rangle & (ea_{\text{del}}) \\ \langle F, \text{skip}; a \rangle &\hookrightarrow \langle F, a \rangle & (ea_{\text{skip}}) \end{aligned}$$

In the following, if  $\rightarrow$  is a transition relation, we note  $\rightarrow_r$  a transition step using rule  $r$ . For instance, for  $\hookrightarrow$  we have  $\langle F, \text{skip}; a \rangle \hookrightarrow_{(ea_{\text{skip}})} \langle F, a \rangle$ .

Before defining the operational semantics of process terms, we introduce first a notion of equivalence between process terms. This equivalence relation,  $\equiv_p$ , is standard: operators  $\parallel$  and  $+$  are commutative and success may vanish.

$$\begin{aligned} \text{success}; p &\equiv_p p & (Eq_{\text{success}}) \\ \text{success} \parallel p &\equiv_p p & (Eq_{\text{success}\parallel}) \\ p_1 \text{ op } p_2 &\equiv_p p_2 \text{ op } p_1 \quad \text{op} \in \{\parallel, +\} & (Eq_{\text{op com}}) \end{aligned}$$

We are now ready to define the transition relation  $\longrightarrow$  describing process executions. Transitions are of the form  $\langle F, p \rangle \longrightarrow \langle F', p' \rangle$  where  $p$  is any process term. Notice that actions (that is guarded sequences of elementary actions see definition 2) are executed in an atomic way (see rule  $(P_{\text{guard}})$ ).

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

$$\begin{aligned} \frac{(q(x_1, \dots, x_n) \Leftarrow \sum_{j=1}^m \alpha_j ; p_j) \in \mathbb{IR} \quad \langle F, (\sum_{j=1}^m \alpha_j ; p_j)[x_i/t_i] \rangle \longrightarrow \langle F', p' \rangle}{\langle F, q(t_1, \dots, t_n) \rangle \longrightarrow \langle F', p' \rangle} & (P_{\text{abs}}) \\ \frac{\langle F, a_1; \dots; a_n; \text{skip} \rangle \hookrightarrow^* \langle F', \text{skip} \rangle \quad \text{eval}(F, g) = \text{True}}{\langle F, [g \Rightarrow a_1; \dots; a_n] \rangle \longrightarrow \langle F', \text{success} \rangle} & (P_{\text{guard}}) \end{aligned}$$

$$\frac{\langle F, p_1 \rangle \longrightarrow \langle F', p'_1 \rangle}{\langle F, p_1; p_2 \rangle \longrightarrow \langle F', p'_1; p_2 \rangle} \quad (P_{;})$$

$$\frac{\langle F, p_1 \rangle \longrightarrow \langle F', p'_1 \rangle}{\langle F, p_1 \parallel p_2 \rangle \longrightarrow \langle F', p'_1 \parallel p_2 \rangle} \quad (P_{\parallel})$$

$$\frac{\langle F, p_1 \rangle \longrightarrow \langle F', p'_1 \rangle}{\langle F, p_1 + p_2 \rangle \longrightarrow \langle F', p'_1 \rangle} \quad (P_{+})$$

### 3. Formalization of Safety

In this section we precisely define what we intend by safety in our setting. We assign to each symbol in store  $F$  a privacy level indicating its status. The higher the privacy level of a value, the more private is the status of this value.

We start with the definition of *privacy levels*. The idea is to assign to every symbol of the signature, i.e., all elements of  $\Omega$ , an element of a lattice  $\mathcal{L}$ . We note  $\sqsubseteq$  the order defined on  $\mathcal{L}$  and we make no difference between the lattice and the set of its elements, i.e.,  $\mathcal{L}$  denotes in the same time the lattice and its carrier. If  $\pi_1, \pi_2$  are two elements of  $\mathcal{L}$  and  $\pi_1 \sqsubseteq \pi_2$ , we say that  $\pi_2$  is more *private* than  $\pi_1$ . We write  $\sqcup$  for the join operation (least upper bound) and  $\sqcap$  for the meet operation (greatest lower bound). The upper bound of  $\mathcal{L}$  is  $\top$  and its lower bound  $\perp$ .

**Definition 5 (privacy map)** Let  $F$  be a store. We call any map  $\ell$  from symbols of  $\Omega$  towards  $\mathcal{L}$  a *privacy map*.

We extend naturally any privacy map  $\ell$  to all terms in  $T(\Sigma, X)$ . The *privacy level* of a term is recursively defined as the least upper bound of privacy level of its subterms:

$$\begin{aligned} \ell(f(t_1, \dots, t_n)) &= (\bigsqcup_{i=1}^n \ell(t_i)) \sqcup \ell(f) \\ \ell(x) &= \perp \end{aligned}$$

where  $x \in X$  is a variable.

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

$$\begin{aligned} \ell([g \Rightarrow a_1; \dots; a_n]) &= \prod_{i=1}^n \ell(a_i) \\ \ell(\text{tell}(l \rightarrow r \mid c)) &= \ell(\text{del}(l \rightarrow r \mid c)) = \ell(l) \\ \ell(\text{skip}) &= \top \end{aligned}$$

We also define a notion of safe rewrite rules. Informally a rewrite rule is safe whenever no information may **fbw** from a higher level towards a lower one. Suppose that  $PIN$  is a constant denoting an information of the highest level:  $\ell(PIN) = \top$ , whereas  $SPY$  is a constant of the lowest level,  $\ell(SPY) = \perp$  then rewrite rules such as  $SPY \rightarrow PIN$ , or  $SPY \rightarrow v \mid PIN = v'$  are not safe since a high level information is transmitted to a lower level one.

**Definition 6 (safe rewrite rules)** A rewrite rule  $l \rightarrow r \mid c$  is safe whenever the following condition holds:

$$(\ell(r) \sqsubseteq \ell(l)) \wedge (\ell(c) \sqsubseteq \ell(l))$$

In other words, rewrite rules are safe when the result of a rewrite step gives an information of a lesser or equal privacy than what it depends upon. It is in line with our informal presentation of safety, where we stated that information may only **fbw** from low levels towards high levels.

We naturally extend this notion of safe rewrite rules to safe stores. A store  $F = \langle \Sigma, \mathcal{R} \rangle$  is safe, if all rewrite rules  $l \rightarrow r \mid c \in \mathcal{R}$  are safe.

In order to define what we intend by safe processes, we have first to define a notion of equivalence between stores up to a given privacy level.

**Definition 7 (store equivalence)** Let  $F_1, F_2$  be two stores, and  $\pi \in \mathcal{L}$ . We say that  $F_1$  and  $F_2$  are  $\pi$ -equivalent up to the privacy map  $\ell$  and write  $F_1 \cong_{\pi}^{\ell} F_2$  iff:

1.  $\forall \rho_1 = l_1 \rightarrow r_1 \mid c_1 \in \mathcal{R}_1$ , and  $\ell(\rho_1) \sqsubseteq \pi$  then up to variable renaming,  $\exists \rho_2 = l_2 \rightarrow r_2 \mid c_2 \in \mathcal{R}_2$ , and  $\rho_1 = \rho_2$ .
2.  $\forall \rho_2 = l_2 \rightarrow r_2 \mid c_2 \in \mathcal{R}_2$ , and  $\ell(\rho_2) \sqsubseteq \pi$  then up to variable renaming,  $\exists \rho_1 = l_1 \rightarrow r_1 \mid c_1 \in \mathcal{R}_1$ , and  $\rho_2 = \rho_1$ .

In the following we write  $\pi\ell$ -equivalent stores instead of  $\pi$ -equivalent up to the privacy map  $\ell$ .

Informally, two stores are  $\pi\ell$ -equivalent if they agree on all information the privacy level of which is less than  $\pi$ . An important property of safe  $\pi\ell$ -equivalent stores, is that the evaluation of a term is independent from the rules of a higher privacy than  $\pi$ .

