Abstract Interpretation of Mobile Systems [1]

Jérôme Feret

Département d'Informatique de l'École Normale Supérieure,
75230 PARIS Cedex 5, FRANCE

Abstract

We propose an Abstract Interpretation-based context-free analysis for mobile systems written in the π-calculus. Our analysis automatically captures a sound – but not complete – description of the potential behaviour of a mobile system interacting with an unknown context. It focuses on both the control flow and the occurrence number of agents during computation sequences.

Control flow analysis detects all the possible interactions between the agents of the system, but also the potential interactions with the context. In order to deal with dynamic creation of both names and agents which is an inherent feature of mobility, our analysis distinguishes between recursive instances of the same agent. This way, we are able to prove the integrity of an *ftp*-server used by an unbounded number of clients. Occurrence counting analysis just consists in abstracting the occurrence number of syntactic agents. Our abstraction is relational; this makes us able to detect both quickly and precisely mutual exclusion and non-exhaustion of resources.

Key words: mobility, π-calculus, static analysis, abstract interpretation, control flow analysis, occurrence counting analysis, worst case analysis.

1 Introduction

The development of large scale communicating distributed systems imposes the design of well-fitted methods for analyzing mobile systems. In such systems, process distribution dynamically changes during their computation, making their analysis a very difficult task. Furthermore, the size of systems, such as the Internet for instance, is large enough to prevent a single person

[1] This work was supported by the RTD project IST-1999-20527 "DAEDALUS" of the European FP5 programme.

from knowing the whole system. This is why we are interested in validating properties on a mobile system which belong to a bigger one, called its context, without having precise knowledge of this context.

We propose a fully automatic Abstract Interpretation-based analysis for detecting and proving some properties of such a mobile system. We will focus on the properties which rely on the control flow and the occurrence number of agents. We would like to prove the integrity of a system by detecting any information flow, even between recursive instances of the same agent. We would also like to prove non-exhaustion; this means we would like to get a bound on the number of agent instances in the system.

We propose to restrict our study to mobile systems described in the π-calculus [23,22] which is a formalism well-fitted for understanding the problems related to mobility. The π-calculus describes systems of agents which may interact via communicating channel names, through channels. When receiving a channel name, an agent gets some control over this name, and it can then communicate with other agents sharing this name. The expressive power of the π-calculus also follows from a replication mechanism, which allows to spawn an arbitrary number of instances of a same agent. Thus, each agent opens its own channels and can then communicate them to other agents. It uses α-conversion in order to deal with an arbitrary amount of channels. But α-conversion breaks any link between agent instances and the channels they have opened, which prevent us from designing an accurate control flow analysis.

So we first refine the semantics of the π-calculus in order to explicitly restore this link. In the style of Venet's work [31], we first infer an unambiguous marker to each instance of agent, and then tag each channel name with the marker of the agent which has opened the corresponding channel. This way, we can express explicitly the fact that two channels have been opened by the same recursive instance of an agent, or the fact that a name has been communicated to the agent instance which has declared it. We then extend our non-standard semantics in order to describe a system part behaviour, without any knowledge about its context. We just abstract this context by the set of the names it shares with the system part we describe. This may have many applications, as it may be used in describing the behaviour of a system within a hostile context, or in focusing on a small module within a wide system.

We then abstract this non-standard semantics in order to get a decidable approximation of it. We focus on properties relying on control flow and agent occurrence number. Control flow analysis consists in detecting for each instance of agents where the names of the channels they have opened can be communicated. This analysis is context-free: it will detect which channel names can be communicated to the context. It is also non-uniform in the sense that it distinguishes between recursive instances of agents. It can prove, for instance, that a name can be communicated to only one other instance of an agent, and not to the other ones. In the case of the *ftp*-server, it detects that the server can return a query only to the good client in the case where clients are created recursively. Our analysis consists in inferring the comparisons between name markers and the markers of the agents they are communicated to. Occurrence counting analysis consists in abstracting the occurrence number of syntactic agents during computation sequences. We propose a well-adapted domain to detect quickly mutual exclusion and non-exhaustion of resources. Our approach relies on the use of a reduced product between a non-relational and a relational domain. Complexity problems are solved by using approximated algorithms for calculating a reduction between these two domains.

As a result we get a polynomial analysis which has been implemented in OCAML. The corresponding prototype can be used on the web at:

`http://www.di.ens.fr/~feret/prototypes/prototypes.html.en`

Some examples and a short tutorial are also provided.

Related work are discussed in Sect. 2. Standard semantics of the π-calculus is given in Sect. 3. It is first refined in Sect. 4 and extended to open systems in Sect. 5. A generic abstract analysis is designed in Sect. 6. It is instantiated in both Sect. 7 and Sect. 8 to get, respectively, a control flow and an occurrence counting analysis. Complexity result and analysis time are given in Sect. 9.

2 Related work

Flow analysis. Control flow analyzes focus on the explicit flow information. Nielson and al. use abstract interpretation in [2,3,1] to infer a uniform description of the interactions between agents and apply Seidl's solver to get a cubic implementation of their analysis in [25]. Hennessy and Riely have designed a type-based analysis with the same expressive power in [18]. These analyses use explicit information flow to detect whether some security constraints specified using a security level cannot be violated. Nevertheless, these analyzes are uniform (or *mono-variant*). They cannot distinguish between distinct instances of the same resource. For example, it is impossible to give a distinct security levels to distinct instances of the same agent.

Cardelli and al. uses group creation in [4] to assign dependent security levels to channel names. A group can be associated with each recursive instance of a resource, and can then be used to prevent names created by other instances from being communicated to this instance. Nevertheless, our analysis is much more expressive: we can infer algebraic comparisons between resource instances, this allows us to express the fact that an instance of a resource can only communicate with the next instance of it or with the previous one. As a consequence, we can prove that a name is sent back to the instance which has previously declared it, even if this name is not confined into the scope of this recursive instance.

Venet has already proposed a non-uniform analysis in [30,31]. This analysis infers a sound non-uniform description of the topology of communications between the agents of *friendly systems* [22], in which resources cannot be nested. In our work, we consider the full π-calculus with nested replications. Moreover, we can analyze any open system, by approximating the interactions between this system and its context.

Occurrence counting analysis. Only very few analyses for counting occurrences of agents have been published. Nevertheless, this problem is very close to the one of approximating the behaviour of a Petri net, and of occurrence counting in mobile ambients. In [17], Nielson and al. propose an exponential analysis for counting occurrences of agents inside ambients. In [26], they use context-dependent counts for inferring a more accurate description of the internal structure of agents, at the expense of a higher time complexity (an exponential number of agents are distinguished). These analyses rely on the use of a non-relational domain to abstract an ambient content. Then, they use disjunctive completion, and abstract the set of all the potential contents of a syntactic ambient in the power set of this abstract domain. These two analyses encounter the same problem: when an agent occurs several times, they cannot decide whether this agent is still occurring one or several times after being computed, so they have to consider the two possible cases, which leads to both a loss of precision and an exponential explosion in complexity. The use of both a relational domain for globally abstracting sets of multi-sets of agents, and an approximated reduction allows us to solve this problem efficiently. We obtain a very accurate analysis which is polynomial in the number of syntactic agents (i.e. polynomial in the size of the initial system configuration).

Behavioral types. In [19], Kobayashi and Igarashi use CCS processes as types for mobile systems and check some behavioral invariants expressed in modal logic. Nevertheless, describing causality between actions leads to an explosion of the size of the types. Another problem is that their type system cannot express properties which deals with dynamic creation of channel names. Rejamani and Rehof have extended this type system in [28], so that it handles dynamic name creation. But type checking may be undecidable. So, they will have to propose an approximation in future work.

Modular analysis. Context-free semantics is an important issue in static analysis. It allows to analyze only a part of a system, without much knowledge of its context. It can be used to abstract the behaviour of an instance of a resource, and detect which names may escape the scope of this part. This can be used to detect some dead-code, for instance. Rajamani and Rehof propose a modular analysis in [28]. Having abstracting the behaviour of two modules, they can calculate an approximation of the parallel composition of them. But this analysis is very restrictive because module types must satisfy some assume-guarantee properties.

3 π-calculus

We introduce in this section the π-calculus and a standard semantics of it. The π-calculus is a formalism used to describe mobile systems. It describes a system as a set of agents which exchange information over channels. These communications enable agent synchronization, but also dynamical modification of the system topology: agents can open new channels, agents can pass control over some channels to other agents and some agents can be dynamically created. Here, we consider a lazy version of the synchronous polyadic π-calculus [22] with internal choice operator. In polyadic π-calculus, agents can communicate tuples of channel names. We use the lazy version of the replication introduced in [29, Chap. 7], in which resources are duplicated only when they are required by a communication.

Let $\mathcal{N}$ be a countable set of channel names and $\mathcal{L}$ a countable set of labels. The syntax of agents is described in Fig. 1(a). We have located syntactic components with labeling actions using distinct elements of $\mathcal{L}$. Input guard, replication guard and name restriction act as name binders, i.e in the agents $c?^j[x_1, ..., x_n]P$, $*d?^j[y_1, ..., y_p]Q$ and $(\nu\ x)R$, the occurrences of x_1, ..., x_n in P, y_1, ..., y_p in Q and x in R are supposed to be bound. We also assume that no one name occurs twice in a whole system, as an argument of an input guard, a replication guard or a name restriction. Usual rules about scope, substitution and α-conversion apply. We denote by $fn(P)$ the set of free names in P, i.e names which are not under the scope of a binder, and by $bn(P)$ the set of bound names in P.

