Confidentiality Analysis of Mobile Systems

Jérôme Feret

jerome.feret@ens.fr
Laboratoire d'Informatique de l'École Normale Supérieure
ENS-LIENS, 45, rue d'Ulm, 75230 PARIS cédex 5, FRANCE
http://www.di.ens.fr/~feret

Abstract. We propose an abstract interpretation-based analysis for automatically detecting all potential interactions between the agents of a part of a mobile system, without much knowledge about the rest of it. We restrict our study to mobile systems written in the π-calculus, and introduce a non-standard semantics which restores the link between channels and the processes that have created them. This semantics also allows to describe the interaction between a system and an unknown context. It is, to the best of our knowledge, the first analysis for this problem. We then abstract this non-standard semantics into an approximated one so as to automatically obtain a non-uniform description of the communication topology of mobile systems which compute in hostile contexts.

1 Introduction

Growing requirements of society impose the use of widely spread mobile systems of processes. In such systems the communication topology dynamically changes during processes computations, so that their analysis is a very difficult task. Furthermore, the size of systems, such as the Internet for instance, is large enough to prevent a single person from knowing the whole system. That is why we are interested in validating properties on a mobile system, which is a part of bigger one, called its context, without having precise knowledge of this context.

We address the problem of proving the confidentiality of such a mobile system: we propose to automatically infer a sound and accurate description of the topology of the interactions between the agents of this mobile system, in order to prove that private information can only be communicated to authorized agents. This description should be non-uniform, in order to distinguish between recursive instances of processes. This allows for instance to prove that in an *ftp* protocol the response to a query is returned to the correct customer.

We propose an automatic abstract interpretation-based analysis for the full π-calculus [18,19] which is a suitable formalism to describe mobile systems of processes. We present a new non-standard semantics, in the style of Venet's work [22], which mainly consists of labeling each recursive instance of processes with markers, in a deterministic way inferred during process creation. This labeling allows to trace precisely the origin of channels and to distinguish between recursive instances of processes, which cannot be done with the standard semantics. Besides, we require no further restrictions on the π-calculus. Moreover, our

semantics is general enough to approximate the potential behaviour of a small known system in any unknown context.

Several abstractions of our non-standard semantics can be deduced, using Abstract Interpretation, we propose a generic abstract semantics to describe the interactions between processes and between the system's context, in order to automatically prove confidentiality properties. The implementation of the product between this semantics and another one focusing on counting occurrences of processes during computations [13] has lead to original results.

The standard semantics is given in Sect. 3. We define the non-standard semantics in Sect. 4. We design a generic abstract analysis to validate confidentiality of systems in Sect. 5 and instantiate it in Sect. 6.

Acknowledgments. We deeply thank anonymous referees for their significant comments on an early version. We wish also to thank Patrick and Radhia Cousot, Arnaud Venet, Ian Mackie, François Maurel, David Monniaux and François Pottier, for their comments and discussions.

2 Related Work

The problem of proving the confidentiality of a mobile system has been studied extensively. Several type-based methods have been proposed to capture non-interference properties between processes in [1,16] or to study information flow properties in [15]. All these works rely on the use of a partial order of security levels, and consist in statically checking that, owing analyzed properties, low security level agents cannot violate higher security level information.

Control flow analysis only focus on the flow information. Several uniform (or monovariant) analyses have been proposed [3,4]. Non-uniform (or polyvariant) analysis allows to describe the interaction between iterations of the same recursive process. This is not achievable by type-based methods, since the same security level is inferred for all recursive instances of the same process. For instance, in [20] to prove the confidentiality of a customer-server protocol, the set of customers has to be known before the beginning of the analysis. Only very few non-uniform analyses are available. Some alias analyses to infer pointer equality relationships are non-uniform, e.g. [12] which refines a uniform analysis (trivially based on type declaration). This has been applied by Colby to design a non-uniform control flow analysis for CML in [6]. This requires a good uniform approximation of the control flow of a system before starting the non-uniform analysis.

Our study follows Venet's work on the π-calculus [22], which allows to infer a sound non-uniform description of the topology of communication between the agents of *friendly systems* [18], in where resources cannot be nested. We consider the full π-calculus with nested replications. Moreover, we consider any open system and approximate the interaction between this system and its context which, to the best of our knowledge has never been achieved in a control flow analysis of the π-calculus. In particular, contrary to [1] we can propagate interactions

without immediately reporting an error. Our framework can be easily adapted to other formalisms such as *the mobile ambients* [5], for instance.

3 π-calculus

π-calculus [18,19] is a formalism used to describe mobile systems, which is based on the use of processes and channels. We consider a lazy synchronous version of the polyadic π-calculus, inspired by the asynchronous version introduced by Turner [21] and the chemical abstract machine [2] in which communication primitives are very simple while ensuring the same expressive power. Let *Channel* be a countable set of channel names. The standard semantics of the π-calculus, given in Fig. 1, relies on the use of both a reduction relation to define results of processes computations, and a congruence relation to reveal redexes by making the processes meet.

Example 1. We model a system $\mathcal{S}$ which describes an *ftp* protocol. A resource creates repeatedly a new customer which sends a query to the server, composed with data and his email address. Data processing is abstracted away to make everything simpler. The server receives the query and returns the data back to the customer's email.

Syntax of $\mathcal{S}$ is given as follows:

$$\mathcal{S} := (\nu \text{ port})(\nu \text{ gen}) \ (\textbf{Server} \mid \textbf{Customer} \mid \text{gen}![])$$

where

$$\textbf{Server} := *\text{port}?[\mathit{info},\mathit{add}] \ (\mathit{add}![\mathit{info}])$$
$$\textbf{Customer} := *\text{gen}?[] \ ((\nu \ \mathit{data}) \ (\nu \ \mathit{email}) \ \text{port}![\mathit{data},\mathit{email}] \mid \text{gen}![])$$

For example, a short computation of $\mathcal{S}$ is given as follows:

$$\begin{aligned}
\mathcal{S} &\rightarrow (\nu \text{ port})(\nu \text{ gen})(\nu \ \mathit{data}_1)(\nu \ \mathit{email}_1) \\
&\quad (\textbf{Server} \mid \textbf{Customer} \mid \text{port}![\mathit{data}_1,\mathit{email}_1] \mid \text{gen}![] \ ) \\
&\rightarrow (\nu \text{ port})(\nu \text{ gen})(\nu \ \mathit{data}_1)(\nu \ \mathit{email}_1) \\
&\quad (\textbf{Server} \mid \textbf{Customer} \mid \mathit{email}_1![\mathit{data}_1] \mid \text{gen}![] \ ) \\
&\rightarrow (\nu \text{ port})(\nu \text{ gen})(\nu \ \mathit{data}_1)(\nu \ \mathit{email}_1) \ (\nu \ \mathit{data}_2)(\nu \ \mathit{email}_2) \\
&\quad (\textbf{Server} \mid \textbf{Customer} \mid \text{gen}![] \mid \mathit{email}_1![\mathit{data}_1] \mid \text{port}![\mathit{data}_2,\mathit{email}_2]) \\
&\rightarrow (\nu \text{ port})(\nu \text{ gen})(\nu \ \mathit{data}_1)(\nu \ \mathit{email}_1) \ (\nu \ \mathit{data}_2)(\nu \ \mathit{email}_2) \\
&\quad (\textbf{Server} \mid \textbf{Customer} \mid \text{gen}![] \mid \mathit{email}_1![\mathit{data}_1] \mid \mathit{email}_2![\mathit{data}_2]) \qquad \square
\end{aligned}$$

As illustrated in the above example, the configuration of a mobile system $\mathcal{S}$ at any stage is always congruent to one of the particular form, $(\nu \ c)(\text{P}_1 \mid ... \mid \text{P}_n)$ where c is a sequence of names, and P_1, ..., P_n are sub-processes beginning with a matching, a message, an input guard or a replication guard. Those are syntactic copies of sub-processes of $\mathcal{S}$, which have been substituted during the communications. Standard semantics does not allow to trace neither the origin of those processes, nor the origin of the channels they have declared, because of the use of α-conversion: in example 1, it is impossible to express that in the sub-process $\mathit{email}_n![\mathit{data}_n]$, channels email_n and data_n have been created by the same recursive instance of the resource **Customer**.

$$
\begin{array}{rll}
P ::= & \text{action}.P & \text{(Action)} \\
| & (P \mid P) & \text{(Parallel composition)} \\
| & \emptyset & \text{(End of a process)}
\end{array}
$$

$$
\begin{array}{rll}
\text{action} ::= & c![x_1, ..., x_n] & \text{(Message)} \\
| & c?[x_1, ..., x_n] & \text{(Input guard)} \\
| & *c?[x_1, ..., x_n] & \text{(Replication guard)} \\
| & (\nu\ x) & \text{(Channel creation)}
\end{array}
$$

where c, x_1, ..., x_n, x, $y \in \mathit{Channel}$, $n \geqslant 0$. Input guard, replication guard and channel creation are the only name binders, i.e in $c?[x_1, ..., x_n]P$ (resp. $(\nu\ x)P$), occurrences of x_1, x_2, ..., x_n (resp. x) in P are considered bound. Usual rules about scopes, substitution and α-conversion apply. We denote by $\mathcal{FN}(P)$ the set of free names of P, i.e names which are not under a scope binder and by $\mathcal{BN}(P)$ the set of bound names of P.