**Lemma 1** Let  $\pi$  be a privacy level,  $\ell$  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  $F_1 \cong_{\pi}^{\ell} F_2$  and  $t$  be a term of  $T(\Sigma, \emptyset)$  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  $l \rightarrow r \mid c$  is  $\ell(l \rightarrow r \mid c) = \ell(l)$ . Now since the privacy level of a term is the least upper bound of the privacy levels of its subterms, if rule  $\rho_1$  can be used at position  $\mathbf{p}$ , then the subterm at this position has a privacy level lower than  $\pi$ , and the level of  $\rho_1$  is also lower than  $\pi$  (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  $\pi$ . By definition of  $\cong_{\pi}^{\ell}$  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).  $\square$

We now define a notion of bisimulation of processes in our framework. Bisimulation is defined up to a privacy level  $\pi$ . Informally two processes  $p_1, p_2$  are bisimilar up to a privacy level  $\pi$  and a privacy map  $\ell$  ( $\pi\ell$ -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 case, we ensure that every execution of  $p_2$  only affects parts of the store with a higher privacy than  $\pi$ . Hence this bisimulation formalizes the idea that information may **fbw** from low privacy levels to high privacy levels but not vice-versa. More formally:

**Definition 8 ( $\pi\ell$ -bisimulation)** Let  $p_1, p_2$  be two  $\mathbb{IR}$ -process terms,  $\pi \in \mathcal{L}$ , and  $F_1, F_2$  two stores, and  $\ell$  a privacy map such that  $F_1 \cong_{\pi}^{\ell} F_2$ . A relation  $\mathcal{B}_{\pi}^{\ell}$  is a  $\pi\ell$ -bisimulation if it is symmetric and if  $\langle F_1, p_1 \rangle \mathcal{B}_{\pi}^{\ell} \langle F_2, p_2 \rangle$  implies:

- Either  $p_1$  reduces, and for all  $p'_1$  such that  $\langle F_1, p_1 \rangle \longrightarrow \langle F'_1, p'_1 \rangle$ , then there exists  $p'_2$  such that  $\langle F_2, p_2 \rangle \longrightarrow \langle F'_2, p'_2 \rangle$ , and  $F'_1 \cong_{\pi}^{\ell} F'_2$  and  $\langle F'_1, p'_1 \rangle \mathcal{B}_{\pi}^{\ell} \langle F'_2, p'_2 \rangle$ .
- Or  $p_1$  does not reduce. Then we have for all  $p'_2$  such that  $\langle F_2, p_2 \rangle \longrightarrow \langle F'_2, p'_2 \rangle$  that  $F_1 \cong_{\pi}^{\ell} F'_2$  and  $\langle F_1, p_1 \rangle \mathcal{B}_{\pi}^{\ell} \langle F'_2, p'_2 \rangle$ .

In the following we write  $\approx_{\pi}^{\ell}$  the largest  $\pi\ell$ -bisimulation and we say for short that two process terms are bisimilar up to  $\pi\ell$  (or simply bisimilar if  $\pi$  and  $\ell$  can be inferred from the context) if they are  $\approx_{\pi}^{\ell}$ -bisimilar.

We now state a lemma expressing the fact that if a process is not  $\pi\ell$ -bisimilar to itself starting with  $\pi\ell$ -equivalent stores, it means that there is a point (reachable in a finite number of transition steps) where either the resulting stores are no longer  $\pi\ell$ -equivalent or that an action can be done under one store but not under the other one. Intuitively it means that a process is not  $\pi\ell$ -bisimilar to itself if it modifies the store on a lower privacy level than  $\pi$  in a different

way starting from two  $\pi$ -equivalent stores. Thus it means that an action depending on values of a higher privacy than  $\pi$  has been executed (since because of lemma 1 we know that values with a lower privacy than  $\pi$  must be the same on two  $\pi$ -equivalent stores) and modifies the store under the privacy  $\pi$ . It is exactly what happens for  $SPY := PIN$  (with  $SPY$  a public variable and  $PIN$  a private one). This case describes a top down information flow: values with a low privacy level are computed from values of higher privacy. The second case of the lemma reflects a more subtle behavior: it occurs when an action can be executed using one store and cannot be executed using another  $\pi$ -equivalent store. This situation occurs when the guard of an action is of a higher level than  $\pi$  (otherwise one would reach the same evaluation of the guard see lemma 1) and evaluates differently on two  $\pi$ -equivalent stores. There is no incidence as long as subsequent store modifications concern higher levels than  $\pi$ . Indeed, in this case both stores remain  $\pi$ -equivalent. However if a modification is executed below the level  $\pi$  then the stores are no longer  $\pi$ -equivalent.

**Lemma 2** *Let  $\ell$  be a privacy map,  $\pi$  a privacy level,  $p_0$  a process term and  $F_0^1, F_0^2$  two stores.  $\langle F_0^1, p_0 \rangle \not\approx_\pi^\ell \langle F_0^2, p_0 \rangle$  implies that there exist an integer  $N$ , and two derivations:*

$$\begin{aligned} \langle F_0^0, p_0 \rangle &\longrightarrow \langle F_1^0, p_1 \rangle \longrightarrow \dots \longrightarrow \langle F_N^0, p_N \rangle \\ \langle F_0^1, p_0 \rangle &\longrightarrow \langle F_1^1, p_1 \rangle \longrightarrow \dots \longrightarrow \langle F_N^1, p_N \rangle \end{aligned}$$

such that for all  $j < N$ ,  $F_j^0 \cong_\pi^\ell F_j^1$  and

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

$$\langle F_N^j, p_N \rangle \xrightarrow{\alpha} \langle F_{N+1}^j, p_{N+1} \rangle \longrightarrow^* \langle F^\#, p^\# \rangle$$

but  $\langle F_N^{1-j}, p_N \rangle \not\rightarrow^\alpha$  and  $F_N^{1-j} \not\approx_\pi^\ell F^\#$ .

**Proof:** Intuitively, lemma 2 is simply the negation of definition 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

$$\langle F_0^0, p_0 \rangle \xrightarrow{\alpha_1} \langle F_1^0, p_1 \rangle \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} \langle F_n^0, p_n \rangle \not\rightarrow^\alpha$$

We prove the lemma by induction on the length  $n$  of the transition sequence  $d_0$ .

*Base Case.* In the case that  $n = 0$ , if we have  $F_0^0 \not\approx_\pi^\ell F_0^1$  the lemma is obviously true (for  $N = 0$ ).

Otherwise, by definition of  $\approx_\pi^\ell$ , there exist  $F^\#, p^\#$  such that  $\langle F_0^1, p_0 \rangle \longrightarrow^+ \langle F^\#, p^\# \rangle$  with  $F_0^0 \not\approx_\pi^\ell F^\#$ . 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  $F_0^0 \not\approx_\pi^\ell F_0^1$ , the lemma is obviously true (for  $N = 0$ ). Otherwise we distinguish two further cases:

- $\langle F_0^1, p_0 \rangle \not\rightarrow^\alpha$ . Assume that  $F_0^1 \cong_\pi^\ell F_i^0$  for all  $i \in \{1, \dots, n\}$ . 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  $\langle F_0^0, p_0 \rangle \xrightarrow{\alpha_1} \langle F_1^0, p_1 \rangle \longrightarrow^* \langle F_{i_0}^0, p_{i_0}^0 \rangle$  but  $\langle F_0^1, p_0 \rangle \not\rightarrow^\alpha$  and  $F_{i_0}^0 \not\approx_\pi^\ell F_0^1$ , i.e., we are in the second situation of lemma 2 for  $N = 0$ .
- There exists  $p'_0$  such that  $\langle F_0^1, p_0 \rangle \xrightarrow{\alpha_1} \langle F_1^1, p'_0 \rangle$ . In this case, we have also the following transition  $\langle F_0^1, p_0 \rangle \xrightarrow{\alpha_1} \langle F_1^1, p_1 \rangle$ . Hence we have  $\langle F_1^0, p_1 \rangle \not\approx_\pi^\ell \langle F_1^1, p_1 \rangle$  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

$$\langle F_0^0, p_0 \rangle \xrightarrow{\alpha_1} \langle F_1^0, p_1 \rangle \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_i} \langle F_i^0, p_i \rangle \xrightarrow{\alpha_{i+1}} \dots$$