We now informally introduce the semantics of the π-calculus. The agent aP first computes the action a before launching the continuation P. The agent $(\nu\ x)P$ declares a new channel, named x, which can be used and communicated by the agent P. In $(P \mid Q)$, P and Q are two concurrent agents which may behave independently, or interact by communicating. The formula $(P \oplus Q)$ denotes an internal choice between two agents. Either P or Q is run, while the other fades away; the choice between P and Q does not depend on the other agents. The agent $\mathbf{0}$ does nothing. The agent $c![x_1, ..., x_n]P$ sends a message via the channel named c, this message is the tuple of chan-

$$
\begin{array}{rcll}
P & ::= & aP & \text{(action)} \\
 & | & (\nu\ x)P & \text{(name restriction)} \\
 & | & (P \mid P) & \text{(parallel composition)} \\
 & | & (P \oplus P) & \text{(internal choice)} \\
 & | & \mathbf{0} & \text{(nil)} \\
a & ::= & c!^{j}[x_1, ..., x_n] & \text{(output)} \\
 & | & c?^{i}[x_1, ..., x_n] & \text{(input guard)} \\
 & | & *c?^{i}[x_1, ..., x_n] & \text{(replication guard)}
\end{array}
$$

where $c, x_1, ..., x_n, x \in \mathcal{N},\ i, j \in \mathcal{L}$ *and* $n \geqslant 0$.

(a) Syntax

$$
\begin{array}{rcll}
(\nu\ x)P & \equiv & (\nu\ y)P[x \leftarrow y] \quad \text{if } y \notin fn(P) & (\alpha\text{-conversion}) \\
P \mid Q & \equiv & Q \mid P & \text{(commutativity)} \\
P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R & \text{(associativity)} \\
P \mid \mathbf{0} & \equiv & P & \text{(end of an agent)} \\
(\nu\ x)\mathbf{0} & \equiv & \mathbf{0} & \text{(garbage collecting)} \\
(\nu\ x)(\nu\ y)P & \equiv & (\nu\ y)(\nu\ x)P & \text{(swapping)} \\
((\nu\ x)P) \mid Q & \equiv & (\nu\ x)(P \mid Q) \quad \text{if } x \notin fn(Q) & \text{(extrusion)}
\end{array}
$$

where $x,\ y \in \mathcal{N}$

(b) Congruence relation

$$
\begin{array}{rl}
c!^{j}[x_1, ..., x_n]P \mid c?^{i}[y_1, ..., y_n]Q \xrightarrow{(i,j)} P \mid \widetilde{Q} & \text{(com...)} \\
c!^{j}[x_1, ..., x_n]P \mid *c?^{i}[y_1, ..., y_n]Q \xrightarrow{(i,j)} P \mid \widetilde{Q} \mid *c?[y_1, ..., y_n]Q & \text{(res. fetching)} \\
P \oplus Q \xrightarrow{\oplus} P & \text{(left choice)} \\
P \oplus Q \xrightarrow{\oplus} Q & \text{(right choice)}
\end{array}
$$

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

where $c,\ x,\ x_1,\ ...,\ x_n,\ y_1,\ ...,\ y_n \in \mathit{Channel},\ i, j \in \mathcal{L}$,
and $\widetilde{Q} = Q[y_1 \leftarrow x_1, ..., y_n \leftarrow x_n]$

(c) Reduction relation

Fig. 1. The chemical semantics

nel names $(x_1, ..., x_n)$. The agent $c?[y_1, ..., y_n]P$ waits for a message on the channel named c, and binds the names y_1,...,y_n to the received names. The agent $*c?[y_1, ..., y_n]P$ is a resource. It duplicates itself just before receiving a message: a new instance of P is launched with y_1,...,y_n bound to the receive channel names while $*c?[y_1, ..., y_n]P$ waits for the next message.

The operational semantics is given by both a congruence relation in Fig. 1(b) and a reduction relation in Fig. 1(c). The congruence relation allows redex revelation, while the reduction relation describes agent computations. Some rules in the congruence relation make agents move: they assert the associativity and commutativity of the parallel composition. Some others extend the scope of names to the agents they are communicated to: α-conversion solves conflicts between names, swapping selects the name of which we want to extend the scope, and extrusion extends its scope to another agent. The reduction relation describes agents communications. A communication is allowed when there are two concurrent processes, such that the first one sends a message on a channel, while the second one waits for a message on the same channel (we also request that both messages have the same arity). The results of such a communication is obtained by applying the substition of the λ-calculus in the continuation of the message receiver. In the case when the receiver is a resource, it is just syntactily duplicated before making the communication; this way the resource is still available after the communication.

We now propose some examples to illustrate both this semantics and the kind of the properties we are interested in. We will find out that the semantics we have considered is not precise enough to handle the properties we are interested in.

Example 1 *An* ftp-*server can be described by the system given in Fig. 2.*

*The first resource creates repeatedly a new client which send a query to a server. This query is composed by a query **request**, and an address **address**. The client sends its query again in case it receives a failure report denoted by the agent **address![]**. The second resource describes the server. When this one receives a query, it duplicates itself. Then, either it uses an available port and computes the query or it reports a failure to the client by spawning the agent **address![]**. Available ports are denoted by agents **port![]**. Data processing just consists in a communication between two agents of the server; most computational features are abstracted away. After this communication, the port is released, while the answer is sent back to the client.*

*Our analysis will prove both the integrity and the non-exhaustion of this system: it will discover that each time an agent **email![rep]** is spawned, the names **email** and **rep** are respectively linked to two channels opened by the restrictions (ν **address**) and (ν **request**) of the same instance of a resource,*

```
((ν make)(ν server)(ν port)
((*make?[](ν address)(ν request)
          (
           (*address?[]server![address, request])
          |
           address![]
          |
           make![]
          ))
|
 (*server?[email, data](ν deal)
          (
           port?[](deal![data] | deal?[rep](email![rep] | port![]))
          ⊕
           email![]
          ))
| port![]
| port![]
| port![]
| make![]))
)
```

Fig. 2. An *ftp*-server

```
((ν make)(ν mon)(ν left0)
(
 ( (*make?[left](ν right)(mon![left, right] | make![right]))
 | (*make?[left](mon![left, left0]))
 | make![left0])
|
 ((*mon?[prev, next]
          (*prev?[](ν crit)(crit?[]next![]
                           | crit![])))
 | left0![])))
```

Fig. 3. A ring of processes

and so the server returns its computed answer to the good client; it captures the fact that no more than three instances of the syntactic agent $\boldsymbol{deal}![\boldsymbol{data}]$ *can occur simultaneously, which means that no more than three simultaneous sessions can be active in the same time.*

Example 2 *We propose in Fig. 3 a mobile system which creates a ring of processes, with a token passed around this ring. Channel names created by name restrictions* $(\nu\ \mathbf{left0})$ *and* $(\nu\ \boldsymbol{right})$ *denote the processes of the ring. The first part of the system describes the ring creation. The first process is created by the restriction* $(\nu\ \mathbf{left0})$*. An agent* $\mathbf{mon}![\boldsymbol{v1};\boldsymbol{v2}]$ *denotes a connection between two processes. Then, each time the first resource is replicated, a new process is created and linked to the previous process, which has been passed as an argument of the replication. The second resource replication closes the ring by linking the last created process to the first created process.*

The second part of the system describes the execution of the processes: an additional resource spawns a resource for each process of the ring. Then a token is put in the ring of processes: the token is denoted by syntactic copies of the agents $\boldsymbol{next}![]$ *and* $\mathbf{left0}![]$*. The name of this agent describes the token location. When a token is available, the corresponding process can duplicate its resource, and as a result the process enters its critical section. The critical section is exited when the two agents* $\boldsymbol{crit}![]$ *and* $\boldsymbol{crit}?[]$ *have interacted; the token is then passed to the next process.*

Our analysis can prove both the integrity and the non-exhaustion of this system: it discovers that each time an agent ***mon![left;right]*** *is spawned, either the name* ***left*** *is linked to the channel opened by the restriction* $(\nu\ \mathbf{left0})$*, or both names* ***left*** *and* ***right*** *are linked to two channels opened by instances of the restriction* $(\nu\ \boldsymbol{right})$*, but the channel linked to the name* ***left*** *has been created by the previous instance of it, which means that a process can only be connected to either the first one or to the next one; it captures the fact that only one simultaneous instance of the syntactic agent* $\boldsymbol{crit}![]$ *can exist. That is to say, that only one process can enter its critical session at a given time.*

Remark 3 *The* α*-conversion prevents from expressing integrity propesties, because it destroyes the link between agent instances and the names of the channel they have opened. If we think about the example of the* ftp*-server. If we cleverly choose the name of created channel by indexing them with the replication number of the client resources. we obtain after two sessions of the server a system of the following form:*

$$(\nu\ \overline{\mathbf{c}})(\nu\ address_1)(\nu\ request_1)(\nu\ address_2)(\nu\ request_2)$$
$$(\mathcal{S}' \mid address_1![request_1] \mid address_2![request_2]).$$

It appears explicitly that request answers are returned at good addresses. But, we can transform by using the α*-conversion the previous system in the follow-*

ing one:

$$(\nu\ \overline{\mathbf{c}})(\nu\ address_1)(\nu\ request_1)(\nu\ address_2)(\nu\ request_2])$$
$$(\mathcal{S}' \mid address_2![request_1] \mid address_1![request_2]).$$

in where this property is lost. So we have to design another semantics in where channel name origin is carefully traced.

4 Refining the semantics

The non-standard semantics is a refined semantics which aims at explicitly specifying the links between the channels and the instances of agents which have declared them. Any instance of an agent is identified unambiguously by a marker in order to distinguish that instance from all others. Tracing the origin of channel names can then easily be done by tagging each new channel name with the marker of the agent which has created it. Venet, in [31], has designed such a non-standard semantics, but it applies only to a small part of the π-calculus, called *the friendly systems* [22]. It especially requires resources not to be nested, and the system to be closed. We propose here a new non-standard semantics in order to remove those restrictions.

This section will be organized as follows: we first describe our marker allocation scheme, we then propose a naive fully operational semantics for describing the behaviour of closed mobile systems, and we finally improve this semantics in order to reduce the number of computation steps. Open systems are dealt in the next section.