(a) Syntax

$$
\begin{array}{rll}
(\nu\ x)P \equiv (\nu\ y)P[x \leftarrow y] & \text{if } y \notin \mathcal{FN}(P) & (\alpha\text{-conversion}) \\
P \mid Q \equiv Q \mid P & & \text{(Commutativity)} \\
P \mid (Q \mid R) \equiv (P \mid Q) \mid R & & \text{(Associativity)} \\
(\nu\ x)(\nu\ y)P \equiv (\nu\ y)(\nu\ x)P & & \text{(Swapping)} \\
((\nu\ x)P) \mid Q \equiv (\nu\ x)(P \mid Q) & \text{if } x \notin \mathcal{FN}(Q) & \text{(Extrusion)}
\end{array}
$$

where c, x, $y \in \mathit{Channel}$

(b) Congruence relation

$$
\begin{array}{r}
c![x_1, ..., x_n]P \mid c?[y_1, ..., y_n]Q \rightarrow P \mid Q[y_1 \leftarrow x_1, ..., y_n \leftarrow x_n] \\
c![x_1, ..., x_n]P \mid *c?[y_1, ..., y_n]Q \rightarrow P \mid Q[y_1 \leftarrow x_1, ..., y_n \leftarrow x_n] \mid *c?[y_1, ..., y_n]Q
\end{array}
$$

$$
\frac{P \rightarrow Q}{(\nu\ x)P \rightarrow (\nu\ x)Q} \qquad \frac{P' \equiv P \quad P \rightarrow Q \quad Q \equiv Q'}{P' \rightarrow Q'} \qquad \frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q}
$$

where c, x_1, ..., x_n, y_1, ..., $y_n \in \mathit{Channel}$

(c) Reduction relation

Fig. 1. The chemical semantics

4 Non-standard Semantics

The non-standard semantics is a refined semantics, which aims at explicitly specifying the links between channels and the instances of processes which have declared them. Any instance of a process is identified unambiguously by a marker in order to distinguish that instance from all others. Tracing the origin of channel names is then easily done by identifying each new channel name with the marker of the process which has created it. Venet, in [22], has designed such a non-standard semantics, but its applies only to a small part of the π-calculus, called *the friendly systems* [18]. It especially requires resources not to be nested, and the system to be closed. We propose a new non-standard semantics in order to remove those restrictions.

4.1 Closed Systems

Let Lbl be an infinite set of labels. We denote by Id the set of all binary trees the leaves of which are all labeled with ε and the nodes of which are labeled with a pair (i, j) where both i and j are in Lbl. The tree having a node labeled a, a left sibling t_1 and a right one t_2 is denoted by $N(a, t_1, t_2)$.

Let us first consider the case of a closed mobile system $\mathcal{S}$ in the π-calculus (the general case will be considered in Sect 4.2) and assume without any loss of generality that two channel binders of $\mathcal{S}$ are never used on the same channel name. We locate syntactic components of $\mathcal{S}$ by marking each sign ? and ! occurring in $\mathcal{S}$ with distinct labels in Lbl, the subset of Lbl used in labeling $\mathcal{S}$ is denoted by Lbl_{used}. A non-standard configuration is a set of thread instances, where a thread instance is a triplet composed with a syntactic component, a marker and an environment. The syntactic component is a copy of a sub-process of $\mathcal{S}$, the marker is calculated at the creation of the thread and the environment specifies the semantic value of each free name of the syntactic component. Thread instances are created at the beginning of the system computation and while processes are running. In both cases, several threads are spawned, corresponding to a set of syntactic components, calculated by applying the function *Agent*, defined as follows, either on $\mathcal{S}$ for initial threads, or on the continuation of running processes.

$$\begin{aligned}
Agent(\emptyset) &= \{\} \\
Agent(x!^i[x_1, ..., x_n]P) &= \{x!^i[x_1, ..., x_n]P\} \\
Agent(y?^i[y_1, ..., y_n]P) &= \{y?^i[y_1, ..., y_n]P\} \\
Agent(*y?^i[y_1, ..., y_n]P) &= \{*y?^i[y_1, ..., y_n]P\} \\
Agent(P \mid Q) &= Agent(P) \cup Agent(Q) \\
Agent((\nu\ x)P) &= Agent(P)
\end{aligned}$$

Markers are binary trees in Id. Initial thread markers are ε, while new threads markers are calculated recursively from the marker of the threads whose computation has lead to their creation:

- when an execution does not involve fetching a resource, the marker of the computed thread is just passed to the threads in its continuation;

– when a resource is fetched, markers of new threads in its continuation are $N((i,j), id_*, id_!)$, where $(*y?^i[\overline{y}]\mathrm{P}, id_*, E_*)$ is the resource thread and $(x!^j[\overline{x}]\mathrm{Q}, id_!, E_!)$ the message thread.

Environments map each free name of syntactic components to a pair (a, b) where a is a bound name of $\mathcal{S}$ and b is a marker. Intuitively, a refers to the binder $(\nu\ a)$ which has been used in declaring the channel, and b is the marker of the thread which has declared it. While threads are running, environments are calculated in order to mimic the standard semantics.

The translation of a labeled system $\mathcal{S}$ into a set of initial threads and non-standard computation rules are given in Fig. 2. Standard and non-standard semantics are in *bisimulation*. The proof relies on that non-standard computations cannot yield conflicts between threads markers.

Example 2. We give the non-standard configuration describing the *ftp* server, just after that two connections with customers have been completed.

Labelling of $\mathcal{S}$ is given as follows:

$$\mathcal{S} := (\nu\ \mathrm{port})(\nu\ \mathrm{gen})\ (\mathbf{Server} \mid \mathbf{Customer} \mid \mathrm{gen}!^6[])$$

where

$$\mathbf{Server} := *\mathrm{port}?^1[info, add]\ (add!^2[info])$$
$$\mathbf{Customer} := *\mathrm{gen}?^3[]\ ((\nu\ data)\ (\nu\ email)\ \mathrm{port}!^4[data, email] \mid \mathrm{gen}!^5[])$$

The non-standard configuration is given as follows, each sub-process is denoted by the first label occurring in its syntax.

$$\left\{ \begin{array}{l} \left(1, \varepsilon, \left\{ \mathrm{port} \mapsto (\mathrm{port}, \varepsilon) \right.\right) \\ \left(3, \varepsilon, \left\{ \begin{array}{ll} \mathrm{gen} & \mapsto (\mathrm{gen}, \varepsilon) \\ \mathrm{port} & \mapsto (\mathrm{port}, \varepsilon) \end{array} \right.\right) \\ \left(2, id_1', \left\{ \begin{array}{ll} add & \mapsto (email, id_1) \\ info & \mapsto (data, id_1) \end{array} \right.\right) \\ \left(2, id_2', \left\{ \begin{array}{ll} add & \mapsto (email, id_2) \\ info & \mapsto (data, id_2) \end{array} \right.\right) \\ \left(5, id_2, \left\{ \mathrm{gen} \mapsto (\mathrm{gen}, \varepsilon) \right.\right) \end{array} \right\} \text{ where } \left\{ \begin{array}{l} id_1 = N((3,6), \varepsilon, \varepsilon) \\ id_1' = N((1,4), \varepsilon, id_1) \\ id_2 = N((3,5), \varepsilon, id_1) \\ id_2' = N((1,4), \varepsilon, id_2) \end{array} \right.$$

We shall remark that, since $\mathcal{S}$ has no embedded resources, markers are all sequences, instead of trees. It explicitly appears that each time a copy of sub-process 2 is created, both channels *add* and *info* are bound to two channel names created by the same replication of the resource 3. □

4.2 Interactions with a Context

We now extend our non-standard semantics to open systems. An open system $\mathcal{S}$ is a part of a bigger closed system, the rest of which is called its context. The context is a set of processes, concurrently running with the processes of $\mathcal{S}$. We

$$C_0(\mathcal{S}) = \{(p, \varepsilon, E_p) \mid p \in Agent(\mathcal{S})\}\,,\ \text{where } E_p = \begin{cases} \mathcal{FN}(p) & \to \mathcal{BN}(\mathcal{S}) \times Id \\ x & \mapsto (x, \varepsilon) \end{cases}.$$