Consider the maximal transition sequence  $d_1$  for  $F_0^1$  and  $p_0$  which corresponds to an execution of (a prefix of) the action sequence  $(\alpha_i)_{i>0}$ . If  $d_1$  is finite, the proof is symmetric to case 1. Thus we suppose we have an infinite transition sequence<sup>3</sup>

$$\langle F_0^1, p_0 \rangle \xrightarrow{\alpha_1} \langle F_1^1, p_1 \rangle \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_i} \langle F_i^1, p_i \rangle \xrightarrow{\alpha_{i+1}} \dots$$

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  $F_n^0 \not\approx_\pi^\ell F_n^1$ .  $\square$

Now we can define the notion of safe process. A process is safe if it is bisimilar to itself for every privacy level. In other words it means that a process is safe if all actions performed at a level  $\pi$  only depend on information having a privacy lower than  $\pi$ .

<sup>1</sup>The transition yields the store  $F_1^1$  since the execution of actions is deterministic.

<sup>2</sup>Since the store and the process term are the same, we just have to use the same inference rules to infer this transition.

<sup>3</sup>We have the same process terms in the transition sequences  $d_0$  and  $d_1$  by a similar reasoning as above (see footnote 2).

**Definition 9 (safe program)** We call a program  $p$  of system  $\langle F, \mathbb{IR} \rangle$  safe relatively to a privacy map  $\ell$ , iff for all  $\pi$ , for all  $F_1, F_2$  such that  $F_1 \cong_{\pi}^{\ell} F_2 \cong_{\pi}^{\ell} F$  then  $\langle F_1, p \rangle \approx_{\pi}^{\ell} \langle F_2, p \rangle$ .

## 4. Abstraction and Constraint Generation

We develop an abstract operational semantics aiming at the generation of a constraint set. Constraints are inequations on privacy levels. This constraint set represents the constraints to be satisfied in order to have a safe program (safe in the sense of definition 9). If the constraint set cannot be satisfied, then it means that the analyzed program may not be safe.

We design an abstract operational semantics that always terminates. It plays the same role as the type systems of [16, 2, 3]. In these papers, type inference is used to analyze non-interference properties of programs, either for imperative, functional or concurrent programs. Types inferred in such analyses are not similar to “standard types”, say  $\text{Int} \rightarrow \text{Int}$ , they are rather like  $\text{Int}^{\pi_1} \rightarrow \text{Int}^{\pi_2}$ . 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 [15]. We follow this strategy. We use the skeleton of an execution in order to collect information. In this abstract operational semantics, we only record the names of symbols used while the store is reduced to a set of inequations.

We first start by the definition of an abstract operational semantics, and then show how one can analyze safety properties of a program using this abstract operational semantics.

### 4.1. Abstract Operational Semantics

We consider given a system  $\mathcal{S} = \langle F, \mathbb{IR} \rangle$ . We define, relatively to this system, an abstract store  $F^A$ , and a new operational semantics operating on this abstract store. Informally, an abstract 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 semantics collects inequations through abstract executions. The abstract semantics is defined in such a way that the number of possible abstract executions is finite. The idea is to produce the whole set of inequations relatively to a program. If a privacy map  $\ell$  defined on  $\mathcal{S}$  is such that the whole set of inequations holds, then we claim that the analyzed program is safe relatively to  $\ell$ . This is a kind of abstract interpretation (see e.g. [4, 14, 13]).

Abstract stores are sets of privacy inequations. We start by giving a precise definition of privacy inequations and then define abstract stores.

**Definition 10 (privacy inequations and formulæ)** We define privacy formulæ  $f$  by the following grammar:

$$f ::= \pi \mid x \mid f \sqcap f \mid f \sqcup f$$

where  $x$  denotes constants. Privacy inequations are statements of the form

$$f_1 \sqsubseteq f_2$$

**Definition 11 (abstract stores)** An abstract store  $F^A$  is a couple of an abstract signature and an abstract set of rules  $\langle \Sigma^A, \mathcal{R}^A \rangle$ , such that:

- $\Sigma^A = \langle S^A, \Omega^A \rangle$  is a one sorted set of constants.
- $\mathcal{R}^A$  is a set of privacy inequations build with symbols belonging to  $\Sigma^A$ .

Let  $F = \langle \Sigma, \mathcal{R} \rangle$  be a store, we define the abstract store,  $F^A = \langle \Sigma^A, \mathcal{R}^A \rangle$  of  $F$  by:

- For all elements  $f \in \Sigma$ , there is a symbol  $x_f$  in  $\Sigma^A$ .
- For all rules  $l \rightarrow r \mid c$  in  $\mathcal{R}$ , there are inequations  $\ell^A(r) \sqsubseteq \ell^A(l), \ell^A(c) \sqsubseteq \ell^A(l)$  in  $\mathcal{R}_t^A$ , where  $\ell^A$  is the function inductively defined by:

$$\begin{aligned} \ell^A(f) &= x_f, \forall f \in \Sigma \\ \ell^A(f(t_1, \dots, t_n)) &= \ell^A(f) \sqcup \ell^A(t_1) \sqcup \dots \sqcup \ell^A(t_n) \end{aligned}$$

We call  $F$  the original store of  $F^A$ .

In the following we write  $t^A$  instead of  $\ell^A(t)$  for short.

The abstract store is intended to collect constraints to be verified in order to have a safe program. These constraints are generated by an abstract operational semantics. The abstract operational semantics is defined by a transition system. Roughly, the states of this transition system are triples  $\langle F^A, p^A, \sigma \rangle$  consisting of an abstract store  $F^A$ , an abstract process term  $p^A$  and a privacy level  $\sigma$  of  $\mathcal{L}$ . Transitions (abstract executions) generate constraints and record them into the abstract store. These constraints depend on  $\sigma$  as well as on the privacy level of terms manipulated by the program.

We start by the abstract execution of sequences of elementary actions. Abstract elementary actions modify an abstract store  $F^A$  relatively to a privacy level  $\sigma$  of  $\mathcal{L}$ . The abstract execution is defined as follows:

$$\langle F^A, f := t; a, \sigma \rangle \xrightarrow{A} \langle F^A \cup \{t^A \sqsubseteq f^A\} \cup \{\sigma \sqsubseteq f^A\}, a, \sigma \rangle \quad (\text{Aea}_{:=})$$

$$\langle F^A, \text{tell}(l \rightarrow r \mid c); a, \sigma \rangle \xrightarrow{A} \langle F^A \cup \{r^A \sqsubseteq l^A\} \cup \{c^A \sqsubseteq l^A\} \cup \{\sigma \sqsubseteq l^A\}, a, \sigma \rangle \quad (\text{Aea}_{\text{tell}})$$

$$\langle F^A, \text{del}(l \rightarrow r \mid c); a, \sigma \rangle \xrightarrow{A} \langle F^A \cup \{\sigma \sqsubseteq l^A\}, a, \sigma \rangle \quad (\text{Aea}_{\text{del}})$$

$$\langle F^A, \text{skip}; a, \sigma \rangle \xrightarrow{A} \langle F^A, a, \sigma \rangle \quad (\text{Aea}_{\text{skip}})$$

The abstract transition relation on process terms has the following form  $\langle F^A, \mathcal{M} \rangle \xrightarrow{A} \langle F^{A'}, \mathcal{M}' \rangle$ , where  $\mathcal{M}$  terms are defined by the following grammar:

$$\mathcal{M} ::= \langle p, \sigma \rangle \mid \mathcal{M} \parallel^A \mathcal{M} \mid \mathcal{M} +^A \mathcal{M} \mid \mathcal{M} ;^A \mathcal{M}$$

The introduction of abstract operators ( $+^A$ ,  $\parallel^A$ ,  $;\ ^A$ ) is needed because we consider for the abstract transition, couples formed by a process ( $p$ ) and a privacy level ( $\sigma$ ) indicating the highest level of a guard used up to this point. The most important abstract operator is  $\parallel^A$ , the others are defined for notational coherence but don't influence the analysis.  $\parallel^A$  is necessary because it is possible that in processes like  $p \parallel q$ , the process  $p$  uses guards of a high level while process  $q$  only works on low privacy levels. If we don't duplicate  $\sigma$ , then such processes would be analyzed as unsafe since constraints generated by  $p$  can be too strong relatively to  $q$ . In some sense, the role of  $\parallel^A$  is similar to the role of the subtyping rule in type based analyses (see for instance [3]). Indeed in a type based analysis it is possible to build different constraints (that is to associate different types) for both parts of a parallel process and then to use the subtyping rule in order to have the same type in both branches in the type derivation tree.