4.1 Fresh name allocation

As explained before, α-conversion prevents from expressing the link between recursive instances and the names they have declared. So we proposes a name allocation scheme to ensure the freshness of allocated names. Such scheme has already been proposed in [12]. Nevertheless, our requirement is quite different. De Bruijn's naming scheme allows avoiding α-conversion in order to make some hand-proofs more easily. We also expect our scheme to allow us to express some integrity properties. For instance, we would like to express in this semantics the fact that two names may have been created by the same instance of a given agent, which is crucial to express some integrity properties. Furthermore, as we want to make static analyses, we expect the scheme to help invariant formation. For that purpose, we want the scheme not to depend on the order in which commutative computation steps are performed.

We denote by $\mathcal{M}$ 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 $\mathcal{L}$. The tree having a node labeled a, a left sibling t_1 and a right one t_2 is denoted by $N(a, t_1, t_2)$. Markers are binary trees in $\mathcal{M}$. Initial thread markers are ε, while the marker of a new thread us calculated recursively from the marker of the threads the computation of which has lead to its creation:

- when a computation step does not involve fetching a resource, the marker of the computed agent is just passed to its continuation;
- when a resource is fetched, a new marker is deterministically allocated: it is given by $N((i, j), id_i, id_j)$ where i is the label of the resource, id_i is the marker of the resource, j is the label of the thread which enforces the resource fetching and id_j is the marker of this thread.

Marker allocation consistency is expressed by the following proposition:

Proposition 4 *During a computation sequence, a syntactic agent cannot be spawned twice with the same marker.*

Prop. 4 relies on the fact for each tagged agent contains explicitly information about an agent which has been consumed to spawn this agent. □

Moreover, in accordance to the following proposition, we can simplify the shape of the markers without losing marker allocation consistency..

Proposition 5 *Let ϕ_1 and ϕ_2 be the two following functions:*

$$\phi_1 : \begin{cases} \mathcal{M} & \rightarrow (\mathcal{L}^2)^\star \\ N(a, b, c) & \mapsto \phi_1(c).a \\ \varepsilon & \mapsto \varepsilon \end{cases} \qquad \phi_2 : \begin{cases} \mathcal{M} & \rightarrow \mathcal{L}^\star \\ N((i, j), b, c) & \mapsto \phi_2(c).j \\ \varepsilon & \mapsto \varepsilon. \end{cases}$$

Marker allocation remains consistent when replacing each marker by its image by ϕ_1 or ϕ_2.

Such simplifications allow us to reduce the cost of our analysis, but also lead to a loss of accuracy, since they merge information related to distinct computation sequences of the system.

4.2 Naive semantics

Let us first consider the case of a closed mobile system $\mathcal{S}$ in the π-calculus. The subset of $\mathcal{L}$ used in labeling $\mathcal{S}$ is denoted by $\mathcal{L}_{\text{used}}$. 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; they map each free name of syntactic components to a pair (x, id), where x is a bound name of $\mathcal{S}$ and id is a marker. Intuitively, (x, id) refers to the name of the channel opened by the instance of the restriction $(\nu\ x)$ tagged with the marker id. 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. 4.

Roughly speaking, the initial configuration contains only one thread: the system itself, tagged with the marker of initial thread, ε. Since the system is closed, its environment is empty. Structural rules mimic and orient congruence relation. A thread composed by two concurrent sub-threads can be replaced by the two corresponding threads. Channel restriction consists in creating a fresh name, and binding corresponding variable to this name in the environment of the continuation. The fresh name is obtained by tagging the name used in the restriction by the marker of the agent which has declared this name. A thread the syntactic agent of which is the empty process, can be removed. Choice rules mimic choice reduction rules. A thread the syntactic agent of which is a choice between two syntactic agent can be replaced by a thread corresponding to one of this agent. Communication rules mimic communication reduction rules. Synchronisation condition is checked in the environment of the communicating threads. Name passing are described by explicit substitution in environments. In the case when a resource is fetched, a fresh marker is infered in accordance to our marker allocation scheme.

We can define the translation $\Pi(C)$ of a non-standard configuration C as follows:

$$\Pi(C) = (\nu\ c_1)...(\nu\ c_k)(E(t_1)|\ ...\ |E(t_l))$$

where $\{c_i \mid i \in [\![1;k]\!]\} = \{E(x) \mid (P,, E) \in C,\ x \in fn(P)\}$ and $C = \{t_i \mid i \in [\![1;l]\!]\}$. The system $\Pi(C)$ is well defined thanks to associativity, commutativity, and swapping rules, we have also abusively assumed that $\mathcal{L}\times\mathcal{M}$ was a subset of $\mathcal{L}$. We now establish the relation between the standard and the non-standard semantics as follows:

Lemma 6 *If* $\mathcal{C}_0^{\mathrm{n}}(\mathcal{S}) \xrightarrow{u}_{\mathrm{n}}^{*} C$,

then $C \xrightarrow{\varepsilon}_{\mathrm{n}} C' \implies \Pi(C) \equiv \Pi(C')$, *and* $C \xrightarrow{\lambda}_{\mathrm{n}} C' \implies \Pi(C) \xrightarrow{\lambda} \Pi(C')$.

Lemma 7 *If* $\mathcal{C}_0^{\mathrm{n}}(\mathcal{S}) \xrightarrow{u}_{\mathrm{n}}^{*} C$, *then* $\Pi(C) \xrightarrow{\lambda} P \implies \begin{cases} \exists C',\ C \xrightarrow{\lambda}_{\mathrm{n}} C' \\ C' \equiv P. \end{cases}$

Theorem 8 *The standard and the non-standard semantics are in* weak bisim-

$$\mathcal{C}_0^{\mathrm{n}}(\mathcal{S}) = (\mathcal{S}, \varepsilon, \emptyset)$$

(a) Initial configuration

$$\begin{aligned}
C \cup \{(P \mid Q, id, E)\} &\xrightarrow{\varepsilon}_{\mathrm{n}} C \cup \{(P, id, E_{|fn(P)}); (Q, id, E_{|fn(Q)})\} \\
C \cup \{((\nu\ x)P, id, E)\} &\xrightarrow{\varepsilon}_{\mathrm{n}} C \cup \{(P, id, E[x \to (x, id)]_{|fn(P)})\} \\
C \cup \{(\mathbf{0}, id, E)\} &\xrightarrow{\varepsilon}_{\mathrm{n}} C
\end{aligned}$$

(b) Structural rules

$$\begin{aligned}
C \cup \{P \oplus Q, id, E\} &\xrightarrow{\oplus}_{\mathrm{n}} C \cup \{(P, id, E_{|fn(P)})\} \\
C \cup \{P \oplus Q, id, E\} &\xrightarrow{\oplus}_{\mathrm{n}} C \cup \{(Q, id, E_{|fn(Q)})\}
\end{aligned}$$

(c) Choice rules

$$\frac{E_?(y) = E_!(x)}{C \cup \left\{ \begin{array}{l} (y?^i[\overline{y}]P, id_?, E_?); \\ (x!^j[\overline{x}]Q, id_!, E_!) \end{array} \right\} \xrightarrow{(i,j)}_{\mathrm{n}} C \cup \left\{ \begin{array}{l} (P, id_?, E_?[\overline{y} \to E_!(\overline{x})]_{|fn(P)}); \\ (Q, id_!, E_{!|fn(Q)}) \end{array} \right\}}$$

$$\frac{E_?(y) = E_!(x)}{C \cup \left\{ \begin{array}{l} (*y?^i[\overline{y}]P, id_?, E_?); \\ (x!^j[\overline{x}]Q, id_!, E_!) \end{array} \right\} \xrightarrow{(i,j)}_{\mathrm{n}} C \cup \left\{ \begin{array}{l} (*y?^i[\overline{y}]P, id_?, E_?); \\ (P, N((i,j), id_?, id_!), E_?[\overline{y} \to E_!(\overline{x})]_{|fn(P)}); \\ (Q, id_!, E_{!|fn(Q)}) \end{array} \right\}}$$

(d) Communication rules

Fig. 4. Naive semantics

ulation.

4.3 Efficient semantics

We now propose to reduce the number of reduction steps in order to make analysis design easier, and also to obtain a more efficient analysis.

4.3.1 Strongly bisimilar semantics

The binary relation $\xrightarrow{\varepsilon}_{\mathrm{n}}$ is nœtherian and locally confluent. So, following [20], it is confluent, and we can define its limit $\Downarrow$ as follows:

$$a \Downarrow b \textit{ if and only if } a \xrightarrow{\varepsilon}{}^{*}_{\mathrm{n}} b \textit{ and } \forall c,\ b \not\xrightarrow{\varepsilon}_{\mathrm{n}} c.$$

We now express explicitly this limit by designing an extraction function β^{i} which performs all structural rules in parallel. The definition of β^{i} is given in Fig. 5. We have $C \Downarrow \bigcup\limits_{t \in C} \beta^{\mathrm{i}}(t)$.

$$\begin{aligned}
\beta^{\mathrm{i}}((\nu\ n)P, id, E) &= \beta^{\mathrm{i}}\left(P, id, (E[n \mapsto (n, id)])\right) \\
\beta^{\mathrm{i}}(\mathbf{0}, id, E) &= \emptyset \\
\beta^{\mathrm{i}}(P \mid Q, id, E) &= \beta^{\mathrm{i}}(P, id, E) \cup \beta^{\mathrm{i}}(Q, id, E) \\
\beta^{\mathrm{i}}(P \ \oplus\ Q, id, E) &= \{(P \ \oplus\ Q, id, E)\} \\
\beta^{\mathrm{i}}(aP, id, E) &= \{(aP, id, E_{|fn(aP)})\}
\end{aligned}$$

Fig. 5. Extraction function

We define the intermediate semantics as the transition system $(\{\mathcal{C}_0^{\mathrm{i}}(\mathcal{S})\}, \longrightarrow_{\mathrm{i}})$:

- $\mathcal{C}_0^{\mathrm{i}}(\mathcal{S}) = \beta^{\mathrm{i}}(\mathcal{C}_0^{\mathrm{n}}(\mathcal{S}))$
- $\forall \lambda \in (\mathcal{L}^2 \cup \oplus), a \xrightarrow{\lambda}_{\mathrm{i}} b$ *if and only if* $a\ (\xrightarrow{\lambda}_{\mathrm{n}} \circ \Downarrow)\ b$

Theorem 9 *The naive and the intermediate semantics are* weakly bisimilar*.*

Corollary 10 *As a consequence, the standard and the intermediate semantics are* strongly bisimilar *(up to $\equiv$).*

4.3.2 Efficient semantics

We propose to focus on communication rules and also factor choice rules. To do this, we have to restrict the set of the traces: to make things easier, we propose to consider only the traces where communications are not performed until some choices can be made. [2]

We denotes by $\xrightarrow{\varepsilon|\oplus}_{\mathrm{n}}$ the binary relation $\xrightarrow{\varepsilon}_{\mathrm{n}} \cup \xrightarrow{\oplus}_{\mathrm{n}}$. The reduction relation $\xrightarrow{\varepsilon|\oplus}_{\mathrm{n}}$ is nœtherian, but not locally confluent. We define the relation $\Longrightarrow$ as follows:

$$a \Longrightarrow b \textit{ if and only if } a \xrightarrow{(\varepsilon|\oplus)^*}{}^{*}_{\mathrm{n}}\, b \textit{ and } \forall c,\ b \overset{\varepsilon|\oplus}{\not\to}_{\mathrm{n}} c.$$

Since $\xrightarrow{\varepsilon}_{\mathrm{n}} \cup \xrightarrow{\oplus}_{\mathrm{n}}$ is not confluent, $\Longrightarrow$ is not deterministic. The set of the successor by $\Longrightarrow$ can be explicitly defined as the set

$$\left\{ \bigcup_{t \in C} \mathrm{Cont}_t \;\middle|\; \forall t \in C, \mathrm{Cont}_t \in \beta^{\mathrm{i}}(t) \right\},$$

where β is given in Fig. 6(a). Intuively, β gives the set of all the choices when spawning a continuation. Spawning a thread the syntactic component of which is a choice $(P \oplus Q)$, we either spawn a choice for P or a choice for Q. Spawning a thread the syntactic component of which is a parallel composition $(P \mid Q)$, consists in choosing a choice for P, choosing a choice for Q, and spawning concurrently these two choices. We define the efficient semantics as the transition system $(\mathcal{C}_0^{\mathrm{e}}(\mathcal{S}), \longrightarrow_{\mathrm{e}})$ defined as follows:

- $\mathcal{C}_0^{\mathrm{e}}(\mathcal{S}) = \beta(\mathcal{S}, \varepsilon, \emptyset)$
- $\forall \lambda \in \mathcal{L}^2, a \xrightarrow{\lambda}_{\mathrm{e}} b$ *if and only if* $a\ (\xrightarrow{\lambda}_{\mathrm{n}} \circ \Longrightarrow)\ b$.

The explicit definition of $(\mathcal{C}_0^{\mathrm{e}}(\mathcal{S}), \longrightarrow_{\mathrm{e}})$ is given in Fig. 6.

5 Context-free semantics

5.1 Construction

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 agents, concurrently running with $\mathcal{S}$. We represent this context by the set of names it shares with $\mathcal{S}$, called *unsafe names*, and approximate its behaviour as if it was an intruder who was able to compose any possible agent working on these channel names. An interaction between $\mathcal{S}$ and its context may only consist in a communication between an agent $p_{\mathcal{S}}$

[2] We have also considered in [13, Sect. 3.3] to restrict the set of the traces to those for which choices are made only when necessary, which comes down to consider external choices instead of internal ones.

$$\begin{aligned}
\beta((\nu\ x)P, id, E) &= \beta\left(P, id, (E[x \mapsto (x, id)])\right) \\
\beta(\mathbf{0}, id, E) &= \{\emptyset\} \\
\beta(P\ \oplus\ Q, id, E) &= \beta(P, id, E) \cup \beta(Q, id, E) \\
\beta(P \mid Q, id, E) &= \{A \cup B \mid A \in \beta(P, id, E),\ B \in \beta(Q, id, E)\} \\
\beta(aP, id, E) &= \{\{(aP, id, E_{|fn(aP)})\}\}
\end{aligned}$$

(a) Extraction function

$$C_0(\mathcal{S}) = \beta(\mathcal{S}, \varepsilon, \emptyset)$$

(b) Initial configurations

$$\frac{\begin{cases} E_?(y) = E_!(x), \\ Ct_P \in \beta(P, id_?, E_?[y_i \mapsto E_!(x_i)]), \\ Ct_Q \in \beta(Q, id_!, E_!) \end{cases}}{C \cup \left\{ \begin{array}{l} (y?^i[\overline{y}]P, id_?, E_?), \\ (x!^j[\overline{x}]Q, id_!, E_!) \end{array} \right\} \xrightarrow{(i,j)}_{\mathrm{e}} (C \cup Ct_P \cup Ct_Q)}$$

$$\frac{\begin{cases} E_*(y) = E_!(x), \\ Ct_P \in \beta(P, N((i,j), id_*, id_!), E_*[y_i \mapsto E_!(x_i)]), \\ Ct_Q \in \beta(Q, id_!, E_!) \end{cases}}{C \cup \left\{ \begin{array}{l} (*y?^i[\overline{y}]P, id_*, E_*), \\ (x!^j[\overline{x}]Q, id_!, E_!) \end{array} \right\} \xrightarrow{(i,j)}_{\mathrm{e}} \Big(C \cup \{(*y?^i[\overline{y}]P, id_*, E_*)\} \cup Ct_P \cup Ct_Q\Big)}$$

(c) Communication rules

Fig. 6. Efficient semantics

of the first one and an agent 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. When spying, the context listens to new channel names which get unsafe. When 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; as a consequence, we have to introduce an infinite set of unsafe names that the context may have created. Eventually, 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 that those channels have been declared by recursive instances of a single process. By choosing $cont_?, cont_! \in \mathcal{L} \setminus \mathcal{L}_{\text{used}}$ and $ext \in \mathcal{N} \setminus bn(\mathcal{S})$, such channels will be seen as if they had been created by the restriction $(\nu\ ext)$ of a recursive instance of a process the marker of which is t_n, where t_n is recursively defined as follows:

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

We then denote by *en* the set $\{(ext, t_n) \mid n \in \mathbb{N}\}$ and assume that all spoiling message are syntactic copies of a single agent, whose first action is labeled with $0 \in \mathcal{L} \setminus (\mathcal{L}_{\text{used}} \cup \{cont_?; cont_!\})$. The coherence of our semantics mainly relies on the fact that during a computation sequence, there cannot be two different instances of a single agent 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 triple (C, U, F), where C is a set of threads, U is a set of pairs (a,id), such that channel names created by the restriction $(\nu\ a)$ of the recursive instance of an agent the marker of which was id is unsafe, and F is a set of fresh markers which have not been used as markers for spoiling messages. The system $\mathcal{S}$ may start with several initial configurations, since free names have to be chosen among the set of initial unsafe names. This assumption is especially useful to analyze an instance of a resource without any knowledge of the relations between channel names having been communicated to it.

The transition relation $\looparrowright$ takes into account the computations inside the mobile system $\mathcal{S}$, as well as the computations involving the system $\mathcal{S}$ and its context. Initial non-standard configurations and computation rules are given in Fig. 7.

$$\mathcal{C}_0^{\mathrm{o}}(\mathcal{S}) = \left\{ (Ct, en, \{t_n \mid n \in \mathbb{N}\}) \;\middle|\; \begin{array}{l} Ct \in \beta(\mathcal{S}, \varepsilon, E), \\ E \in \mathcal{F}(fn(\mathcal{S}) \to un(\mathcal{S})), \\ \forall x \in fn(\mathcal{S}),\ \exists n \in \mathbb{N},\ E(x) = (ext, t_n) \end{array} \right\}$$

(a) Initial configurations

$$\frac{C \xrightarrow{\lambda}_{\mathrm{e}} C'}{(C, U, F) \overset{\lambda}{\hookrightarrow} (C', U, F)}$$

(b) Internal interactions

$$\frac{t = (x!^j[x_1, ...x_n]P, id, E),\ E(x) \in U,\ Ct \in \beta(P, id, E)}{(C \cup \{t\}, U, F) \overset{(0,j)}{\hookrightarrow} (C \cup Ct, U \cup \{E(x_k) \mid k \in [\![1; n]\!]\}, F)}$$

(c) Spied communication

$$\frac{\begin{cases} t = (y?^i[y_1, ...y_n]P, id, E), \\ E(y) \in U,\ c_1, ..., c_n \in U, \\ Ct \in \beta(P, id, E[y_k \mapsto c_k]) \end{cases}}{(C \cup \{t\}, U, F) \overset{(i,0)}{\hookrightarrow} (C \cup Ct, U, F)}$$

(d) Spoiled communication

$$\frac{\begin{cases} t = (*y?^i[y_1, ...y_n]P, id, E), \\ E(y), c_1, ..., c_n \in U, \\ id_! \in F, \\ id_* = N((i, 0), id, id_!) \\ Ct \in \beta(P, id_*, E[y_k \mapsto c_k]) \end{cases}}{(C \cup \{t\}, U, F) \overset{(0,j)}{\hookrightarrow} (C \cup \{t\} \cup Ct, U, F \setminus \{id_!\})}$$

(e) Spoiled resource fetching

Fig. 7. Context-free non-standard semantics

We propose to establish the relation between the non-standard semantics of closed and open systems. Let $\mathcal{S}_{\mathrm{I}}(x_1, ..., x_n)$ be an open system the set of free names of which is exactly the set $\{x_i \mid i \in [\![1; n]\!]\}$. We want to construct a projection function Π, such that any non-standard computation sequence τ of a closed system of the form $(\nu\ c_1)...(\nu\ c_k)(\mathcal{S}_{\mathrm{I}}(c_{i_1}, ..., c_{i_n}) \mid \mathcal{S}_{\mathrm{c}}(c_{j_1}, ...c_{j_l}))$, is mapped to a non-standard computation sequence of the open system $\mathcal{S}_I$, and also such that, for any non-standard computation sequences τ' of the open system $\mathcal{S}_{\mathrm{I}}$, there exists a closed system $(\nu\ c_1)...(\nu\ c_k)(\mathcal{S}_{\mathrm{I}}(c_{i_1}, ..., c_{i_n}) \mid \mathcal{S}_{\mathrm{c}}(c_{j_1}, ...c_{j_l}))$ and a computation sequence τ of it, with $\tau' = \Pi(\tau)$.