(a) C_0 calculates the set of initial threads

Let C be a non-standard configuration,
if there are $\lambda, \mu \in C$,
with $\lambda{=}(y?^i[y_1, ..., y_n]P, id_?, E_?)$ and $\mu{=}(x!^j[x_1, ..., x_n]Q, id_!, E_!)$,
such that $E_?(y) = E_!(x)$,
then $C \to_2 C'$,
where $C' = (C \setminus \{\lambda, \mu\}) \cup (f_?(Agent(P))) \cup (f_!(Agent(Q)))$,

$$f_? : Ag \mapsto \left(Ag, id_?, \begin{cases} z \mapsto E_?(z) & \text{if } z \in \mathcal{FN}(Ag) \cap \mathcal{FN}(y?^i[y_1, ..., y_n]P) \\ y_k \mapsto E_!(x_k) & \text{if } y_k \in \mathcal{FN}(Ag) \\ z \mapsto (z, id_?) & \text{if } \begin{cases} z \in \mathcal{FN}(Ag) \cap \mathcal{BN}(y?^i[y_1, ..., y_n]P) \\ z \notin \{y_k \mid k \in [|1; n|]\} \end{cases} \end{cases} \right)$$

$$\text{and } f_! : Ag \mapsto \left(Ag, id_!, \begin{cases} z \mapsto E_!(z) & \text{if } z \in \mathcal{FN}(Ag) \cap \mathcal{FN}(x!^j[x_1, ..., x_n]Q) \\ z \mapsto (z, id_!) & \text{if } z \in \mathcal{FN}(Ag) \cap \mathcal{BN}(x!^j[x_1, ..., x_n]Q) \end{cases} \right).$$

(b) Non-standard communication

Let C be a non-standard configuration,
if there are $\lambda, \mu \in C$,
with $\lambda{=}(*y?^i[y_1, ..., y_n]P, id_?, E_?)$ and $\mu = (x!^j[x_1, ..., x_n]Q, id_!, E_!)$,
such that $E_?(y) = E_!(x)$,
then $C \to_2 C'$,
where $C' = (C \setminus \{\mu\}) \cup (f_?(Agent(P))) \cup (f_!(Agent(Q)))$,
$id_* = N((i, j), id_?, id_!)$

$$f_? : Ag \mapsto \left(Ag, id_*, \begin{cases} z \mapsto E_?(z) & \text{if } z \in \mathcal{FN}(Ag) \cap \mathcal{FN}(y?^i[y_1, ..., y_n]P) \\ y_k \mapsto E_!(x_k) & \text{if } y_k \in \mathcal{FN}(Ag) \\ z \mapsto (z, id_*) & \text{if } \begin{cases} z \in \mathcal{FN}(Ag) \cap \mathcal{BN}(y?^i[y_1, ..., y_n]P) \\ z \notin \{y_k \mid k \in [|1; n|]\} \end{cases} \end{cases} \right)$$

$$\text{and } f_! : Ag \mapsto \left(Ag, id, \begin{cases} z \mapsto E_!(z) & \text{if } z \in \mathcal{FN}(Ag) \cap \mathcal{FN}(x!^j[x_1, ..., x_n]Q) \\ z \mapsto (z, id_!) & \text{if } z \in \mathcal{FN}(Ag) \cap \mathcal{BN}(x!^j[x_1, ..., x_n]Q) \end{cases} \right).$$

(c) Non-standard resource fetching

Fig. 2. Non-standard semantics for close systems

represent this context by the set of names it shares with $\mathcal{S}$, called unsafe names, and approximate it as an intruder which is able to form any possible process working on these channel names. An interaction between $\mathcal{S}$ and its context, may only consist in a communication between a process $p_{\mathcal{S}}$ of the first one and a process p_{cont} of the second one, via an unsafe name. This communication is called spying when p_{cont} is the receiver, and spoiling when p_{cont} is the message sender. While spying, the context listens to new channel names which get unsafe. While spoiling, the context may pass any name to $\mathcal{S}$, either an unsafe name created by a binder of $\mathcal{S}$ or a name created by the context itself, thus we have to introduce an infinite set of unsafe names that the context may have created. At last, spoiling may lead to a resource fetching, which would require to create an unambiguous marker, otherwise the consistency of the semantics would not be preserved.

Since α-conversion allows us to choose the names of new channels created by the context, we may assume those channels have been declared by recursive instances of a single process. Choosing $cont_?,\ cont_! \in Lbl \setminus Lbl_{used}$ and $ext \in Channel \setminus \mathcal{BN}(\mathcal{S})$, such channels will be seen as if they were created by the binder $(\nu\ ext)$ of a recursive instance of a process whose marker is t_n, where t_n is recursively defined as follows:

$$\begin{cases} t_0 = N((cont_?,\ cont_!), \varepsilon, \varepsilon) \\ t_{n+1} = N((cont_?,\ cont_!), \varepsilon, t_n) \end{cases}.$$

We then denote by $\mathcal{EN}$ the set $\{(ext, t_n) \mid n \in \mathbb{N}\}$ and assume that every spoiling message are syntactic copies of a single process, whose first sign is labeled with $cont_!$. The coherence of our semantics mainly relies on the fact that during a computation, there cannot have been two different instances of a single process with the same marker. We guarantee this property by associating to each spoiling message an hypothetical fresh marker t_n.

A non-standard configuration is now a triplet (C, *Unsafe*, *Unused*), where C is a set of threads, *Unsafe* is a set of pairs (a,id), such that channel names created by the binder $(\nu\ a)$ of the recursive instance of a process whose marker was id is unsafe, and *Unused* is a set of fresh markers which have not been used as markers for spoiling message. $\mathcal{S}$ may start with several initial configurations, since free names have to be chosen among the set of initial unsafe names. The transition relation $\overset{\tau}{\hookrightarrow}$ holds both with computations inside the mobile system $\mathcal{S}$ and computations involving the system $\mathcal{S}$ and its context. Initial non-standard configurations and computation rules are given in Figs. 3 and 4.

$$\begin{array}{l} (\{(p, \varepsilon, E_p) \mid p \in Agent(\mathcal{S})\}, \mathcal{EN}, \{t_n \mid n \in \mathbb{N}\}) \in C_0(\mathcal{S}) \\ \iff \begin{cases} \forall p \in Agent(\mathcal{S}),\ \forall x \in \mathcal{FN}(p), \\ \quad E_p(x) = (x, \varepsilon)\ \textit{if}\ x \in \mathcal{BN}(\mathcal{S}), \\ \quad \exists n\ \textit{such that}\ E_p(x) = (ext, t_n)\ \textit{otherwise} \end{cases} \end{array}$$

Fig. 3. Non-standard initial configurations for open systems

$$\frac{C \rightarrow_2 C'}{(C,\, \mathit{Unsafe},\, \mathit{Unused}) \looparrowright (C',\, \mathit{Unsafe},\, \mathit{Unused})}$$

(a) Non-standard safe transitions

Let $(C, \mathit{Unsafe}, \mathit{Unused})$ be a non-standard configuration,
if there is $\lambda \in C$, with $\lambda{=}(x!^j[x_1, ..., x_n]P, id_!, E_!)$,
such that $E_!(x) \in \mathit{Unsafe}$,
then $(C, \mathit{Unsafe}, \mathit{Unused}) \looparrowright (C', \mathit{Unsafe}', \mathit{Unused})$,
where $C' = (C \setminus \{\lambda\}) \cup (f_!(\mathit{Agent}(P)))$,

$$f_! : Ag \mapsto \left(Ag, id_!, \begin{cases} z \mapsto E(z) & \text{if } z \in \mathcal{FN}(Ag) \cap \mathcal{FN}(x!^j[x_1, ..., x_n]P) \\ z \mapsto (z, id_!) & \text{if } z \in \mathcal{FN}(Ag) \cap \mathcal{BN}(x!^j[x_1, ..., x_n]P) \end{cases} \right)$$

and $\mathit{Unsafe}' = \mathit{Unsafe} \cup \{E(x_k) \mid k \in [|1; n|]\}$.

(b) Non-standard spied communication

Let $(C, \mathit{Unsafe}, \mathit{Unused})$ be a non-standard configuration,
if there are $\lambda \in C$, with $\lambda{=}(y?^i[y_1, ..., y_n]P, id_?, E_?)$ and $u_1, ..., u_n \in \mathit{Unsafe}$,
such that $E_?(y) \in \mathit{Unsafe}$,
then $(C, \mathit{Unsafe}, \mathit{Unused}) \looparrowright (C', \mathit{Unsafe}, \mathit{Unused})$,
where $C' = (C \setminus \{\lambda\}) \cup (f_?(\mathit{Agent}(P)))$,

$$f_? : Ag \mapsto \left(Ag, id_?, \begin{cases} z \mapsto E_?(z) & \text{if } z \in \mathcal{FN}(Ag) \cap \mathcal{FN}(y?^i[y_1, ..., y_n]P) \\ y_k \mapsto u_k & \text{if } y_k \in \mathcal{FN}(Ag) \\ z \mapsto (z, id_?) & \text{if } \begin{cases} z \in \mathcal{FN}(Ag) \cap \mathcal{BN}(y?^i[y_1, ..., y_n]P) \\ z \notin \{y_k \mid k \in [|1; n|]\} \end{cases} \end{cases} \right).$$

(c) Non-standard spoilt communication

Let $(C, \mathit{Unsafe}, \mathit{Unused})$ be a non-standard configuration,
if there are $\lambda \in C$, with $\lambda{=}(*y?^i[y_1, ..., y_n]P, id_?, E_?)$, $u_1, ..., u_n \in \mathit{Unsafe}$, $id \in \mathit{Unused}$,
such that $E_?(y) \in \mathit{Unsafe}$,
then $(C, \mathit{Unsafe}, \mathit{Unused}) \looparrowright (C', \mathit{Unsafe}\ , \mathit{Unused}')$
where $C' = C \cup (f_?(\mathit{Agent}(P)))$,
$id_* = N((i, cont_!), id_?, id)$,

$$f_? : Ag \mapsto \left(Ag, id_*, \begin{cases} z \mapsto E_?(z) & \text{if } z \in \mathcal{FN}(Ag) \cap \mathcal{FN}(y?^i[y_1, ..., y_n]P) \\ y_k \mapsto u_k & \text{if } y_k \in \mathcal{FN}(Ag) \\ z \mapsto (z, id_*) & \text{if } \begin{cases} z \in \mathcal{FN}(Ag) \cap \mathcal{BN}(y?^i[y_1, ..., y_n]P) \\ z \notin \{y_k \mid k \in [|1; n|]\} \end{cases} \end{cases} \right),$$

and $\mathit{Unused}' = \mathit{Unused} \setminus \{id\}$.