We define a function  $\phi$  from  $\mathcal{M}$  terms to process terms, it gives the original term of an  $\mathcal{M}$  term.

$$\begin{aligned} \phi(\langle p, \sigma \rangle) &= p \\ \phi(\mathcal{M} \parallel^A \mathcal{M}') &= \phi(\mathcal{M}) \parallel \phi(\mathcal{M}') \\ \phi(\mathcal{M} +^A \mathcal{M}') &= \phi(\mathcal{M}) + \phi(\mathcal{M}') \\ \phi(\mathcal{M} ;^A \mathcal{M}') &= \phi(\mathcal{M}) ; \phi(\mathcal{M}') \end{aligned}$$

We define  $\equiv^A$ , a structural congruence on  $\mathcal{M}$ -terms, by:

$$\begin{aligned} \frac{p_1 \equiv_p p_2}{\langle p_1, \sigma \rangle \equiv^A \langle p_2, \sigma \rangle} & \quad (\text{AE}q_{\equiv_p}) \\ \mathcal{M}_1 \parallel^A \mathcal{M}_2 \equiv^A \mathcal{M}_2 \parallel^A \mathcal{M}_1 & \quad (\text{AE}q_{\parallel^A}) \\ \mathcal{M}_1 +^A \mathcal{M}_2 \equiv^A \mathcal{M}_2 +^A \mathcal{M}_1 & \quad (\text{AE}q_{+^A}) \end{aligned}$$

This congruence is used to manage abstract operators, and is going to be used to define the abstract operational semantics of process terms.

The abstract operational semantics of process terms is defined on couples of the form  $\langle F^A, \mathcal{M} \rangle$ . When  $\mathcal{M}$  is of the form  $\langle p_1 \parallel p_2, \sigma \rangle$ , we need to “translate” the  $\parallel$  operator into a  $\parallel^A$  one. Indeed, it is by the use of  $\parallel^A$  that we can collect constraints generated by the execution of  $p_1$  and  $p_2$ . Conversely when both processes of the parallel terminate with success we need to join the results of both parts. To this aim we define a transformation relation  $\mapsto$ , which introduces and eliminates  $\parallel^A$ . The introduction occurs when the process term of a  $\mathcal{M}$ -term is a parallel, and the elimination occurs when both processes are success.

$$\langle p_1 \parallel p_2, \sigma \rangle \mapsto \langle p_1, \sigma \rangle \parallel^A \langle p_2, \sigma \rangle \quad (\text{A} \parallel^A - I)$$

$$\langle p_1 + p_2, \sigma \rangle \mapsto \langle p_1, \sigma \rangle +^A \langle p_2, \sigma \rangle \quad (\text{A} +^A - I)$$

$$\langle p_1 ; p_2, \sigma \rangle \mapsto \langle p_1, \sigma \rangle ;^A \langle p_2, \sigma \rangle \quad (\text{A} ;^A - I)$$

$$\langle \text{success}, \sigma_1 \rangle \parallel^A \langle \text{success}, \sigma_2 \rangle \mapsto \langle \text{success}, \sigma_1 \sqcup \sigma_2 \rangle \quad (\text{A} \parallel^A - E)$$

$$\langle \text{success}, \sigma_1 \rangle ;^A \langle p_2, \sigma_2 \rangle \mapsto \langle p_2, \sigma_1 \sqcup \sigma_2 \rangle \quad (\text{A} ;^A - E)$$

It is clear that  $\mapsto$  is confluent and strongly normalizing.

We write  $\overline{\mathcal{M}}^{\text{nf}}$  the  $\mapsto$  normal form of  $\mathcal{M}$ . The idea is to transform all  $\parallel$  into  $\parallel^A$  to be able to compute constraints generated by both processes of the  $\parallel$  separately (rule  $(\text{A} \parallel^A - I)$ ) and conversely to merge results when both processes running in parallel have terminated successfully (rule  $(\text{A} \parallel^A - E)$ ).

The abstract operational semantics is defined by the following inference rules:

$$\frac{\overline{\mathcal{M}}_1^{\text{nf}} \equiv^A \mathcal{M}_2 \quad \overline{\mathcal{M}}_3^{\text{nf}} \equiv^A \mathcal{M}_4 \quad \langle F^A, \mathcal{M}_2 \rangle \xrightarrow{A} \langle F^{A'}, \mathcal{M}_3 \rangle}{\langle F^A, \mathcal{M}_1 \rangle \xrightarrow{A} \langle F^{A'}, \mathcal{M}_4 \rangle} \quad (\text{AP}_{\equiv^A})$$

$$\frac{\text{q}(x_1, \dots, x_n) \Leftarrow \Sigma_{j=1}^m \alpha_j ; p_j \in \mathbb{R} \quad \langle F^A, \langle (\Sigma_{j=1}^m \alpha_j ; p_j)[x_i/t_i], \sigma \rangle \rangle \xrightarrow{A} \langle F^{A'}, \mathcal{M}' \rangle}{\langle F^A, \langle \text{q}(t_1, \dots, t_n), \sigma \rangle \rangle \xrightarrow{A} \langle F^{A'}, \mathcal{M}' \rangle} \quad (\text{AP}_{abs})$$

$$\frac{\langle F^A, \langle a_1 ; \dots ; a_n ; \text{skip}, \sigma \sqcup g^A \rangle \rangle \xrightarrow{A^*} \langle F^{A'}, \langle \text{skip}, \sigma \sqcup g^A \rangle \rangle}{\langle F^A, \langle [g \Rightarrow a_1 ; \dots ; a_n], \sigma \rangle \rangle \xrightarrow{A} \langle F^{A'}, \langle \text{success}, \sigma \sqcup g^A \rangle \rangle} \quad (\text{AP}_{guard})$$

$$\frac{\langle F^A, \langle p_1, \sigma \rangle \rangle \xrightarrow{A} \langle F^{A'}, \langle p'_1, \sigma' \rangle \rangle}{\langle F^A, \langle p_1 ; \dots ; p_n, \sigma' \rangle \rangle \xrightarrow{A} \langle F^{A'}, \langle p'_1 ; \dots ; p_n, \sigma' \rangle \rangle} \quad (\text{AP}_{;})$$

$$\frac{\langle F^A, \mathcal{M}_1 \rangle \xrightarrow{A} \langle F^{A'}, \mathcal{M}'_1 \rangle}{\langle F^A, \mathcal{M}_1 \parallel^A \mathcal{M}_2 \rangle \xrightarrow{A} \langle F^{A'}, \mathcal{M}'_1 \parallel^A \mathcal{M}_2 \rangle} \quad (\text{AP}_{\parallel^A})$$

$$\frac{\langle F^A, \langle p_1, \sigma \rangle \rangle \xrightarrow{A} \langle F^{A'}, \mathcal{M} \rangle}{\langle F^A, \langle p_1 + p_2, \sigma \rangle \rangle \xrightarrow{A} \langle F^{A'}, \mathcal{M} \rangle} \quad (\text{AP}_{+})$$

Inspection of the different inference rules allows to prove (in the following two lemmas) that to each concrete transition step corresponds an abstract reduction step.