Let $\mathcal{L}_{\mathrm{I}} \subseteq \mathcal{L}$ be the subset of the labels occurring in $\mathcal{S}_{\mathrm{I}}$ and $\mathcal{N}_{\mathrm{I}} \subseteq \mathcal{N}$ be the subset of the names occurring in name restriction off $\mathcal{S}_{\mathrm{I}}$. We introduce two one-to-one functions in order to interpret names and agents created by the context of $\mathcal{S}_{\mathrm{I}}$: let $\Phi_{\mathcal{M}}$ be a one-to-one map from the set $(\mathcal{L} \times \mathcal{M})$ into the set $\mathbb{N}$, and Φ_{chan} be a one-to-one function from the set $(\mathcal{N} \times \mathcal{M})$ into the set $\mathbb{N}$. We now define the projection $\Pi_\tau(\mathcal{S}_{\mathrm{I}}, \Phi_{\mathcal{M}}, \Phi_{\mathrm{chan}})$ which transforms each computation sequence of a closed mobile system $\mathcal{S} = (\nu\ \overline{c})(\mathcal{S}_{\mathrm{I}}(c_{i_1}, ..., c_{i_n}) \mid \mathcal{S}_{\mathrm{c}}(c_{j_1}, ...c_{j_l}))$ into a computation sequence of the part $\mathcal{S}_{\mathrm{I}}$ of the system $\mathcal{S}$.

We first project each syntactic component label, by replacing each label which does not occur in $\mathcal{S}_{\mathrm{I}}$ by the unique label of the context, that is to say 0.

Definition 11 *The projection $\Pi_{\mathrm{l}}(l)$ of a syntactic component label $l \in \mathcal{L}$ is defined as follows:*

$$\Pi_{\mathrm{l}}(l) = \begin{cases} l & \textit{if } l \in \mathcal{L}_{\mathrm{I}} \\ 0 & \textit{otherwise} \end{cases}$$

We then apply pair-wisely the syntactic component label projection on transition labels.

Definition 12 *The projection $\Pi_\lambda((i,j))$ of transition label $(i,j) \in \mathcal{L}^2$ is defined as follows:*

$$\Pi_\lambda((i,j)) = (\Pi_{\mathrm{l}}(i), \Pi_{\mathrm{l}}(j))$$

Next, we project the instance markers of the syntactic components of $\mathcal{S}_{\mathrm{I}}$. In such a marker, only the right sibling can be the marker of a syntactic component of the context, as we know that the resource fetched belongs to $\mathcal{S}_{\mathrm{I}}$. When a resource is fetched by a message of the context, we replace its syntactic component, and calculate a coherent marker according to $\Phi_{\mathcal{M}}$.

Definition 13 *Marker projection is defined as follows:*

$$\Pi_{\mathcal{M}} : \begin{cases} N((i,j),t_1,t_2) \mapsto N(\Pi_\lambda((i,j)),\Pi_{\mathcal{M}}(t_1),\Pi_{\mathcal{M}}(t_2)) & \textit{if } j \in \mathcal{L}_{\mathrm{I}} \\ N((i,j),t_1,t_2) \mapsto N(\Pi_\lambda((i,j)),\Pi_{\mathcal{M}}(t_1),\Phi_{\mathcal{M}}(j,t_2))) & \textit{if } j \notin \mathcal{L}_{\mathrm{I}} \\ \qquad\varepsilon \qquad\quad \mapsto \varepsilon & \end{cases}$$

We now project channel names. For a channel having been opened by a name restriction of $\mathcal{S}_{\mathrm{I}}$, we just project the marker. For those having been opened by the context, we replace the name restriction by the unique restriction (ν *ext*) of the context, and calculate the coherent marker according to Φ_{chan}.

Definition 14 *Channel name projection is defined as follows:*

$$\Pi_{\mathcal{N}}((x,id_x)) = \begin{cases} (x,\Pi_{\mathcal{M}}(id_x)) & \textit{if } x \in \mathcal{N}_{\mathrm{I}} \\ (ext,\Phi_{\mathrm{chan}}((x,id_x))) & \textit{otherwise} \end{cases}$$

We can easily project an instance of a syntactic component of $\mathcal{S}_{\mathrm{I}}$ by projecting its marker, and each channel inside its environment.

Definition 15 *Thread projection is defined as follows:*

$$\Pi_{\mathrm{t}}((P,id,E)) = (P,\Pi_{\mathcal{M}}(id),[x \mapsto \Pi_{\mathcal{N}}(E(x))])$$

We can now define the projection of a computation sequence. At each computation step, we obtain the set of the threads, by projecting all the instances of syntactic components of $\mathcal{S}_{\mathrm{I}}$, and throwing away instances of syntactic components of the context. Unfortunately, the set of unsafe names, and the set of fresh markers, cannot be constructed without any knowledge of the previous computation steps, so we construct them incrementally: at each computation step, we add new unsafe names, and remove used marker. We also remove all computation steps only involving the context. The following definition uses the function *lab* which maps a thread to the first label of its syntactic component.

Definition 16 *Computation sequence projection is then defined as follows:*

Let $\tau = C_0 \xrightarrow{\lambda_1}_{\mathrm{e}} ... \xrightarrow{\lambda_n}_{\mathrm{e}} C_n$ *be a non-standard computation sequence, with* $C_0 \in \mathcal{C}_0^{\mathrm{n}}(\mathcal{S})$*, we define the projection of* τ*,* $\Pi_\tau(\mathcal{S}_{\mathrm{I}},\Phi_{\mathcal{M}},\Phi_{\mathrm{chan}})(\tau)$ *as the non-standard computation sequence*

$$(A_0,U_0,F_0) \overset{\Pi_\lambda(\lambda_{a_1})}{\hookrightarrow} ... \overset{\Pi_\lambda(\lambda_{a_p})}{\hookrightarrow} (A_p,U_p,F_p)$$

of the open system $\mathcal{S}_{\mathrm{I}}$*, where*

- *$a_1, ..., a_p$ is the strictly increasing sequence of the elements of the set $\{i \in [\![1;n]\!] \mid \lambda_i \notin (\mathcal{L} \setminus \mathcal{L}_\mathrm{I})^2\}$;*
- *the initial configuration (A_0, U_0, F_0) is the following triple:*

$$(\{\Pi_\mathrm{t}(T) \mid T \in C_0,\ lab(T) \in \mathcal{L}_\mathrm{I}\}, en, \{t_n \mid n \in \mathbb{N}\});$$

- *for $k \in [\![1;p]\!]$, the configuration (A_k, U_k, F_k) is defined as follows:*
  - $A_k = \{\Pi_\mathrm{t}(T) \mid T \in C_{a_k},\ lab(T) \in \mathcal{L}_\mathrm{I}\}$,
  - $U_k = U_{k-1} \cup \{\Pi_\mathcal{N}(E(x)) \mid \exists T = (a, id, E) \in C_{a_k},\ lab(T) \notin \mathcal{L}_\mathrm{I},\ x \in fn(a)\}$
  - $F_k = \begin{cases} F_{k-1} & \text{if } snd(\lambda_{a_k}) \in \mathcal{L}_\mathrm{I} \\ F_{k-1} & \text{if } \xrightarrow{\lambda_{a_k}}_\mathrm{e} \text{ is not a resource fetching} \\ F_{k-1} \setminus \{\Phi_\mathcal{M}(snd(\lambda_{a_k}), id)\} & \text{otherwise,} \end{cases}$

  where id is the marker of the sent message in $\xrightarrow{\lambda_{a_k}}_\mathrm{e}$ which is the marker of the unique element of the set $\{t \in C_{a_k} \setminus C_{a_k-1} \mid lab(t) = snd(\lambda_k)\}$.

We establish the relation between the non-standard semantic of closed and open systems, as follows:

Theorem 17 (Soundness) *Let τ be a non-standard computation sequence of the following closed system:*

$$(\nu\ c_1)...(\nu\ c_k)(\mathcal{S}_\mathrm{I}(c_{i_1}, ..., c_{i_n}) \mid \mathcal{S}_\mathrm{c}(c_{j_1}, ...c_{j_l})).$$

Then $\Pi_\tau(\mathcal{S}_\mathrm{I}, \Phi_\mathcal{M}, \Phi_\mathrm{chan})(\tau)$ is a non-standard computation sequence of the open system $\mathcal{S}_\mathrm{I}$.

Theorem 18 (Completeness) *Let τ' be a non-standard computation sequence of an open system $\mathcal{S}_\mathrm{I}$. We denote τ' by*

$$(C_0, U_0, F_0) \overset{(i_1,j_1)}{\looparrowright} \ ...\ \overset{(i_n,j_n)}{\looparrowright} (C_n, U_n, F_n),$$

where $(C_0, U_0, F_0) \in \mathcal{C}_0^\mathrm{o}(\mathcal{S})$.

Then there exists:

- *a closed system $\mathcal{S} = (\nu\ \overline{c})(\mathcal{S}_\mathrm{I}(c_{i_1}, ..., c_{i_n}) \mid \mathcal{S}_\mathrm{c}(c_{j_1}, ...c_{j_l}))$,*
- *two one-to-one functions Φ_chan and $\Phi_\mathcal{M}$,*
- *a non-standard computation sequence τ of the system $\mathcal{S}$,*

such that $\Pi_\tau(\mathcal{S}_\mathrm{I}, \Phi_\mathcal{M}, \Phi_\mathrm{chan})(\tau) = \tau'$.