(d) Non-standard spoilt resource fetching

Fig. 4. Non-standard transitions for open systems

5 Abstract Semantics

We denote by $\mathcal{C}$ the set of all possible non-standard configurations. The set of all possible non-standard configurations a system may take during a finite computation is given by its collecting semantics [7], and can be expressed as the least fix point of the $\cup$-complete endomorphism $\mathbb{F}$ on the complete lattice $(\wp(\mathcal{C}), \subseteq, \cup, \emptyset, \cap, \mathcal{C})$ defined as follows:

$$\mathbb{F}(X) = C_0(\mathcal{S}) \cup \{C \mid \exists C' \in X \ : \ C' \nrightarrow C\} \ .$$

Usually, this semantics is not decidable, we use abstract interpretation [8] to design an abstract domain in which a description of the collecting semantics will be finitely designed. We introduce two lattices, left as parameter of our abstraction, $(Id_1^\sharp, \sqsubseteq_1, \cup_1, \bot_1, \cap_1, \top_1)$ and $(Id_2^\sharp, \sqsubseteq_2, \cup_2, \bot_2, \cap_2, \top_2)$, to respectively represent sets of markers and sets of pairs of markers, related to their concrete domains via two monotone maps, γ_1 and γ_2:

$$\begin{array}{r} \gamma_1 \ : \ (Id_1^\sharp, \sqsubseteq_1) \rightarrow (\wp(Id), \subseteq), \text{ with } \gamma_1(\bot_1) = \emptyset \ , \\ \gamma_2 \ : \ (Id_2^\sharp, \sqsubseteq_2) \rightarrow (\wp(Id \times Id), \subseteq) \text{ with } \gamma_2(\bot_2) = \emptyset \ . \end{array}$$

Let Pro be the set of all sub-processes of $\mathcal{S}$, and Can be the set $\{(p, x, y) \mid p \in Pro, \ x \in \mathcal{FN}(p), \ \ y \in \{ext\} \cup \mathcal{BN}(\mathcal{S})\}$. Can is the set of all possible interactions between agents of $\mathcal{S}$. The interaction (p, x, y) denotes the fact that the channel name $x \in \mathcal{FN}(p)$ has been declared by the $(\nu \ y)$ binder, if $y \neq ext$, or has been declared by the context otherwise. Our abstract domain is then the product of three functional domains:

$$\mathcal{C}^\sharp = (C_{Pro}^\sharp \times C_{Com}^\sharp \times C_{Esc}^\sharp) \text{ where } \left\{ \begin{array}{rcl} C_{Pro}^\sharp : & Pro & \rightarrow Id_1^\sharp \\ C_{Com}^\sharp : & Can & \rightarrow Id_2^\sharp \\ C_{Esc}^\sharp : & \mathcal{BN}(\mathcal{S}) \cup \{ext\} & \rightarrow Id_1^\sharp \end{array} \right. .$$

$C_{Pro}^\sharp$ maps each sub-process to the set of markers it may be marked with. $C_{Com}^\sharp$ maps each interaction (P, x, y) to the set of pairs of markers (id_1, id_2), such that x is the free name of a thread whose marker was id_1; this thread may have been declared by the $(\nu \ y)$ binder of a thread whose marker was id_2. Finally $C_{Esc}^\sharp$ maps each names x to the set of markers id such that a channel name declared by the binder $(\nu \ x)$ of a thread whose marker was id may be unsafe.

An abstract configuration $C^\sharp = (f_{pro}^\sharp, f_{can}^\sharp, f_{esc}^\sharp)$ is then related to a set of concrete configurations by the monotone map γ, where $\gamma(C^\sharp)$ is:

$$\left\{ A \in \wp(\mathcal{C}) \ \middle| \ \begin{array}{l} \forall (C, esc, unused) \in A, \\ \left\{ \begin{array}{lcl} \exists (x, id) \in esc & \Longrightarrow & id \in \gamma_1(f_{esc}^\sharp(x)) \\ \exists (P, id, E) \in C, & \Longrightarrow & id \in \gamma_1(f_{pro}^\sharp(P)) \\ \left(\begin{array}{l} \exists (P, id, E) \in C, \\ \exists x \in \mathcal{FN}(P) \end{array} \right) & \Longrightarrow & \left(\begin{array}{c} (id, id') \in \gamma_2(f_{can}^\sharp(P, x, y)) \\ where \ E(x) = (y, id') \end{array} \right) \end{array} \right. \end{array} \right\} .$$

The abstract semantics is given by an initial abstract configuration $C_0^\sharp$ and a transition relation $\rightharpoonup$, in Figs. 5 to 10. In the relation $C_1^\sharp \rightharpoonup C_2^\sharp$, $C_2^\sharp$ represents new interactions between agents induced by an abstract computation. The transition relation $\rightharpoonup$ uses several abstract primitives, which must satisfy some soundness conditions:

- two abstract projections: $\mathit{\Pi}_1 : Id_2^\sharp \to Id_1^\sharp$ and $\mathit{\Pi}_2 : Id_2^\sharp \to Id_1^\sharp$,
  such that $\{u \in Id \mid \exists v \in Id \text{ such that } (u,v) \in \gamma_2(c)\} \subseteq \gamma_1(\mathit{\Pi}_1(c))$
  and $\{v \in Id \mid \exists u \in Id \text{ such that } (u,v) \in \gamma_2(c)\} \subseteq \gamma_1(\mathit{\Pi}_2(c))$;
- two abstract injections: $\mathit{Inj}_1 : Id_1^\sharp \to Id_2^\sharp$ and $\mathit{Inj}_2 : Id_1^\sharp \to Id_2^\sharp$,
  such that $\{(u,v) \in Id \times Id \mid u \in \gamma_1(c)\} \subseteq \gamma_2(\mathit{Inj}_1(c))$
  and $\{(u,v) \in Id \times Id \mid v \in \gamma_1(c)\} \subseteq \gamma_2(\mathit{Inj}_2(c))$;
- two abstract joins: $\underset{=\rightarrow}{\bowtie} : Id_2^\sharp \times Id_2^\sharp \to Id_2^\sharp$ and $\underset{\leftarrow=}{\bowtie} : Id_2^\sharp \times Id_2^\sharp \to Id_2^\sharp$,
  such that $\left\{(u,v) \in Id \times Id \;\middle|\; \exists w \in \Sigma^* \begin{matrix}(w,u) \in \gamma_2(c)\\(w,v) \in \gamma_2(c')\end{matrix}\right\} \subseteq \gamma_2(\underset{=\rightarrow}{\bowtie}(c,c'))$
  and $\left\{(u,v) \in Id \times Id \;\middle|\; \exists w \in Id \begin{matrix}(v,w) \in \gamma_2(c)\\(u,w) \in \gamma_2(c')\end{matrix}\right\} \subseteq \gamma_2(\underset{\leftarrow=}{\bowtie}(c,c'))$;
- two abstract marker creators: $\mathit{push}_1^{(i,j)} : Id_1^\sharp \times Id_1^\sharp \to Id_1^\sharp$
  and $\mathit{push}_2^{(i,j)} : Id_1^\sharp \times Id_2^\sharp \to Id_2^\sharp$,
  such that $\left\{N((i,j),u,v) \;\middle|\; \begin{matrix}u \in \gamma_1(c_g)\\v \in \gamma_1(c_d)\end{matrix}\right\} \subseteq \gamma_1(\mathit{push}_1^{(i,j)}(c_g,c_d))$
  and $\left\{(N((i,j),u,v),w) \;\middle|\; \begin{matrix}u \in \gamma_1(c_g)\\(v,w) \in \gamma_2(c_d)\end{matrix}\right\} \subseteq \gamma_2(\mathit{push}_2^{(i,j)}(c_g,c_d))$;
- an abstract marker duplicator: $\mathit{dpush} : Id_1^\sharp \to Id_2^\sharp$,
  such that $\{(u,u) \mid u \in \gamma_1(c)\} \subseteq \gamma_2(\mathit{dpush}(c))$;
- initial abstract markers: $\varepsilon_1^\sharp \in Id_1^\sharp$, $\varepsilon_2^\sharp \in Id_2^\sharp$, $t^\sharp \in Id_1^\sharp$ such that
  $\{\varepsilon\} \subseteq \gamma_1(\varepsilon_1^\sharp)$, $\{(\varepsilon,\varepsilon)\} \subseteq \gamma_2(\varepsilon_2^\sharp)$ and $\{t_n \mid n \in \mathbb{N}\} \subseteq \gamma_1(t^\sharp)$.