**Lemma 3** *We have that  $\langle F, a ; a \rangle \hookrightarrow \langle F', a \rangle$  implies that for all  $\sigma$  and constraint sets  $C$  there exists  $C' \supseteq C$  such that*

$$\langle F^A \cup C, \langle a ; a, \sigma \rangle \rangle \xrightarrow{A} \langle F'^A \cup C', \langle a, \sigma \rangle \rangle$$

**Proof:** We analyze our four elementary actions separately.

skip: According to  $(ea_{\text{skip}})$ ,  $F' = F$ . We choose  $C' = C$  and lemma 3 holds due to  $(Aea_{\text{skip}})$ .

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

del( $l \rightarrow r \mid c$ ): We distinguish two situations:

On the one hand, if  $l \rightarrow r \mid c \notin F$ , then we have by  $(ea_{\text{del}})$  that  $F' = F$ , and inspection of  $(Aea_{\text{del}})$  shows lemma 3 holds for  $C' = C \cup \{\sigma \sqsubseteq l^A\}$ .

On the other hand, if  $l \rightarrow r \mid c \in F$ , then we have that  $F'^A = F^A \cup \{r^A \sqsubseteq l^A; c^A \sqsubseteq l^A\}$ . Hence lemma 3 holds by choosing

$$C' = C \cup \{r^A \sqsubseteq l^A; c^A \sqsubseteq l^A\} \cup \{\sigma \sqsubseteq l^A\}$$

$f := t$ : By a similar reasoning as for tell and del<sup>4</sup>, inspection of  $(Aea_{:=})$  and  $(ea_{:=})$  shows that lemma 3 holds for  $C' = C \cup D \cup \{\sigma \sqsubseteq f^A\}$ , where  $D$  is defined as the following set of privacy inequations

$$D = \{r \sqsubseteq l; c \sqsubseteq l \text{ such that } \exists(l \rightarrow r \mid c) \in F \setminus F'\} \quad \square$$

**Lemma 4** Let  $\mathcal{S} = \langle F, \mathbb{R} \rangle$ , 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  $\langle F^A \cup C, \overline{\mathcal{M}}^{\text{nf}} \rangle \xrightarrow{A} \langle F'^A \cup C', \mathcal{M}' \rangle$ ,  $\phi(\mathcal{M}') \equiv_p p'$  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  $(AP_{\text{guard}})$ . Notice first that  $\phi(\mathcal{M}) = [g \Rightarrow a_1; \dots; a_n]$  implies that  $\mathcal{M} = \overline{\mathcal{M}}^{\text{nf}}$  and  $\mathcal{M}$  is of the form  $\langle [g \Rightarrow a_1; \dots; a_n], \sigma \rangle$ , where  $\sigma$  is a privacy level. Applying lemma 3  $n$  times, we have that  $\langle F, a_1; \dots; a_n; \text{skip} \rangle \hookrightarrow \langle F', \text{skip} \rangle$  implies that for all  $\sigma$  and constraint sets  $C$  there exists  $C' \supseteq C$  such that

$$\langle F^A \cup C, \langle a_1; \dots; a_n; \text{skip}, \sigma \rangle \rangle \xrightarrow{A} \langle F'^A \cup C', \langle \text{skip}, \sigma \rangle \rangle$$

Defining  $\mathcal{M}' = \langle \text{success}, \sigma \rangle$ , we have by rule  $(AP_{\text{guard}})$  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.

<sup>4</sup>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  $\langle F^A, \overline{\mathcal{M}_2}^{\text{nf}} \rangle \xrightarrow{A} \langle F'^A, \mathcal{M}_3 \rangle$ ,  $\phi(\mathcal{M}_3) \equiv_p p_3$  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  $\overline{\mathcal{M}_2}^{\text{nf}} \equiv^A \overline{\mathcal{M}_1}^{\text{nf}}$  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  $\langle F^A, \overline{\mathcal{M}_2}^{\text{nf}} \rangle \xrightarrow{A} \langle F'^A, \mathcal{M}_3 \rangle$ ,  $\phi(\mathcal{M}_3) = p_3$  and  $C \subseteq C'$ . Defining  $\mathcal{M}_4 = \overline{\mathcal{M}_3}^{\text{nf}}$ , we have obviously  $\phi(\mathcal{M}_4) \equiv_p p_4$ . Thus by application of rule  $(AP_{\equiv^A})$ , lemma 4 holds.

$(P_{\parallel})$ : The premise of  $(P_{\parallel})$  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  $\langle F^A, \overline{\mathcal{M}_1}^{\text{nf}} \rangle \xrightarrow{A} \langle F'^A, \mathcal{M}'_1 \rangle$ ,  $\phi(\mathcal{M}'_1) = p'_1$  and  $C \subseteq C'$ . Hence we have by rules  $(AP_{\parallel^A})$  and  $(AP_{\equiv^A})$  an abstract transition  $\langle F^A, \mathcal{M}_1 \parallel^A \mathcal{M}_2 \rangle \xrightarrow{A} \langle F'^A, \mathcal{M}'_1 \parallel^A \mathcal{M}_2 \rangle$  for all  $\mathcal{M}_2$ , and in particular, for all  $\mathcal{M}_2$  such that  $\phi(\mathcal{M}_1 \parallel^A \mathcal{M}_2) = p_1 \parallel p_2$ , which proves lemma 4.

$(P_{\text{abs}}), (P_{;})$ , or  $(P_{+})$ : Since these cases are proven in a similar way as the case for rule  $(P_{\parallel})$ , we omit them here.  $\square$

## 4.2. Program Analysis

The idea of our program analysis is to perform all possible abstract executions of a program and to collect all computed privacy inequations. We claim that if there exists a substitution of privacy formulæ variables to  $\mathcal{L}$  elements, then the program analyzed is safe relatively to a privacy map generated by this substitution, that is a privacy map assigning the same privacy to  $f$  than the substitution associates to  $x_f$  (see definition 11).

The point is that abstract executions of programs are not infinite. More precisely, up to a certain point no more privacy inequations are created. It comes from the fact that elementary actions do not modify values of the store (see rules  $(Aea_{:=}), (Aea_{\text{tell}}), (Aea_{\text{del}})$ ) but increase the number of privacy inequities. Nevertheless, abstract executions are purely symbolic. Since the number of symbols appearing in a program is finite so is the number of inequalities that can be generated by this program. Therefore it is possible to consider the whole execution tree of a program (infinite branches being cut where no more information left can be collected). The union of all abstract set of inequalities of the leafs of this program abstract execution is the result of the analysis.

**Definition 12 (analysis reduction)** We define the analysis reduction  $\rightsquigarrow$  as the following relation between triples of the form  $\langle F^A, \mathcal{M}, \mathfrak{H} \rangle$ , where  $\mathfrak{H}$  (denoting the  $\mathfrak{H}$ istory of executed process calls) is a set of couples of the form  $\langle q, [\ell^A(t_1); \dots; \ell^A(t_n)] \rangle$ . It is defined as follows:

- If  $\langle F^A, \mathcal{M} \rangle \xrightarrow{A} \langle F^{A'}, \mathcal{M}' \rangle$  using a reduction rule different from  $(AP_{abs})$ , then

$$\langle F^A, \mathcal{M}, \mathfrak{H} \rangle \rightsquigarrow \langle F^{A'}, \mathcal{M}, \mathfrak{H} \rangle$$

- If  $\mathcal{M} = \langle q(t_1, \dots, t_n), \sigma \rangle$ , where the process  $q$  is defined by  $\langle q(x_1, \dots, x_n) \Leftarrow \sum_{j=1}^m \alpha_j; p_j \rangle \in \mathbb{IR}$  and  $\langle F^A, \langle (\sum_{j=1}^m \alpha_j; p_j)[x_i/t_i], \sigma \rangle \rangle \xrightarrow{A} \langle F^{A'}, \mathcal{M}' \rangle$ , then

$$\langle F^A, \mathcal{M}, \mathfrak{H} \rangle \rightsquigarrow \begin{cases} \langle F^A, \langle \text{success}, \sigma \rangle, \mathfrak{H} \rangle & \text{if } \langle q, [t_1^A; \dots; t_n^A] \rangle \in \mathfrak{H} \\ \langle F^{A'}, \mathcal{M}', \mathfrak{H} \cup \langle q, [t_1^A; \dots; t_n^A] \rangle \rangle & \text{otherwise} \end{cases}$$

The abstract operational semantics is purely symbolic. It only uses abstract terms ( $t^A$ ) instead of terms. Abstract terms are privacy formulæ (see definitions 10 and 11). Therefore the number of non equivalent abstract process calls is finite. Indeed  $\sqcup$  is idempotent and associative, thus, for instance,  $\ell^A(f(g(f(x)))) = \ell^A(f(g(x)))$ . Thus there are no infinite  $\rightsquigarrow$  reduction sequences.