Soundness is ensured by construction. Completeness relies on the existence of the most general context which can be used in simulating any context. It is given in Fig. 8. It uses a global channel, named **unsafe** via which unsafe names are sent an arbitrary amount of times. It is composed by three kinds of resources. The resource **new** opens a new unsafe channel; the resource $\mathbf{spy}_k$

$$
\begin{array}{rl}
\mathcal{S}_{\mathrm{c}} = (\nu\ \mathrm{select})(\nu\ \mathrm{new}) & \\
& (\ \mathbf{new} \\
& \mid \mathbf{spy}_0 \mid ... \mid \mathbf{spy}_n \\
& \mid \mathbf{spoil}_0 \mid ... \mid \mathbf{spoil}_n \\
& \mid \mathrm{select}![] \mid \mathrm{new}![] \\
&)
\end{array}
$$

where
- $\mathbf{new} := (*\mathrm{new}?[]$
  $((\nu\ \mathit{channel})\ *\mathrm{select}?[](\mathrm{unsafe}![\mathit{channel}] \mid \mathrm{select}![]))$
  $\mid \mathrm{new}![])$
- $\mathbf{spy}_i := (*\mathrm{unsafe}?[c]c?[y_1, ..., y_i](\mathrm{new}![y_1] \mid ... \mid \mathrm{new}![y_i]))$
- $\mathbf{spoil}_i := (*\mathrm{unsafe}?[c]\mathrm{unsafe}?[x_1]...\mathrm{unsafe}?[x_i]c![x_1, ..., x_n])$

Fig. 8. The most general context

collects an unsafe channel c, and receive through this channel a message of arity k; the resource $\mathbf{spoil}_k$ collect an unsafe channel, and k unsafe names, and sends the k names through the channel c. Resources $\mathbf{spy}_0$ and $\mathbf{spoil}_0$ only enforce some synchronization with the system. In Fig. 8, n denotes the greatest arity of the messages occurring in the system part we analyze.

Then the closed system can be chosen as the following one:

$$
\begin{array}{c}
(\nu\ \mathrm{unsafe})(\nu\ x_1)...(\nu\ x_p) \\
(\mathrm{unsafe}![x_1] \mid ... \mid \mathrm{unsafe}![x_p] \mid \mathcal{S}_{\mathrm{I}}(x_{i_1}, ..., x_{i_n}) \mid \mathcal{S}_{\mathrm{c}}(\mathrm{unsafe}))
\end{array}
$$

The non-standard computation sequence is obtained by mimicing spied and spoiled computation step in $\mathcal{S}_{\mathrm{c}}$. □

6 Abstract Interpretation

We denote by $\mathcal{C}$ the set of all possible non-standard configurations and by Σ the set of transition labels. We are actually interested in the set $\mathcal{C}(\mathcal{S})$ of all the configurations a system may take during a finite sequence of computation steps. This is given by its collecting semantics, which is defined in [6]. It can be expressed as the least fix-point of a $\cup$-complete endomorphism $\mathbb{F}$ on the complete lattice $\wp(\Sigma^\star \times \mathcal{C})$ defined as follows:

$$
\mathbb{F}(X) = \{(\varepsilon, \beta(\mathcal{S}))\} \cup \left\{ (u.\lambda, C') \middle| \exists C \in \mathcal{C},\ (u, C) \in X \ \mathit{and}\ C \overset{\lambda}{\looparrowright} C' \right\}
$$

This least fix-point is usually not decidable, so we use the Abstract Interpretation framework [7] to compute a sound – but not necessary complete approximation of it. More precisely, we use the relaxed version of Abstract Interpretation [9], in which the abstract domain is not supposed to be complete under lowest upper bound; furthermore, no abstraction function is required.

Definition 19 *An abstraction is a tuple* $(\mathcal{C}^\sharp, \sqsubseteq^\sharp, \sqcup^\sharp, \perp^\sharp, \gamma, C_0^\sharp, \rightsquigarrow, \nabla)$ *such that*

(1) $(\mathcal{C}^\sharp, \sqsubseteq^\sharp)$ *is a pre-order;*
(2) $\bigsqcup^\sharp : \wp_{\mathit{finite}}(\mathcal{C}^\sharp) \rightarrow \mathcal{C}^\sharp$ *such that* $\forall A^\sharp \in \wp_{\mathit{finite}}(\mathcal{C}^\sharp),\ \forall a^\sharp \in A^\sharp,\ a^\sharp \sqsubseteq^\sharp \bigsqcup^\sharp(A^\sharp)$*;*
(3) $\perp^\sharp \in \mathcal{C}^\sharp$ *satisfies* $\forall a^\sharp \in \mathcal{C}^\sharp,\ \perp^\sharp \sqsubseteq^\sharp a^\sharp$*;*
(4) $\gamma : \mathcal{C}^\sharp \rightarrow \wp(\Sigma^\star \times \mathcal{C})$ *is a monotonic map which satisfies* $\gamma(\perp^\sharp) = \emptyset$*;*
(5) $C_0^\sharp \in \mathcal{C}^\sharp$ *is such that* $\{(\varepsilon, c_0) \mid c_0 \in C_0(\mathcal{S})\} \in \gamma(C_0^\sharp)$*;*
(6) $\rightsquigarrow \in \wp(\mathcal{C}^\sharp \times \Sigma \times \mathcal{C}^\sharp)$ *is an abstract deterministic labeled transition relation over* $\mathcal{C}^\sharp$ *such that :* $\forall C^\sharp \in \mathcal{C}^\sharp,\ \forall (u, C) \in \gamma(C^\sharp),\ \forall \lambda \in \Sigma,\ \forall \overline{C} \in \mathcal{C}$,

$$C \overset{\lambda}{\looparrowright} \overline{C} \Longrightarrow \exists \overline{C}^\sharp \in \mathcal{C}^\sharp,\ (C^\sharp \overset{\lambda}{\rightsquigarrow} \overline{C}^\sharp \text{ and } (u.\lambda, \overline{C}) \in \gamma(\overline{C}^\sharp));$$

(7) $\nabla : \mathcal{C}^\sharp \times \mathcal{C}^\sharp \rightarrow \mathcal{C}^\sharp$ *is a widening operator which satisfies:*
- $\forall C_1^\sharp,\ C_2^\sharp \in \mathcal{C}^\sharp,\ C_1^\sharp \sqsubseteq^\sharp C_1^\sharp \nabla C_2^\sharp$ *and* $C_2^\sharp \sqsubseteq C_1^\sharp \nabla C_2^\sharp$,
- $\forall (C_n^\sharp)_{n\in\mathbb{N}} \in \left(\mathcal{C}^\sharp\right)^{\mathbb{N}}$*, the sequence* $(C_n^\nabla)_{n\in\mathbb{N}}$ *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.

The set $\mathcal{C}^\sharp$ is an abstract domain. It captures the properties we are interested in, and abstracts away many other properties. The pre-order $\sqsubseteq^\sharp$ describes the amount of information which is known about the properties we approximate. We use only a pre-order to allow some concrete properties to be described by several unrelated abstract elements. The abstract union $\bigsqcup^\sharp$ is used to gather the information described by several abstract elements, for the sake of generality, it does not necessarily compute the lowest upper bound of a finite set of abstract elements which may not even exist. $\perp^\sharp$ describes the empty set, it provides the basis for our abstract iteration. γ is a concretization function which maps each abstract property to the set of the concrete elements which satisfy this property. $C_0^\sharp$ is an abstract element which describes the properties satisfied by the initial configuration of the system. $\rightsquigarrow$ is used to mimic the concrete transition system in the abstract domain and ∇ is used to ensure the convergence of the analysis in finitely many iterations.

In accordance with Def. 19.6, the abstract counterpart $\mathbb{F}^\sharp$ to $\mathbb{F}$, defined as:

$$\mathbb{F}^\sharp(X^\sharp) = \bigsqcup^\sharp \left(\left\{\overline{C}^\sharp \mid \exists \lambda \in \Sigma,\ C^\sharp \in X^\sharp,\ C^\sharp \overset{\lambda}{\rightsquigarrow} \overline{C}^\sharp\right\} \cup \{C_0^\sharp; X^\sharp\}\right)$$

satisfies the soundness condition $\forall C^\sharp \in \mathcal{C}^\sharp$, $\mathbb{F} \circ \gamma(C^\sharp) \subseteq \gamma \circ \mathbb{F}^\sharp(C^\sharp)$. Since $\mathbb{F}$ is monotonic, we have $\forall n \in \mathbb{N}, \forall C^\sharp \in \mathcal{C}^\sharp$, $\mathbb{F}^n \circ \gamma(C^\sharp) \subseteq \gamma \circ \mathbb{F}^{\sharp^n}(C^\sharp)$. On the other hand, since $\mathbb{F}$ is a $\cup$-complete endomorphism, Kleene's theorem ensures that we have $\mathit{lfp}_\emptyset\, \mathbb{F} = \bigcup_{n\in\mathbb{N}} \mathbb{F}^n(\emptyset)$. As a consequence, we obtain the soundness of our analysis:

Theorem 20 $\mathit{lfp}_\emptyset\, \mathbb{F} \subseteq \bigcup_{n\in\mathbb{N}} \left[\gamma \circ \mathbb{F}^{\sharp^n}\right](\perp^\sharp)$.