Roughly speaking, $\mathit{\Pi}_1$ (resp. $\mathit{\Pi}_2$) projects every pair of markers onto its first (resp. second) component. Inj_1 (resp. Inj_2) constructs for every marker the set of pair whose first (resp. second) component is that marker. $\underset{\leftarrow=}{\bowtie}$ and $\underset{=\rightarrow}{\bowtie}$ allow to calculate and propagate relational information between the components of a pair of markers throughout communication and resource fetching computations. $\mathit{push}_1^{(i,j)}$ constructs every marker of the processes created during a replication, using the set of markers of the resource given in its first argument and the set of markers of the message sender given in its second argument. $\mathit{push}_2^{(i,j)}$ acts similarly to $\mathit{push}_1^{(i,j)}$, but it takes into account relational information about markers of the free channels names of the message sender. *dpush* duplicates every marker into a pair of markers, this is used while creating new channel names.

All these conditions imply the following proposition:

Proposition 1. *If $C \in \gamma(C^\sharp)$ and $C \leadsto D$, then there exists $D^\sharp$ such that $C^\sharp \rightharpoonup D^\sharp$ and $D \in \gamma(C^\sharp \sqcup D^\sharp)$.*

$$C_0^\sharp = (i^\sharp_{pro}, i^\sharp_{can}, i^\sharp_{esc})$$

$$\text{where}\begin{cases} i^\sharp_{pro} = \begin{cases} Pro \to \mathit{Id}^\sharp_1 \\ P \mapsto \varepsilon^\sharp_1 \ \ \text{if } P \in (\mathit{Agent}(\mathcal{S})) \\ P \mapsto \bot^\sharp_1 \ \ \text{otherwise} \end{cases} \\ i^\sharp_{can} = \begin{cases} Can \to \mathit{Id}^\sharp_2 \\ (P,x,x) \mapsto \varepsilon^\sharp_2 & \text{if } \begin{cases} P \in (\mathit{Agent}(\mathcal{S})) \\ x \in \mathcal{BN}(\mathcal{S}) \cap \mathcal{FN}(P) \end{cases} \\ (P,x,ext) \mapsto (\mathit{Inj}_1(\varepsilon^\sharp_1)) \cap_2 (\mathit{Inj}_2(t^\sharp)) & \text{if } \begin{cases} P \in (\mathit{Agent}(\mathcal{S})) \\ x \in \mathcal{FN}(\mathcal{S}) \cap \mathcal{FN}(P) \end{cases} \\ (P,x,y) \mapsto \bot^\sharp_2 & \text{otherwise} \end{cases} \\ i^\sharp_{esc} = \begin{cases} \mathcal{BN}(\mathcal{S}) \cup \{ext\} \to \mathit{Id}^\sharp_1 \\ ext \mapsto t^\sharp \\ x \mapsto \bot^\sharp_1 \ \text{if } x \in \mathcal{BN}(\mathcal{S}) \end{cases} \end{cases}$$

Fig. 5. Initial abstract configuration

As a consequence, the abstract counterpart $\mathbb{F}^\sharp$ of $\mathbb{F}$, defined by

$$\mathbb{F}^\sharp(C^\sharp) = C_0^\sharp \sqcup C^\sharp \sqcup \{\overline{C}^\sharp \mid C^\sharp \rightharpoonup \overline{C}^\sharp\},$$

satisfies the soundness condition $\mathbb{F} \circ \gamma \subseteq \gamma \circ \mathbb{F}^\sharp$. Using Kleene's theorem, we prove the soundness of our analysis:

Theorem 1. $\mathit{lfp}_\emptyset \mathbb{F} \subseteq \bigcup_{n\in\mathbb{N}} \gamma({\mathbb{F}^\sharp}^n(\bot_\sharp))$

We compute a sound approximation of our abstract semantics by using a widening operator [7,9] $\nabla : C^\sharp \times C^\sharp \to C^\sharp$ satisfying the following properties:

- $\forall C_1^\sharp,\ C_2^\sharp \in C^\sharp,\ C_1^\sharp \sqcup C_2^\sharp \sqsubseteq C_1^\sharp \nabla C_2^\sharp$
- for all increasing sequence $(C_n^\sharp) \in \left(C^\sharp\right)^{\mathbb{N}}$, the sequence (C_n^∇) defined as

$$\begin{cases} C_0^\nabla = C_0^\sharp \\ C_{n+1}^\nabla = C_n^\nabla \nabla C_{n+1}^\sharp \end{cases}$$

  is ultimately stationary.

We can easily construct a widening operator ∇ on our abstract domain from existing widening operators ∇_1 on Id_1 and ∇_2 on Id_2.

Theorem 2. ***Abstract iteration[9,10]*** *The abstract iteration* (C_n^∇)*of* $\mathbb{F}^\sharp$ *defined as follows*

$$\begin{cases} C_0^\nabla = \bot \\ C_{n+1}^\nabla = \begin{cases} C_n^\nabla & \textit{if } \mathbb{F}^\sharp(C_n^\nabla) \sqsubseteq C_n^\nabla \\ C_n^\nabla \nabla \mathbb{F}^\sharp(C_n^\nabla) & \textit{otherwise} \end{cases} \end{cases}$$

is ultimately stationary and its limit C^∇ *satisfies* $\mathit{lfp}_\emptyset \mathbb{F} \subseteq \gamma(C^\nabla)$.

Let $(f^\sharp_{pro}, f^\sharp_{can}, f^\sharp_{esc})$ be an abstract configuration,
we consider $u \in \mathit{Channel}$ and $y?^i[y_1,...,y_n]P$, $x!^j[x_1,...,x_n]Q$ two sub-processes, such that

$f^\sharp_{can}(y?^i[y_1,...,y_n]P, y, u) = id^?_\sharp$,
$f^\sharp_{can}(x!^j[x_1,...,x_n]Q, x, u) = id^!_\sharp$,
$\bot_1 \neq (\Pi_2(id^?_\sharp)) \cap_1 (\Pi_2(id^!_\sharp))$,

and we introduce

$id_{can} \triangleq (\Pi_2(id^?_\sharp)) \cap_1 (\Pi_2(id^!_\sharp))$,
$idpro^? \triangleq \Pi_1(id^?_\sharp \cap_2 \mathit{Inj}_2(id_{can}))$,
$idpro^! \triangleq \Pi_1(id^!_\sharp \cap_2 \mathit{Inj}_2(id_{can}))$,
$id^t_k \triangleq \underset{= \rightarrow}{\bowtie} (\underset{\leftarrow =}{\bowtie} (id^?_\sharp, id^!_\sharp), f^\sharp_{can}(x!^j[x_1,...,x_n]Q, x_k, t)) \cap_2 \mathit{Inj}_1(idpro^?)$.

Then we have

$(f^\sharp_{pro}, f^\sharp_{can}, f^\sharp_{esc}) \rightharpoonup (g^\sharp_{pro}, g^\sharp_{can}, \emptyset)$

where $g^\sharp_{pro} = \begin{cases} p \mapsto idpro^? & \text{if } p \in (\mathit{Agent}(P)) \\ q \mapsto idpro^! & \text{if } q \in (\mathit{Agent}(Q))\} \end{cases}$

and $g^\sharp_{can} =$

$$\begin{cases} (p,z,z) \mapsto \mathit{dpush}(idpro^?) & \text{if } \begin{cases} p \in (\mathit{Agent}(P)) \\ z \in \mathcal{BN}(P) \cap \mathcal{FN}(p) \\ z \notin \{y_i \mid i \in [|1;n|]\} \end{cases} \\ (p,y_k,t) \mapsto id^t_k & \text{if } \begin{cases} p \in (\mathit{Agent}(P)) \\ y_k \in \mathcal{BN}(P) \cap \mathcal{FN}(p) \end{cases} \\ (p,z,t) \mapsto f^\sharp_{can}(y?^i[y_1,...,y_n]P, z, t) \cap_2 \mathit{Inj}_1(idpro^?) & \text{if } \begin{cases} p \in (\mathit{Agent}(P)) \\ z \in \mathcal{FN}(P) \cap \mathcal{FN}(p) \\ z \neq y \end{cases} \\ (p,y,u) \mapsto f^\sharp_{can}(y?^i[y_1,...,y_n]P, y, u) \cap_2 \mathit{Inj}_2(id_{can}) & \text{if } \begin{cases} p \in (\mathit{Agent}(P)) \\ y \in \mathcal{FN}(P) \cap \mathcal{FN}(p) \end{cases} \\ (q,z,z) \mapsto \mathit{dpush}(idpro^!) & \text{if } \begin{cases} p \in (\mathit{Agent}(Q)) \\ z \in \mathcal{BN}(Q) \cap \mathcal{FN}(q) \end{cases} \\ (q,z,t) \mapsto f^\sharp_{can}(x!^j[x_1,...,x_n]Q, z, t) \cap_2 \mathit{Inj}_1(idpro^!) & \text{if } \begin{cases} q \in (\mathit{Agent}(Q)) \\ z \in \mathcal{FN}(Q) \cap \mathcal{FN}(q) \\ z \neq x \end{cases} \\ (q,x,u) \mapsto f^\sharp_{can}(x!^j[x_1,...,x_n]Q, x, u) \cap_2 \mathit{Inj}_2(id_{can}) & \text{if } \begin{cases} p \in (\mathit{Agent}(Q)) \\ x \in \mathcal{FN}(Q) \cap \mathcal{FN}(q). \end{cases} \end{cases}$$