**Theorem 1** Relation  $\rightsquigarrow$  is strongly normalizing.

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

Now suppose that there is an infinite number of reduction steps using rule  $(AP_{abs})$ . Since the size of the store is finite, so is the number of terms like  $\ell^A(t)$ . Indeed, the definition of  $\ell^A$  (see def 11) uses the idempotent operator  $\sqcup$ . Finally the number of process definition is also finite, therefore the number of couples of the form  $\langle q, [t_1^A; \dots; t_n^A] \rangle$  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  $\langle q, [t_1^A; \dots; t_n^A] \rangle$  has already been integrated in  $\mathfrak{H}$ , and by definition of  $\rightsquigarrow$  it yields the process success, hence no longer  $\xrightarrow{A}$  reduction step can be executed. Thus there can be no infinite number of reduction steps using rule  $(AP_{abs})$ .  $\square$

We now define the notion of program skeletons. Informally, it is the collection of all possible abstract stores that can be computed using  $\rightsquigarrow$  from a program and an initial store. We get the following definition:

**Definition 13 (program skeleton)** Let  $p$  be a program of a system  $\mathcal{S} = \langle F, \mathbb{IR} \rangle$ . We call skeleton of program  $p$ , and write  $p_F^\sharp$ , the set  $\{F_i^A\}_{i \in I}$  for an index set  $I$  such that for all  $\langle F_i^A, \text{success}, \sigma_i \rangle$  reachable from  $\langle F^A, p, \perp \rangle$  using relation  $\rightsquigarrow$ , then  $i$  is in  $I$ .

Since  $\rightsquigarrow$  is strongly normalizing and since the number of rules that are applicable is always finite, we conclude that the skeleton of any program  $p$  is finite and can be computed.  $p_F^\sharp$  is the result of the secrecy analysis of  $p$ . 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  $\ell$ , and if every constraint of  $p_F^\sharp$  is compatible with  $\ell$  then the process is safe, otherwise there could be (but it is not sure) information flow from high privacy levels to lower ones. In order to precisely define this idea we start by the definition of constraint set satisfaction.

**Definition 14 (constraint set satisfaction)** We say that a constraint set  $F^A$  is satisfied by a substitution  $\mathfrak{S}$  from  $\Sigma^A$  to  $\mathcal{L}$ , iff for every rule  $l \sqsubseteq r$  of  $\mathcal{R}^A$ , then  $\mathfrak{S}(l) \sqsubseteq \mathfrak{S}(r)$  is correct, where  $\mathfrak{S}(l)$  is the natural extension of  $\mathfrak{S}$  to terms of the form  $x_1 \sqcup \dots \sqcup x_n$ .

Now we define a notion of compatibility between an abstract store and a privacy map. Informally, compatibility expresses that if the privacy map assigns  $\pi$  to a symbol  $f$  of the store, then the substitution  $\mathfrak{S}$  that assigns  $\pi$  to  $x_f$  satisfies the abstract store. In other words, the inequations of the abstract store are verified for some privacy map defined on its original store.

**Definition 15 (store compatibility)** An abstract store  $F^A$  is told compatible with a privacy map  $\ell$  iff  $\mathfrak{S}_\ell$  satisfies  $F^A$ . 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, then we claim that this program is safe w.r.t. this privacy map.

## 5 Example

We give a direct translation of an example given in [3]. It is a minimal example showing how control flow can lead to information flow. It shows the limitations of the analysis of [17] in a concurrency context, and it is a simplified version of the example given by Smith and Volpano in [16].

$\alpha$	$\Leftarrow$	$\left[ \begin{array}{l} c_\alpha = \text{TT} \Rightarrow \text{SPY} := \text{FF}; \\ c_\beta := \text{TT} \end{array} \right]; \text{success}$
$\beta$	$\Leftarrow$	$\left[ \begin{array}{l} c_\beta = \text{TT} \Rightarrow \text{SPY} := \text{TT}; \\ c_\alpha := \text{TT} \end{array} \right]; \text{success}$
$\gamma$	$\Leftarrow$	$[\text{PIN} = \text{TT} \Rightarrow c_\alpha := \text{TT}]; \text{success}$
	$+$	$[\text{PIN} = \text{FF} \Rightarrow c_\beta := \text{TT}]; \text{success}$

$PIN, SPY, c_\alpha$  and  $c_\beta$  are constants of sort *bool* defined by the two constructors  $\top$  and  $\perp$ .  $c_\alpha$  and  $c_\beta$  are initiated to  $\perp$ . Execution of process  $\alpha \parallel \beta \parallel \gamma$  copies secret value  $PIN$  into the public area, namely the variable  $SPY$ .

Now how does our analysis detect this information flow? Suppose that we have a privacy map  $\ell$  such that  $\ell(PIN) = \top$  and  $\ell(SPY) = \perp$  (with  $\top \neq \perp$ ). The abstract execution of  $\alpha \parallel \beta \parallel \gamma$  produces a collection of constraints separately generated by  $\alpha, \beta, \gamma$  (consider rules  $(AP_{\parallel \mathcal{A}})$  and  $(AP_{\equiv \mathcal{A}})$ ). Consider process  $\gamma$ , from rule  $(AP_{guard})$  and rule  $(Aea_{:=})$  (condition  $\sigma \sqsubseteq c^{\mathcal{A}}$  in definition of  $(Aea_{:=})$ ) we have the constraints that  $\top \sqsubseteq \ell(c_\alpha)$  and  $\top \sqsubseteq \ell(c_\beta)$ . On the other hand if you consider process  $\alpha$ , the same combination of rules (which imply that the assignment of a variable following a guard should be done only on variable of a higher or equal privacy level than the privacy level of the guard) leads to the constraint  $\ell(c_\alpha) \sqsubseteq \perp$ . The non trivial constraints generated by all the abstract executions are:

$$\begin{array}{lll} \ell(SPY) \sqsubseteq \ell(c_\alpha) & \ell(c_\beta) \sqsubseteq \ell(c_\alpha) & (\alpha) \\ \ell(SPY) \sqsubseteq \ell(c_\beta) & \ell(c_\alpha) \sqsubseteq \ell(c_\beta) & (\beta) \\ \ell(c_\alpha) \sqsubseteq \ell(PIN) & \ell(c_\beta) \sqsubseteq \ell(PIN) & (\gamma) \end{array}$$

It is clear that it is not possible to find a privacy map satisfying constraints generated by  $\gamma$  and  $\alpha$ . Indeed we have  $\top \sqsubseteq c_\alpha \sqsubseteq \perp$ .