Following [6], we compute a sound and decidable approximation of our abstract semantics by using the widening operator ∇:

Theorem 21 *The abstract iteration [9,10] of $\mathbb{F}^\sharp$ defined as follows:*

$$\begin{cases} \mathbb{F}_0^\nabla & = \perp^\sharp \\ \mathbb{F}_{n+1}^\nabla & = \begin{cases} \mathbb{F}_n^\nabla & \textit{if } \mathbb{F}^\sharp(\mathbb{F}_n^\nabla) \sqsubseteq \mathbb{F}_n^\nabla \\ \mathbb{F}_n^\nabla \nabla \mathbb{F}^\sharp(\mathbb{F}_n^\nabla) & \textit{otherwise} \end{cases} \end{cases}$$

is ultimately stationary and its limit $\mathbb{F}^\nabla$ satisfies $\mathcal{C}(\mathcal{S}) \subseteq \gamma(\mathbb{F}^\nabla)$.

6.1 Abstraction algebra

Our framework is highly extensible. We now give some operations over abstractions to compose them.

6.1.1 Product

Let $(\mathcal{C}_1^\sharp, \sqsubseteq_1^\sharp, \bigsqcup_1^\sharp, \perp_1^\sharp, \gamma_1, C_{0_1}^\sharp, \leadsto_1, \nabla_1)$ and $(\mathcal{C}_2^\sharp, \sqsubseteq_2^\sharp, \bigsqcup_2^\sharp, \perp_2^\sharp, \gamma_2, C_{0_2}^\sharp, \leadsto_2, \nabla_2)$ be two abstractions.

The following tuple $(\mathcal{C}^\sharp, \sqsubseteq^\sharp, \bigsqcup^\sharp, \perp^\sharp, \gamma, C_0^\sharp, \leadsto, \nabla)$ where

- $\mathcal{C}_0^\sharp = \mathcal{C}_1^\sharp \times \mathcal{C}_2^\sharp$;
- $\sqsubseteq^\sharp$, $\bigsqcup^\sharp$, $\perp^\sharp$ and ∇ are defined pair-wise;
- $\gamma \ : \begin{cases} \mathcal{C}^\sharp \to \wp(\Sigma^\star \times \mathcal{C}) \\ (a_1^\sharp, a_2^\sharp) \mapsto \gamma_1(a_1^\sharp) \cap \gamma_2(a_2^\sharp); \end{cases}$
- $C_0^\sharp(\mathcal{S}) = (C_{0_1}^\sharp, C_{0_2}^\sharp)$;
- $\leadsto$ is defined by:
  $(a_1, a_2) \overset{\lambda}{\leadsto} (b_1, b_2)$ if and only if $a_1 \overset{\lambda}{\leadsto}_1 b_1$ and $a_2 \leadsto_2 b_2$.

is also an abstraction.

6.1.2 Reduction

Let $(\mathcal{C}^\sharp, \sqsubseteq^\sharp, \bigsqcup^\sharp, \bot^\sharp, \gamma, C_0^\sharp, \rightsquigarrow, \nabla)$ be an abstraction, and ρ be a reduction operator[3] $\rho : \mathcal{C} \to \mathcal{C}$ which satisfies:

$$\forall a^\sharp \in \mathcal{C},\ \gamma(a^\sharp) \subseteq \gamma(\rho(a^\sharp))$$

The following tuple $(\mathcal{C}^\sharp, \sqsubseteq^\sharp, \bigsqcup^\sharp, \bot^\sharp, \gamma, C_{0_\rho}^\sharp, \rightsquigarrow_\rho, \nabla)$ where

- $C_{0_\rho}^\sharp = \rho(C_0^\sharp)$;
- $\rightsquigarrow_\rho$ is defined by:
  $a \rightsquigarrow_\rho c$ if and only if there exists $b \in \mathcal{C}$, such that $\rho(a) \rightsquigarrow b$ and $c = \rho(b)$.

is also an abstraction.

Furthermore, ρ can also be used to simplify the final result of the abstract iteration.

7 Control flow analysis

7.1 Generic control flow analysis

We propose to compute a description of both the potential interactions between the agents of a mobile system, and the potential interactions between its agents and a hostile context. For that purpose, we will compute for each name restriction an approximation of the set of the syntactic agents the name it declares may be communicated to. As we want a non-uniform description, we will also compute a comparison between the markers of both the agents which have created channel names, and the agents they have been communicated to. Due to the approximation, some of the discovered interactions may be ineffective. But the analyzer detects all the interactions. So, if the analyzer does not detect an interaction between two components, there cannot be any no interaction between them.

The main difficulty is to synthesize comparisons between markers throughout computation steps. We use agent markers as pivots to synthesize the comparison between the markers of a thread. Furthermore, we use synchronization conditions on the channel names via which the communication is performed in establishing a comparison between the thread markers of the two involved threads. Our main strategy is easy: we first gather all the information we have

[3] ρ allows simplifying the properties obtained in the abstract domain.

about the pairs of markers (this means we will abstract sets of tuples of markers). Then, synchronization conditions give equality relations between tuple components. If the equality relations are satisfiable, the abstract computation step is enabled and we compute, for each new thread, the comparison between its thread marker and its other markers.

We introduce for each $n \in \mathbb{N}$ an abstract pre-order $(Id_n^\sharp, \sqsubseteq_n)$ to represent sets of n-tuples of thread markers. Thus, each $Id_n^\sharp$ is related to $\wp(Id^n)$ by a monotonic concretization function γ_n. We introduce few abstract primitives to handle these domains: a representation of the empty set $\perp_n^\sharp$, a representation of the initial marker $\varepsilon^\sharp$, an abstract union $\bigsqcup_n$, a widening operator ∇_n, an associative abstract concatenation $\bullet$ to gather the abstraction of tuple sets, an abstract join *sync* to enforce synchronization conditions, an abstract projection Π and an abstract push operator *push* which is used to calculate the abstraction of the set of the new markers when fetching a resource. These primitives shall satisfy the following properties:

(1) $\gamma_n(\perp_n^\sharp) = \emptyset$;
(2) $\varepsilon \in \gamma_1(\varepsilon^\sharp)$;
(3) $\forall A \in \wp_{\text{finite}}(Id_n^\sharp),\ \bigsqcup_n(A) \in Id_n^\sharp$ and $\forall a^\sharp \in A,\ a^\sharp \sqsubseteq_n \bigsqcup_n(A)$;
(4) $\nabla_n : (Id_n^\sharp)^2 \to Id_n^\sharp$ is a widening operator;
(5) $\forall a \in Id_n^\sharp, b \in Id_m^\sharp,\ (a \bullet b) \in Id_{n+m}^\sharp$ and
$$\left\{ (id_i)_{i\in[\![1;n+m]\!]} \;\middle|\; \begin{array}{l} (id_i)_{i\in[\![1;n]\!]} \in \gamma_n(a^\sharp), \\ (id_{i+n})_{i\in[\![1;m]\!]} \in \gamma_m(b^\sharp) \end{array} \right\} \subseteq \gamma_{n+m}(a \bullet b);$$
(6) $\forall a^\sharp \in Id_n^\sharp,\ \forall A \in \wp([\![1;n]\!]^2),\ sync(A, a^\sharp) \in Id_n^\sharp$ and
$\{(id_i)_{i\in[\![1;n]\!]} \mid (id_i) \in \gamma_n(a^\sharp),\ \forall (k,l) \in A,\ id_k = id_l\} \subseteq \gamma_n(sync(A, a^\sharp))$;
(7) $\forall a^\sharp \in Id_n^\sharp, \forall p \in \mathbb{N},\ \forall (s_k)_{k\in[\![1;p]\!]} \in [\![1;n]\!]^{[\![1;p]\!]}$ one-to-one sequence,
$\underset{(s_k)}{\Pi}(a^\sharp) \in Id_p^\sharp$ and $\{(id_{s_k})_{k\in[\![1;p]\!]} \mid (id_i)_{i\in[\![1;n]\!]} \in \gamma_n(a^\sharp)\} \subseteq \gamma_p(\underset{(s_k)}{\Pi}(a^\sharp))$;
(8) $\forall a^\sharp \in Id_3^\sharp,\ \underset{(i,j)}{push}(a^\sharp) \in Id_2^\sharp$ and
$\{(N((i,j), id_1, id_2), id_3) \mid (id_1, id_2, id_3) \in \gamma_3(a^\sharp)\} \subseteq \gamma_2(\underset{(i,j)}{push}(a^\sharp))$.

Moreover we define the operator $dpush \in \mathcal{F}(Id_1^\sharp \to Id_2^\sharp)$ by:

$$\forall a \in Id_1^\sharp,\ dpush(a) = sync(\{(1,2)\}, a \bullet a).$$

dpush satisfies the following property:

$$\forall a \in Id_1^\sharp,\ \{(id, id) \mid id \in \gamma_1(a)\} \subseteq \gamma_2(dpush(a)).$$

We denote by $\mathcal{P}$ the set of all sub-processes of $\mathcal{S}$. We introduce $\nu n(\mathcal{S})$ as the set of all the syntactic names used in name restrictions, including the name *ext*. We then denote by I the set $\{(p,x,y) \mid p \in \mathcal{P},\ x \in fn(p),\ y \in \nu n(\mathcal{S})\}$.

I is the set of all possible interactions between agents of $\mathcal{S}$. The interaction (p, x, y) denotes the fact that the channel name $x \in fn(p)$ may have been declared by the restriction $(\nu\ y)$, if $y \neq ext$, or may have been declared by the context otherwise.

In this way, our abstraction $(\mathcal{C}^\sharp, \sqsubseteq^\sharp, \bigsqcup^\sharp, \bot^\sharp, \gamma, C_0^\sharp, \rightsquigarrow, \nabla)$ is defined as follows:

- $\mathcal{C}^\sharp = (\mathcal{F}(I \to id_2^\sharp) \times \mathcal{F}(\nu n(\mathcal{S}) \to id_1^\sharp))$;
- $\sqsubseteq^\sharp, \bigsqcup^\sharp, \nabla$ are defined component-wise and pair-wise respectively from $(\sqsubseteq_1, \sqsubseteq_2)$, $(\bigsqcup_1, \bigsqcup_2)$ and (∇_1, ∇_2);
- $\bot^\sharp$ is given by the pair of functions which relate any elements respectively to $\bot_2^\sharp$ and $\bot_1^\sharp$;
- $\gamma((f, g))$ is given by the set of configurations (C, U, F) which satisfy:

(1) $(P, id_P, E) \in C, x \in fn(P), E(x) = (y, id_x) \Longrightarrow (id_P, id_x) \in \gamma_2(f(P, x, y))$,
(2) $(x, id) \in U \Longrightarrow id \in \gamma_1(g(x))$;

- $C_0^\sharp$ and $\rightsquigarrow$ are given in Figs. 10-12.