Fig. 6. Abstract communication

Let $(f^\sharp_{pro}, f^\sharp_{can}, f^\sharp_{esc})$ be an abstract configuration,
we consider $u \in \mathit{Channel}$ and $*y?^i[y_1,...,y_n]P$, $x!^j[x_1,...,x_n]Q$ two sub-processes, such that

$f^\sharp_{can}(*y?^i[y_1,...,y_n]P, y, u) = id^?_\sharp$,
$f^\sharp_{can}(x!^j[x_1,...,x_n]Q, x, u) = id^!_\sharp$,
$\bot_1 \neq (\Pi_2(id^?_\sharp)) \cap_1 (\Pi_2(id^!_\sharp))$,

and we introduce

$id_{can} \triangleq (\Pi_2(id^?_\sharp)) \cap_1 (\Pi_2(id^!_\sharp))$,
$idpro^? \triangleq \Pi_1(id^?_\sharp \cap_2 \mathit{Inj}_2(id_{can}))$,
$idpro^! \triangleq \Pi_1(id^!_\sharp \cap_2 \mathit{Inj}_2(id_{can}))$,
$id_{s,t} \triangleq \mathit{Inj}_1(idpro^!) \cap_2 \underset{= \rightarrow}{\bowtie} (\underset{\leftarrow =}{\bowtie} (id^!_\sharp, id^?_\sharp), f^\sharp_{can}(y?^i[y_1,...,y_n]P, s, t))$,
$id^t_k \triangleq f^\sharp_{can}(x!^j[x_1,...,x_n]Q, x_k, t)$.

Then we have

$$(f^\sharp_{pro}, f^\sharp_{can}, f^\sharp_{esc}) \rightharpoonup (g^\sharp_{pro}, g^\sharp_{can}, \emptyset)$$

where $g^\sharp_{pro} = \begin{cases} p \mapsto idpro^? \mid p \in (\mathit{Agent}(P)) \\ q \mapsto idpro^! \mid q \in (\mathit{Agent}(Q)) \end{cases}$

and $g^\sharp_{can} =$

$$\begin{cases}
(p,z,z) \mapsto dpush(push_1^{(i,j)}(idpro^?, idpro^!)) & \text{if} \begin{cases} p \in (\mathit{Agent}(P)) \\ z \in \mathcal{BN}(P) \cap \mathcal{FN}(p) \\ z \notin \{y_i \mid i \in [|1;n|]\} \end{cases} \\
(p,y_k,t) \mapsto push_2^{(i,j)}(idpro^?, id^t_k) & \text{if} \begin{cases} p \in (\mathit{Agent}(P)) \\ y_k \in \mathcal{BN}(P) \cap \mathcal{FN}(p) \end{cases} \\
(p,z,t) \mapsto push_2^{(i,j)}(idpro^?, id_{z,t}) & \text{if} \begin{cases} p \in (\mathit{Agent}(P)) \\ z \in \mathcal{FN}(P) \cap \mathcal{FN}(p) \\ z \neq y \end{cases} \\
(p,y,u) \mapsto push_2^{(i,j)}(idpro^?, id_{y,u}) \cap_2 \mathit{Inj}_2(id_{can}) & \text{if} \begin{cases} p \in (\mathit{Agent}(P)) \\ y \in \mathcal{FN}(P) \cap \mathcal{FN}(p) \end{cases} \\
(q,z,z) \mapsto dpush(idpro^!) & \text{if} \begin{cases} q \in (\mathit{Agent}(Q)) \\ z \in \mathcal{BN}(Q) \cap \mathcal{FN}(q) \end{cases} \\
(q,z,t) \mapsto f^\sharp_{can}(x!^j[x_1,...,x_n]Q, z, t) \cap_2 \mathit{Inj}_1(idpro^!) & \text{if} \begin{cases} q \in (\mathit{Agent}(Q)) \\ z \in \mathcal{FN}(Q) \cap \mathcal{FN}(q) \\ z \neq x \end{cases} \\
(q,x,u) \mapsto f^\sharp_{can}(x!^j[x_1,...,x_n]Q, x, u) \cap_2 \mathit{Inj}_2(id_{can}) & \text{if} \begin{cases} q \in (\mathit{Agent}(Q)) \\ x \in \mathcal{FN}(Q) \cap \mathcal{FN}(q). \end{cases}
\end{cases}$$

Fig. 7. Abstract resource fetching

Let $(f^\sharp_{pro}, f^\sharp_{can}, f^\sharp_{esc})$ be an abstract configuration,
we consider $u \in \mathit{Channel}$ and $x!^j[x_1, ..., x_n]Q$ a sub-process, such that
$f^\sharp_{can}(x!^j[x_1, ..., x_n]Q, x, u) = id^!_\sharp$,
$\bot_1 \neq (f^\sharp_{esc}(u)) \cap_1 (\Pi_2(id^!_\sharp))$,
and we introduce
$id_{can} \triangleq (f^\sharp_{esc}(u)) \cap_1 (\Pi_2(id^!_\sharp))$,
$idpro^! \triangleq \Pi_1(id^!_\sharp \cap_2 Inj_2(id_{can}))$.
Then we have $(f^\sharp_{pro}, f^\sharp_{can}, f^\sharp_{esc}) \rightharpoonup (g^\sharp_{pro}, g^\sharp_{can}, g^\sharp_{esc})$
where
$g^\sharp_{pro} = \{q \mapsto idpro^! \text{ if } q \in (\mathit{Agent}(Q))\}$,
$g^\sharp_{can} =$

$$\begin{cases} (q, z, z) \mapsto dpush(idpro^!) & \text{if } \begin{cases} q \in (\mathit{Agent}(Q)) \\ z \in \mathcal{BN}(Q) \cap \mathcal{FN}(q) \end{cases} \\ (q, z, t) \mapsto f^\sharp_{can}(x!^i[x_1, ..., x_n]Q, z, t) \cap_2 Inj_1(idpro^!) & \text{if } \begin{cases} q \in (\mathit{Agent}(Q)) \\ z \in \mathcal{FN}(Q) \cap \mathcal{FN}(q) \\ z \neq x \end{cases} \\ (q, x, u) \mapsto f^\sharp_{can}(x!^i[x_1, ..., x_n]Q, x, u) \cap_2 Inj_2(id_{can}) & \text{if } \begin{cases} q \in (\mathit{Agent}(Q)) \\ x \in \mathcal{FN}(Q) \cap \mathcal{FN}(q), \end{cases} \end{cases}$$

$g^\sharp_{esc} = \{t \mapsto \Pi_2(f^\sharp_{can}(Q, x_k, t) \cap_2 Inj_1(idpro^!)),\ \forall k \in [|1; n|]\}$.

Fig. 8. Abstract spied communication

Let $(f^\sharp_{pro}, f^\sharp_{can}, f^\sharp_{esc})$ be an abstract configuration,
we consider $u \in \mathit{Channel}$ and $y?^i[y_1, ..., y_n]P$ a sub-process, such that
$f^\sharp_{can}(y?^i[y_1, ..., y_n]P, y, u) = id^?_\sharp$,
$\bot_1 \neq (f^\sharp_{esc}(u)) \cap_1 (\Pi_2(id^?_\sharp))$,
and we introduce
$id_{can} \triangleq (f^\sharp_{esc}(u)) \cap_1 (\Pi_2(id^?_\sharp))$,
$idpro^? \triangleq \Pi_1(id^?_\sharp \cap_2 Inj_2(id_{can}))$.
Then we have $(f^\sharp_{pro}, f^\sharp_{can}, f^\sharp_{esc}) \rightharpoonup (g^\sharp_{pro}, g^\sharp_{can}, \emptyset)$
where
$g^\sharp_{pro} = \{p \mapsto idpro^? \text{ if } p \in (\mathit{Agent}(P))\}$,
$g^\sharp_{can} =$

$$\begin{cases} (p, z, z) \mapsto dpush(idpro^?) & \text{if } \begin{cases} p \in (\mathit{Agent}(P)) \\ z \in \mathcal{BN}(P) \cap \mathcal{FN}(p) \\ z \notin \{y_i \mid i \in [|1; n|]\} \end{cases} \\ (p, z, t) \mapsto f^\sharp_{can}(y?^i[y_1, ..., y_n]Q, z, t) \cap_2 Inj_1(idpro^?) & \text{if } \begin{cases} p \in (\mathit{Agent}(P)) \\ z \in \mathcal{FN}(P) \cap \mathcal{FN}(p) \\ z \neq y \end{cases} \\ (p, y, u) \mapsto f^\sharp_{can}(y?^i[y_1, ..., y_n]P, y, u) \cap_2 Inj_2(id_{can}) & \text{if } \begin{cases} p \in (\mathit{Agent}(P)) \\ y \in \mathcal{FN}(P) \cap \mathcal{FN}(p) \end{cases} \\ (p, y_k, t) \mapsto (Inj_1(idpro^?) \cap_2 Inj_2(f^\sharp_{esc}(t))) & \text{if } \begin{cases} p \in (\mathit{Agent}(P)) \\ y_k \in \mathcal{BN}(P) \cap \mathcal{FN}(p). \end{cases} \end{cases}$$