## 6. Adequacy

We now prove the main result of the paper: a program is safe relatively to a privacy map if its skeleton is compatible with this privacy map. The converse is not true. There are safe programs analyzed as “unsafe” by our analysis. Just consider the following process term

$$q(x) \leftarrow + \begin{array}{l} [PIN = 12 \Rightarrow SPY := 0; \text{skip}] ; \text{success} \\ [PIN \neq 12 \Rightarrow SPY := 0; \text{skip}] ; \text{success} \end{array}$$

is analyzed as unsafe relatively to a safety map that assigns  $\top$  to  $PIN$  and  $\perp$  to  $SPY$ , whereas in fact, in this program, the final value of  $SPY$  does not depend on the actual value of  $PIN$ : there is no information flow from high levels toward low levels.

We start by stating a lemma relating 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  $\pi$  operate on data of a privacy level higher than  $\pi$ .

**Lemma 5** *Let  $\mathcal{S} = \langle F_0, \mathbb{R} \rangle$  be a system,  $p = p_0$  a program on  $\mathcal{S}$ ,  $\ell$  a privacy map for  $F$ ,  $p_{F_0}^\#$  the skeleton of program  $p_0$  and  $p_{F_0}^\#$  be compatible with  $\ell$ . Consider a transition sequence starting from  $\langle F_0, p_0 \rangle$ :*

$$\langle F_0, p_0 \rangle \xrightarrow{\alpha_1} \langle F_1, p_1 \rangle \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} \langle F_n, p_n \rangle$$

where  $\alpha_1 = [g^1 \Rightarrow a^1, \dots, a_{n_1}^1]$ , 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

$$\begin{array}{l} \langle F_0^{\mathcal{A}} \cup C_0, \mathcal{M} \rangle \xrightarrow{A} \langle F_1^{\mathcal{A}} \cup C_1, \mathcal{M}_1 \rangle \\ \xrightarrow{A} \dots \\ \xrightarrow{A} \langle F_n^{\mathcal{A}} \cup C_n, \mathcal{M}_n \rangle \end{array}$$

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

By inspection of rule  $(AP_{guard})$  and because the skeleton  $p_{F_0}^\#$  is compatible with  $\ell$ , we have that  $\ell(g^1) \sqsubseteq \sigma$  for all  $\sigma$  occurring in  $\mathcal{M}_i$  for  $i \in \{1, \dots, n\}$ . Let  $\alpha_{i_0}$  be  $[g^{i_0} \Rightarrow a_1^{i_0}; \dots; a_{n_{i_0}}^{i_0}]$ . Thus we have an abstract transition

$$\langle F^{\mathcal{A}}, a_1^{i_0}; \dots; a_{n_{i_0}}^{i_0}; \text{skip}, \sigma_{i_0} \rangle \xrightarrow{A^*} \langle F^{\mathcal{A}'}, \text{skip}, \sigma_{i_0} \rangle.$$

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

$\text{tell}(l \rightarrow r \mid c)$ : According to rule  $(Aea_{\text{tell}})$ ,  $p_{F_0}^\#$  implies  $\sigma_{i_0} \sqsubseteq \ell(l)$ , which is in contradiction to the hypothesis, since  $\ell(\text{tell}(l \rightarrow r \mid c)) = \ell(l)$ .

$\text{del}(l \rightarrow r \mid c)$ : According to rule  $(Aea_{\text{del}})$ ,  $p_{F_0}^\#$  implies  $\sigma_{i_0} \sqsubseteq \ell(l)$ , which is in contradiction to the hypothesis, since  $\ell(\text{del}(l \rightarrow r \mid c)) = \ell(l)$ .

$(f := t)$ : According to rule  $(Aea_{:=})$ ,  $p_{F_0}^\#$  implies  $\sigma_{i_0} \sqsubseteq \ell(f)$ , which is in contradiction to the hypothesis, since  $\ell(f := t) = \ell(f)$ .

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

We are now in position to prove that if a privacy map  $\ell$  satisfies a program skeleton of a program  $p$  on a store  $F$ , then  $p$  is safe with respect to  $\ell$ . Intuitively it comes from the following fact: a process is safe if it is bisimilar to itself for every privacy level, thus by contradiction if a process is not safe, it means that there is a privacy level  $\pi$ , such that for two  $\pi$ -equivalent stores  $F_0, F_1$  such that  $F_0 \cong_\pi^\ell F \cong_\pi^\ell F_1$ , there exists a derivation path from  $\langle F_0, p \rangle$  that produces a store  $F^\#$  such that there is no derivation path from  $\langle F_1, p \rangle$  that produces a  $F^\#$   $\pi$ -equivalent store. We have to show that this is not possible if  $\ell$  is compatible with  $p_{F_0}^\#$ . Basically it is the case because of the following two cases: either all guards are lower than  $\pi$  thus abstract execution ensures that the level of the elementary actions performed is less than  $\pi$  and in this case they should give the same result (lemma 1), or a guard is higher than  $\pi$  but in this case

elementary actions modify the store in a region higher than  $\pi$  thus the transformation should not be visible with respect to  $\pi\ell$ -equivalence. In both cases we find a contradiction.

**Theorem 2 (main theorem)** *Let  $\mathcal{S} = \langle F, \mathbb{IR} \rangle$  be a system,  $p$  a program on  $\mathcal{S}$ ,  $\ell$  a privacy map for  $F$ ,  $\rho_F^\sharp$  the skeleton of program  $p$ , then if  $\rho_F^\sharp$  is compatible with  $\ell$  then  $p$  is safe relatively to  $\ell$ .*

**Proof:** We reason by contradiction. Suppose that  $p$  is not safe relatively to  $\ell$ . Definition 9 implies that it exists a privacy level  $\pi$ , two stores  $F_0, F_1$  such that  $F_0 \cong_\pi^\ell F_1 \cong_\pi^\ell F$  and  $\langle F_0, p \rangle \not\cong_\pi^\ell \langle F_1, p \rangle$ .

Now from lemma 2 we have that there exist an integer  $N$  and two derivations (with  $p_0 = p$ ):

$$\begin{aligned} \langle F_0^0, p_0 \rangle &\longrightarrow \langle F_1^0, p_1 \rangle \longrightarrow \dots \longrightarrow \langle F_N^0, p_N \rangle \\ \langle F_0^1, p_0 \rangle &\longrightarrow \langle F_1^1, p_1 \rangle \longrightarrow \dots \longrightarrow \langle F_N^1, p_N \rangle \end{aligned}$$

such that for all  $j < N$ ,  $F_j^0 \cong_\pi^\ell F_j^1$  and we have two cases:

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

The first case corresponds to the analysis of [16]. A contradiction can be derived using lemma 4, and the fact that the program skeleton is satisfied. Indeed if  $F_N^0 \not\cong_\pi^\ell F_N^1$ , then it is because the last transition is done on a process term of the form  $[g \Rightarrow a_1; \dots; a_n]$ . We have

$$\begin{aligned} \langle F_{N,0}^j, a_1; \dots; a_n; \text{skip} \rangle &\hookrightarrow \langle F_{N,1}^j, a_2; \dots; a_n; \text{skip} \rangle \\ &\hookrightarrow \langle F_{N,2}^j, a_3; \dots; a_n; \text{skip} \rangle \\ &\vdots \\ &\hookrightarrow \langle F_{N,n}^j, \text{skip} \rangle \end{aligned}$$

where  $F_{N,0}^j = F_N^j$  for  $j \in \{0, 1\}$ .

Thus there must exist an elementary action  $a_i$  (for  $i \in \{1; \dots; n\}$ ) 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.

**tell( $\rho$ ):** The abstract action  $\text{tell}(\rho^A)$  has been executed during the analysis (because of lemma 4), i.e., the computation of the constraints  $\rho_F^\sharp$ . We distinguish the following two cases:

- $\rho^A \sqsubseteq \pi$ : The rule  $\rho$  added to the store is the same for both,  $F_{N,i-1}^0$  and  $F_{N,i-1}^1$ . Thus, we have that  $F_{N,i}^0 \cong_\pi^\ell F_i^1$ , in contradiction to the assumption.