Fig. 9. Abstract spoilt communication

Let $(f^\sharp_{pro}, f^\sharp_{can}, f^\sharp_{esc})$ be an abstract configuration,
we consider $u \in \mathit{Channel}$ and $*y?^i[y_1,...,y_n]P$ a sub-process, such that
$f^\sharp_{can}(*y?^i[y_1,...,y_n]P, y, u) = id^?_\sharp$
$\bot_1 \neq (\Pi_2(id^?_\sharp)) \cap_1 (f^\sharp_{esc}(u))$,
and we introduce
$id_{can} \triangleq (\Pi_2(id^?_\sharp)) \cap_1 (f^\sharp_{esc}(u))$,
$idpro^? \triangleq \Pi_1(id^?_\sharp \cap_2 Inj_2(id_{can}))$,
$idpro^* \triangleq push_1^{(i,cont_!)}(idpro^?, t^\sharp)$,
$id_{s,t} \triangleq Inj_1(t^\sharp) \cap_2 Inj_2(\Pi_2(f^\sharp_{can}(y?^i[y_1,...,y_n]P, s, t) \cap_2 Inj_1(idpro_?)))$.
Then we have $(f^\sharp_{pro}, f^\sharp_{can}, f^\sharp_{esc}) \rightharpoonup (g^\sharp_{pro}, g^\sharp_{can}, \emptyset)$
where $g^\sharp_{pro} = \left\{ p \mapsto idpro^* \mid p \in (\mathit{Agent}(P)) \right.$
and
$g^\sharp_{can} =$

$$\begin{cases} (p,z,z) \mapsto dpush(idpro^*) & \text{if} \begin{cases} p \in (\mathit{Agent}(P)) \\ z \in \mathcal{BN}(P) \cap \mathcal{FN}(p) \\ z \notin \{y_i \mid i \in [|1;n|]\} \end{cases} \\ (p,y_k,t) \mapsto Inj_1(idpro^*) \cap_2 Inj_2(f^\sharp_{esc}(t)) & \text{if} \begin{cases} p \in (\mathit{Agent}(P)) \\ y_k \in \mathcal{BN}(P) \cap \mathcal{FN}(p) \end{cases} \\ (p,z,t) \mapsto push_2^{(i,cont_!)}(idpro^?, id_{z,t}) & \text{if} \begin{cases} p \in (\mathit{Agent}(P)) \\ z \in \mathcal{FN}(P) \cap \mathcal{FN}(p) \\ z \neq y \end{cases} \\ (p,y,u) \mapsto push_2^{(i,cont_!)}(idpro^?, id_{y,u}) \cap_2 Inj_2(id_{can}) & \text{if} \begin{cases} p \in (\mathit{Agent}(P)) \\ y \in \mathcal{FN}(P) \cap \mathcal{FN}(p). \end{cases} \end{cases}$$

Fig. 10. Abstract spoilt resource fetching

6 Abstract Domains

Many domains can be used to instantiate the parametric domains $Id^\sharp_1$ and $Id^\sharp_2$ and their associated primitives, depending on which complexity and which level of accuracy we expect. We have explained in [13, Sect. 6.1] that an unexpensive uniform analysis can be obtained by instantiating both $Id^\sharp_1$ and $Id^\sharp_2$ with the lattice $\{\bot, \top\}$ with the following concretization functions:

$$\gamma_1 \ : \ \begin{cases} \bot \mapsto \emptyset \\ \top \mapsto Id \end{cases} \quad \text{and} \quad \gamma_2 \ : \ \begin{cases} \bot \mapsto \emptyset \\ \top \mapsto Id \times Id \end{cases} .$$

For the sake of comparison, this analysis is at least as accurate as the one of Nielson et al. [3] and takes into account unreachable code.

Non-uniform analyses [6,22] are much more expensive. We obtain an accurate analysis by using the reduced product of both a non relational domain Id^{un}_1 (resp. Id^{un}_2) and a relational domain Id^{rel}_1 (resp. Id^{rel}_2) to represent sets of

markers (resp. sets of pairs of markers). Non relational domains provide an intelligible description of markers, whereas relational domains are used in comparing processes and channels markers.

We denote by $\Sigma = (Lbl_{used} \cup \{cont_?, cont_!\})^2$ the set of useful pairs of labels. For the sake of simplicity, we first approximate every tree marker by the words of Σ^* written on the right combs of its elements, where the right comb of a tree is defined as follows:

$$\begin{cases} right\_comb(\varepsilon) = \varepsilon \\ right\_comb(N(a,b,c)) = a.right\_comb(c) \end{cases}.$$

This abstraction is motivated by the following theorem:

Theorem 3. *Let* $(C_0, unsafe_0, unused_0) \looparrowright ... \looparrowright (C_n, unsafe_n, unused_n)$, *be a non-standard computation sequence, where* $(C_0, unsafe_0, unused_0) \in C_0(\mathcal{S})$.

If there exist $i, j \in [|0, n|]$, $(p, id, E) \in C_i$ *and* $(p', id', E') \in C_j$, *such that* $right\_comb(id) = right\_comb(id')$ *then* $id = id'$.

Our non relational domain is based on the use of *regular languages* on Σ. Id_1^{un} is the set of regular languages on Σ, while Id_2^{un} is the set of pairs of languages on Σ, defined pair wise. Associated abstract primitives are not detailed due to lack of space, a full description of these primitives is given in [13, Sect. 6.3]. Since there exist infinite chains in Id_1^{un}, a computable analysis needs a widening operator ∇. A convenient choice for $L_1 \nabla L_2$ consists in the quotient of the minimal automaton $(G, \rightarrow)$ of $L_1 \cup L_2$ by the relation $\sim_n$, defined as follows, where n is a fixed integer:

- $a \sim_0 b \iff true$;
- $a \sim_{n+1} b \iff \begin{cases} \forall c \text{ such that } a \xrightarrow{\lambda} c, \exists c' \text{ such that } c \sim_n c' \text{ and } b \xrightarrow{\lambda} c' \\ \forall c \text{ such that } b \xrightarrow{\lambda} c, \exists c' \text{ such that } c \sim_n c' \text{ and } a \xrightarrow{\lambda} c' \end{cases}$.

Our relational approximation abstracts numerical relations between the number of occurrences of each label in sets of words and in sets of pairs of words. We assign to each $\lambda \in \Sigma$ two distinct variables x_λ and y_λ. We denote by $\mathcal{V}_1$ the set $\{x_\lambda \mid \lambda \in \Sigma\}$, and by $\mathcal{V}_2$ the set $\{x_\lambda \mid \lambda \in \Sigma\} \cup \{y_\lambda \mid \lambda \in \Sigma\}$. The abstract domains $\wp(\mathbb{N}^{\mathcal{V}_1})$ and $\wp(\mathbb{N}^{\mathcal{V}_2})$ are respectively related to $\wp(Id)$ and $\wp(Id \times Id)$ by two monotone maps $\gamma_{\mathcal{V}_1}$ and $\gamma_{\mathcal{V}_2}$ defined as follows:

$$\gamma_{\mathcal{V}_1}(A) = \{u \in \Sigma^* \mid \exists (n_t)_{t \in \mathcal{V}_1} \in A,\ \forall \lambda \in \Sigma, n_{x_\lambda} = |u|_\lambda\},$$
$$\gamma_{\mathcal{V}_2}(A) = \left\{ (u,v) \in \Sigma^* \times \Sigma^* \;\middle|\; \exists (n_t)_{t \in \mathcal{V}_2} \in A,\ \forall \lambda \in \Sigma, \begin{cases} n_{x_\lambda} = |u|_\lambda \\ n_{y_\lambda} = |v|_\lambda \end{cases} \right\}.$$

Many relational numerical domains have been introduced in the literature [17,11,14]. We propose to use Karr's affine relationship equality relations domain. We define Id_1^{rel} by the set of affine equality relations systems on the set of variables $\mathcal{V}_1$, while Id_2^{rel} is the set of affine equality relations systems on the set of variables $\mathcal{V}_\in$. Those domains are fully described in [17].