$\pi \sqsubset \rho^A$ : In this case, the rules of a lower or equal privacy level than  $\pi$  are not modified, and thus we cannot have  $F_{N,i}^0 \not\cong_\pi^\ell F_{N,i}^1$ .

**del( $\rho$ ):** We distinguish two cases:

$\rho^A \sqsubseteq \pi$ : In this case, the rule  $\rho$  is present in  $F_{N,i-1}^0$  if and only if  $\rho$  is present in  $F_{N,i-1}^1$ . Thus the removal of  $\rho$  has the same effect, and we have that  $F_{N,i}^1 \cong_\pi^\ell F_{N,i}^0$ , in contradiction to the assumption.

$\pi \sqsubset \rho^A$ : Since the rules of the store which have a lower or equal privacy level than  $\pi$  are not modified by the execution of this action, we have  $F_{N,i}^0 \cong_\pi^\ell F_{N,i}^1$ , in contradiction to the assumption.

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

$c^A \sqsubseteq \pi$ : Since  $F_{N,i-1}^0 \cong_\pi^\ell F_{N,i-1}^1$ , we have also that  $\text{norm}(F_{i-1}^0, v) = \text{norm}(F_{N,i-1}^1, v)$  (using lemma 1). Consequently  $F_{N,i}^0 \cong_\pi^\ell F_{N,i}^1$ , in contradiction to the assumptions.

$\pi \sqsubset c^A$ : In this case, the assignment does modify only rules of a higher privacy level than  $\pi$ , and thus we cannot have  $F_{N,i}^0 \not\cong_\pi^\ell F_{N,i}^1$ .

The second case corresponds to the extension of [16] done in [3]. In this case the contradiction comes from the following fact : if  $F^\# \not\cong_\pi^\ell F_N^{1-j}$ , then it means that an information of a privacy level *lower or equal* to  $\pi$  has been modified, but on store  $1 - j$  an action  $\alpha$  cannot have been performed ( $\langle F_N^{1-j}, p_N \rangle \not\xrightarrow{\alpha}$ ) 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  $\pi$  (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  $\pi$  and since the skeleton of the program is satisfied, we have that actions must be done on a strictly higher level than  $\pi$ .  $\square$

## 7. Conclusion

In this paper we have addressed the question of secrecy. More precisely we have defined an 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 [8], may be used for many security issues.

Our work tackles the same problem as [3, 16]. However, the techniques we propose, namely abstract computations and constraint solving, completely depart from the

type systems used in [3, 16]. They also considered languages with severe limitations: for instance, there is no way to have recursive processes, nor to dynamically launch new processes. We have overcome these limitations in our proposition.

Notice that our algorithm can be adapted to cope with new actions. It suffices to add the appropriate definitions of  $\xrightarrow{A}$ . In addition, from the class of programs we consider, 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.

## References

- [1] M. Abadi. Secrecy by typing in security protocols. In *Proc. of 3rd Theoretical Aspects of Computer Software, LNCS 1281*, pages 611–638. Springer, 1997.
- [2] M. Abadi, N. Banerjee, N. Heintze, and J. G. Riecke. A core calculus of dependency. In *Proceedings of the 26th Annual ACM Symposium on Principles of Programming Languages*, pages 147–160, January 1999.
- [3] G. Boudol and I. Castellani. Non-interference for concurrent programs. In F. Orejas, P. Spirakis, and J. van Leeuwen, editors, *Proceedings of the 28<sup>th</sup> International Colloquium on Automata, Languages and Programming (ICALP'01)*, volume 2076 of *lns*, pages 382–395., Cesena, July 2001. Springer Verlag.
- [4] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4th ACM Symposium on Principles of Programming Languages (POPL '77)*, pages 238–252, New York, 1977. ACM Press.
- [5] 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.
- [6] R. Echahed and W. Serwe. Combining mobile processes and declarative programming. In J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, and P. J. Stuckey, editors, *Proceedings of the 1<sup>st</sup> International Conference on Computational Logic (CL 2000)*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 300 – 314, London, July 2000. Springer Verlag.
- [7] R. Echahed and W. Serwe. A component-based approach to concurrent declarative programming. In M. Hanus, editor, *Proceedings of the International Workshop on Functional and (Constraint) Logic Programming (WFLP 2001)*, pages 285 – 298, Kiel, Sept. 2001. Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität Kiel. Bericht Nr. 2017.
- [8] R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In *Proceedings of ICALP'00*, volume 1853 of *Lecture Notes in Computer Science*. Springer Verlag, 2000.
- [9] W. Fokkink. *Introduction to Process Algebra*. Texts in Theoretical Computer Science. Springer Verlag, 2000.
- [10] N. Heintze and J. G. Riecke. The SLam calculus: programming with secrecy and integrity. In *Conference Record of the Twenty-Fifth Annual ACM Symposium on Principles of Programming Languages*, pages 365–377. ACM, 1998.
- [11] M. Hennessy and J. Riely. Information flow vs. resource access in the asynchronous pi-calculus. In *Automata, Languages and Programming, 27th International Colloquium, (ICALP'2000)*, LNCS 1853, pages 415–427. Springer, 2000.
- [12] K. Honda, V. Vasconcelos, and N. Yoshida. Secure information flow as typed process behaviour. In Springer, editor, *Proc. of 9th European Symposium on Programming (ESOP'2000)*, LNCS 1782, pages 180–199, 2000.
- [13] S. Hunt. *Abstract Interpretation of functional languages : from theory to Practice*. PhD thesis, Department of Computing, Imperial College, London, 1991.
- [14] A. Mycroft. *Abstract Interpretation and Optimising Transformations for Applicative Programs*. PhD thesis, University of Edimburgh, 1981.
- [15] F. Prost. A static calculus of dependencies for the  $\lambda$ -cube. In *Proc. of IEEE 15th Ann. Symp. on Logic in Computer Science (LICS'2000)*. IEEE Computer Society Press, 2000.
- [16] G. Smith and D. M. Volpano. Secure information flow in a multi-threaded imperative language. In *Proceedings of the 25<sup>th</sup> ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '98)*, pages 355 – 364, San Diego, Jan. 1998.
- [17] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):1–21, 1996.

**Le laboratoire Leibniz est fortement pluridisciplinaire. Son activité scientifique couvre un large domaine qui comprend aussi bien des thèmes fondamentaux que des thèmes très liés aux applications, aussi bien en mathématiques qu'en informatique.**

**Les recherches sur les Environnements Informatiques d'Apprentissage Humain et la didactique des mathématiques ouvrent cette pluridisciplinarité sur les sciences humaines, elles jouent un rôle particulier en favorisant les coopérations entre différentes composantes du laboratoire.**

- \* mathématiques discrètes et recherche opérationnelle
- \* logique et mathématique pour l'informatique
- \* informatique de la connaissance
- \* EIAH et didactique des mathématiques

*Les cahiers du laboratoire Leibniz* ont pour vocation la diffusion des rapports de recherche, des séminaires ou des projets de publication réalisés par des membres du laboratoire. Au-delà, Les cahiers peuvent accueillir des textes de chercheurs qui ne sont pas membres du laboratoire Leibniz mais qui travaillent sur des thèmes proches et ne disposent pas de tels supports de publication. Dans ce dernier cas, les textes proposés sont l'objet d'une évaluation par deux membres du Comité de Rédaction.

### **Comité de rédaction**

- \* mathématiques discrètes et recherche opérationnelle  
Gerd Finke, Andrés Sebõ
- \* logique et mathématique pour l'informatique  
Ricardo Caferra, Rachid Echahed
- \* informatique de la connaissance  
Pierre Bessière, Yves Demazeau, Daniel Memmi,
- \* EIAH et didactique des mathématiques  
Nicolas Balacheff, Jean-Luc Dorier, Denise Grenier

Contact Gestion & Réalisation : Jacky Coutin  
Directeur de la publication : Nicolas Balacheff  
ISSN : 1298-020X - © laboratoire Leibniz