Example 3. Our analysis may be used in proving the confidentiality of our *ftp* server. We denote by $C^\sharp = (f^\sharp_{pro}, f^\sharp_{can}, f^\sharp_{esc})$ the result of our analysis. $f^\sharp_{can}$ is given in Fig. 11 while $f^\sharp_{esc}$ maps each channel to $\bot_1$. These results are precise enough to prove that channels *data* (resp. *email*) can be passed only to channels *info* (resp. *add*), and cannot be listened to by any processes of an unknown context. Furthermore, using theorem 3, we can conclude that anytime a process 4 is spawned, its free channels are bound to channels created by the same recursive instance of resource 3. □

7 Conclusion

We have designed a powerful non-standard semantics, which is able to describe the behaviour of any mobile system expressed in the π-calculus, while tracing precisely the origin of channels. Moreover, our abstraction allows to analyze the behaviour of a part of a system whatever its context may be, which has many applications, such as analyzing a system part by part or analyzing the security of a known system inside a hostile context.

We have approximated this non-standard semantics into an abstract one, which focuses on confidentiality properties. We have obtained results which have at least the same level of accuracy than those obtained on *the friendly systems* [22], while our semantics works on the full π-calculus and can be applied to open mobile system. Analyzing a system part by part may lead to more accurate results than analyzing a complete system, especially while detecting mutual exclusion [13].

References

1. Martin Abadi. Secrecy by typing in security protocol. In *Proc.* 5^{th} *FPCA*, volume 523 of *Lecture Notes in Computer Science*, pages 427–447. Springer-Verlag, 1991.
2. G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.
3. C. Bodei, P. Degano, F. Nielson, and H. R. Nielson. Control flow analysis for the π-calculus. In *Proc. CONCUR'98*, number 1466 in Lecture Notes in Computer Science, pages 84–98. Springer-Verlag, 1998.
4. C. Bodei, P. Degano, F. Nielson, and H. R. Nielson. Static analysis of processes for no read-up and no write-down. In *Proc. FOSSACS'99*, number 1578 in Lecture Notes in Computer Science, pages 120–134. Springer-Verlag, 1999.
5. L. Cardelli and A. D. Gordon. Mobile ambients. In *Foundations of Software Science and Computational Structures*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer, 1998.
6. C. Colby. Analyzing the communication topology of concurrent programs. In *Symposium on Partial Evaluation and Program Manipulation*, 1995.
7. P. Cousot. Semantic foundations of program analysis. In S.S. Muchnick and N.D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 10, pages 303–342. Prentice-Hall, Inc., Englewood Cliffs, 1981.

$$
\begin{cases}
(1, \mathrm{port}, \mathrm{port}) & \mapsto \left((\varepsilon,\varepsilon), \left\{ x_\lambda = y_\lambda = 0,\ \forall \lambda \in \Sigma \quad \right)\right. \\
(2, \mathit{add}, \mathit{email}) & \mapsto \left(\begin{array}{l} ((1,4)(3,5)^*(3,6), (3,5)^*(3,6)), \\ \begin{cases} y_\lambda = x_\lambda,\ \forall \lambda \in \Sigma \setminus \{(1,4)\} \\ x_\lambda = 0,\ \forall \lambda \in \Sigma \setminus \{(1,4);(3,5);(3,6)\} \\ x_{(3,6)} = x_{(1,4)} = 1 \\ y_{(1,4)} = 0 \end{cases} \end{array} \right) \\
(2, \mathit{info}, \mathit{data}) & \mapsto \left(\begin{array}{l} ((1,4)(3,5)^*(3,6), (3,5)^*(3,6)), \\ \begin{cases} y_\lambda = x_\lambda,\ \forall \lambda \in \Sigma \setminus \{(1,4)\} \\ x_\lambda = 0,\ \forall \lambda \in \Sigma \setminus \{(1,4);(3,5);(3,6)\} \\ x_{(3,6)} = x_{(1,4)} = 1 \\ y_{(1,4)} = 0 \end{cases} \end{array} \right) \\
(3, \mathrm{port}, \mathrm{port}) & \mapsto \left((\varepsilon,\varepsilon), \left\{ x_\lambda = y_\lambda = 0,\ \forall \lambda \in \Sigma \quad \right)\right. \\
(3, \mathrm{gen}, \mathrm{gen}) & \mapsto \left((\varepsilon,\varepsilon), \left\{ x_\lambda = y_\lambda = 0,\ \forall \lambda \in \Sigma \quad \right)\right. \\
(4, \mathrm{port}, \mathrm{port}) & \mapsto \left(\begin{array}{l} ((3,5)^*(3,6), \varepsilon), \\ \begin{cases} y_\lambda = 0,\ \forall \lambda \in \Sigma \\ x_\lambda = 0,\ \forall \lambda \in \Sigma \setminus \{(3,5);(3,6)\} \\ x_{(3,6)} = 1 \end{cases} \end{array} \right) \\
(4, \mathit{email}, \mathit{email}) & \mapsto \left(\begin{array}{l} ((3,5)^*(3,6), (3,5)^*(3,6)), \\ \begin{cases} y_\lambda = x_\lambda,\ \forall \lambda \in \Sigma \\ x_\lambda = 0,\ \forall \lambda \in \Sigma \setminus \{(3,5);(3,6)\} \\ x_{(3,6)} = 1 \end{cases} \end{array} \right) \\
(4, \mathit{data}, \mathit{data}) & \mapsto \left(\begin{array}{l} ((3,5)^*(3,6), (3,5)^*(3,6)), \\ \begin{cases} y_\lambda = x_\lambda,\ \forall \lambda \in \Sigma \\ x_\lambda = 0,\ \forall \lambda \in \Sigma \setminus \{(3,5);(3,6)\} \\ x_{(3,6)} = 1 \end{cases} \end{array} \right) \\
(5, \mathrm{gen}, \mathrm{gen}) & \mapsto \left(\begin{array}{l} (3,5)^*(3,6), \varepsilon), \\ \begin{cases} y_\lambda = 0,\ \forall \lambda \in \Sigma) \\ x_\lambda = 0,\ \forall \lambda \in \Sigma \setminus \{(3,5);(3,6)\} \\ x_{(3,6)} = 1 \end{cases} \end{array} \right) \\
(6, \mathrm{gen}, \mathrm{gen}) & \mapsto \left((\varepsilon,\varepsilon), \left\{ x_\lambda = y_\lambda = 0,\ \forall \lambda \in \Sigma \quad \right)\right.
\end{cases}
$$

Fig. 11. $f^\sharp_{can}$: the *ftp* server analysis

8. 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 Fourth ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, U.S.A., 1977.
9. P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of logic and computation*, 2(4):511–547, August 1992.
10. P. Cousot and R. Cousot. Comparing the Galois connection and widening--narrowing approaches to abstract interpretation. In *Programming Language Implementation and Logic Programming, Proceedings of the Fourth International Symposium, PLILP'92*, volume 631 of *Lecture Notes in Computer Science*, pages 269–295. Springer-Verlag, 1992.
11. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the Fifth Conference on Principles of Programming Languages*. ACM Press, 1978.
12. A. Deutsch. A storeless model of aliasing and its abstraction using finite representations of right-regular equivalence relations. In *Proceedings of the 1992 International Conference on Computer Languages*, pages 2–13. IEEE Computer Society Press, Los Alamitos, California, U.S.A., 1992.
13. J. Feret. Conception de π-*sa* : un analyseur statique générique pour le π-calcul. Mémoire de dea, SPP, september 1999. Electronically available at http://www.di.ens.fr/ feret/dea.html.
14. P. Granger. Static analysis of linear congruence equalities among variables of a program. In *TAPSOFT'91*, volume 493. Lecture Notes in Computer Science, 1991.
15. Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. In U. Nestmann and B. Pierce, editors, *3rd International Workshop on High-Level Concurrent Languages (HLCL'98)*, volume 16(3) of *Electronic Notes in Theoretical Computer Science*, Nice, September 1998. Elsevier. Available from `http://www.elsevier.nl/locate/entcs`. Full version available as Sussex CSTR 98/02, 1998. Available from `http://www.cogs.susx.ac.uk/`.
16. K. Honda, V. Vasconcelos, and N. Yoshida. Secure information flow as types process behaviour. In *Proc. ESOP'00*, number 1782 in Lecture Notes in Computer Science. Springer-Verlag, 2000.
17. M. Karr. Affine relationships among variables of a program. *Acta Informatica*, pages 133–151, 1976.
18. R. Milner. The polyadic π-calculus: a tutorial. In *Proceedings of the International Summer School on Logic and Algebra of Specification*. Springer Verlag, 1991.
19. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100:1–77, 1992.
20. James Riely and Matthew Hennessy. Secure resource access for mobile agents. Draft. Available from `http://www.depaul.edu/~jriely`, June 1999.
21. D. N. Turner. *The Polymorphic Pi-Calculus: Theory and Implementation*. PhD thesis, Edinburgh University, 1995.
22. A. Venet. Automatic determination of communication topologies in mobile systems. In *Proceedings of the Fifth International Static Analysis Symposium SAS'98*, volume 1503 of *Lecture Notes in Computer Science*, pages 152–167. Springer-Verlag, 1998.