An abstract configuration is given by two functions. The first one maps each interaction (P, x, y) to the description of the set of marker pairs (id_P, id_x) such that, in the instance of P tagged with the marker id_P, the free name x may be linked to the channel name created by the instance of the name restriction $(\nu\ y)$ tagged with id_x. The second one maps each name x to a description of the set of the markers id such that a name created by the instance of the name restriction $(\nu\ x)$ tagged with the marker id may be communicated to the context. Abstract transition rules just mimic the non-standard ones. Their definition uses an abstract extraction function $\beta^\sharp$ defined in Fig. 9.

$$\begin{aligned}
\beta^\sharp((\nu\ n)P, id^\sharp, E^\sharp) &= \beta^\sharp(P, id^\sharp, E^\sharp[(n, n) \mapsto dpush(id^\sharp)]) \\
\beta^\sharp(\mathbf{0}, id^\sharp, E^\sharp) &= \bot^\sharp \\
\beta^\sharp(P \oplus Q, id^\sharp, E^\sharp) &= \sqcup^\sharp\{\beta^\sharp(P, id^\sharp, E^\sharp); \beta^\sharp(P, id^\sharp, E^\sharp)\} \\
\beta^\sharp(P \mid Q, id^\sharp, E^\sharp) &= \sqcup^\sharp\{\beta^\sharp(P, id^\sharp, E^\sharp); \beta^\sharp(P, id^\sharp, E^\sharp)\} \\
\beta^\sharp(aP, id^\sharp, E^\sharp) &= ([(io, m, n) \mapsto E^\sharp(m, n)\ \ if\ m \in fn(aP)], \emptyset)
\end{aligned}$$

Fig. 9. Abstract semantics: abstract extraction

$\beta^\sharp$ is an abstract counterpart to β. It calculates all the interactions obtained by spawning a continuation having a description of both its marker and its environment: given $P \in \mathcal{P}$, $id^\sharp \in Id_1^\sharp$ and $E^\sharp \in \mathcal{F}((bn(\mathcal{S}) \times \nu n(\mathcal{S})) \to Id_2^\sharp)$, the element $\beta^\sharp(P, id^\sharp, E^\sharp)$ belongs to the abstract domain $\mathcal{C}^\sharp$ and satisfies the following soundness property:

Proposition 22 $\forall P \in \mathcal{P},\ \forall id \in \gamma_1(id^\sharp),\ \forall E \in \mathcal{F}(fn(P) \to (\nu n(\mathcal{S}) \times \mathcal{M}))$ *such that* $\forall m, n \in bn(\mathcal{S}),\ \forall id_n \in \mathcal{M},\ [E(m) = (n, id_n) \implies (id, id_n) \in$

$$C_0^\sharp = \sqcup^\sharp\{\beta^\sharp(\mathcal{S}, \epsilon^\sharp, [(n,0) \mapsto \epsilon^\sharp \bullet t^\sharp, \forall n \in \mathit{fn}(\mathcal{S})]); \{\emptyset; [\mathit{ext} \mapsto t^\sharp]\}\}$$

Fig. 10. Abstract semantics: initial configurations

Let $(f^\sharp_{\text{can}}, f^\sharp_{\text{esc}})$ be an abstract configuration,
if there are $\lambda = y?^i[y_1, ..., y_n]P$, $\mu = x!^j[x_1, ..., x_n]Q$,
if $\sqcup_4\{A(u) \mid u \in \nu n(\mathcal{S})\} \neq \bot_4^\sharp$,
then $(f^\sharp_{\text{can}}, f^\sharp_{\text{esc}}) \rightsquigarrow_{\text{cfa}} \sqcup^\sharp \{(f^\sharp_{\text{can}}, f^\sharp_{\text{esc}}); \beta^\sharp(P, id^\sharp_P, E^\sharp_P); \beta^\sharp(Q, id^\sharp_Q, E^\sharp_Q)\}$ where

- $A(u) = \mathit{sync}(\{2,4\}, f^\sharp_{\text{can}}(\lambda, y, u) \bullet f^\sharp_{\text{can}}(\mu, x, u))$,
- $id^\sharp_P = \sqcup_1\{\prod_{(1)}(A(u)) \mid u \in \nu n(\mathcal{S})\}$,
- $id^\sharp_Q = \sqcup_1\{\prod_{(3)}(A(u)) \mid u \in \nu n(\mathcal{S})\}$,
- $E^\sharp_P = [(m,n) \mapsto \sqcup_2\{I_P(m,n,u) \mid u \in \nu n(\mathcal{S})\}, \forall m \in \mathit{fn}(P), n \in \nu n(\mathcal{S})]$
  where $I_P(m,n,u) = \begin{cases} \prod_{(1,6)} \mathit{sync}(\{3,5\}, A(u) \bullet f^\sharp_{\text{can}}(\mu, x_i, n)) & \mathit{if}\ m = y_i \\ \prod_{(1,6)} \mathit{sync}(\{1,5\}, A(u) \bullet f^\sharp_{\text{can}}(\lambda, m, n)) & \mathit{otherwise}, \end{cases}$
- $E^\sharp_Q = [(m,n) \mapsto \sqcup_2\{I_Q(m,n,u) \mid u \in \nu n(\mathcal{S})\}, \forall m \in \mathit{fn}(Q), n \in \nu n(\mathcal{S})]$
  where $I_Q(m,n,u) = \begin{cases} \prod_{(3,6)} \mathit{sync}(\{3,5\}, A(u) \bullet f^\sharp_{\text{can}}(\mu, m, n)). \end{cases}$

(a) Abstract communication

Let $(f^\sharp_{\text{can}}, f^\sharp_{\text{esc}})$ be an abstract configuration,
if there are $\lambda = *y?^i[y_1, ..., y_n]P$, $\mu = x!^j[x_1, ..., x_n]Q$,
if $\sqcup_4\{A(u) \mid u \in \nu n(\mathcal{S})\} \neq \bot_4^\sharp$,
then $(f^\sharp_{\text{can}}, f^\sharp_{\text{esc}}) \rightsquigarrow_{\text{cfa}} \sqcup^\sharp \{(f^\sharp_{\text{can}}, f^\sharp_{\text{esc}}); \beta^\sharp(P, id^\sharp_P, E^\sharp_P); \beta^\sharp(Q, id^\sharp_Q, E^\sharp_Q)\}$ where

- $A(u) = \mathit{sync}(\{2,4\}, f^\sharp_{\text{can}}(\lambda, y, u) \bullet f^\sharp_{\text{can}}(\mu, x, u))$,
- $id^\sharp_P = \sqcup_1\{\prod_{\{1\}}(\mathit{push}_{(i,j)}(\prod_{(1,3,4)}(A(u)))) \mid u \in \nu n(\mathcal{S})\}$,
- $id^\sharp_Q = \sqcup_1\{\prod_{\{3\}}(A(u)) \mid u \in \nu n(\mathcal{S})\}$,
- $E^\sharp_P = [(m,n) \mapsto \sqcup_2\{I_P(m,n,u) \mid u \in \nu n(\mathcal{S})\}, \forall m \in \mathit{fn}(P), n \in \nu n(\mathcal{S})]$
  where
  $I_P(m,n,u) = \begin{cases} \mathit{push}_{(i,j)}(\prod_{(1,3,6)} \mathit{sync}(\{3,5\}, A(u) \bullet f^\sharp_{\text{can}}(\mu, x_i, n))) & \mathit{if}\ m = y_i \\ \mathit{push}_{(i,j)}(\prod_{(1,3,6)} \mathit{sync}(\{1,5\}, A(u) \bullet f^\sharp_{\text{can}}(\lambda, m, n))) & \mathit{otherwise}, \end{cases}$
- $E^\sharp_Q = [(m,n) \mapsto \sqcup_2\{I_Q(m,n,u) \mid u \in \nu n(\mathcal{S})\}, \forall m \in \mathit{fn}(Q), n \in \nu n(\mathcal{S})]$
  where $I_Q(m,n,u) = \begin{cases} \prod_{(3,6)} \mathit{sync}(\{3,5\}, A(u) \bullet f^\sharp_{\text{can}}(\mu, m, n)). \end{cases}$

(b) Abstract resource fetching

Fig. 11. Abstract semantics: internal interactions

Let $(f^\sharp_{\text{can}}, f^\sharp_{\text{esc}})$ be an abstract configuration,
if there is $\mu = x!^j[x_1, ..., x_n]Q$,
if $\sqcup_3\{A(u) \mid u \in \nu n(\mathcal{S})\} \neq \bot^\sharp_3$,
then $(f^\sharp_{\text{can}}, f^\sharp_{\text{esc}}) \rightsquigarrow_{\text{cfa}} \sqcup^\sharp \{(f^\sharp_{\text{can}}, f^\sharp_{\text{esc}}); \beta^\sharp(Q, id^\sharp_Q, E^\sharp_Q); (\emptyset, \mathit{Escape}^\sharp)\}$ where

- $A(u) = \mathit{sync}(\{(2,3)\}, f^\sharp_{\text{can}}(\mu, x, u) \bullet f^\sharp_{\text{esc}}(u))$,
- $id^\sharp_Q = \sqcup_1\{\prod_{(1)}(A(u)) \mid u \in \nu n(\mathcal{S})\}$,
- $E^\sharp_Q = [(m,n) \mapsto \sqcup_2\{I_Q(m,n,u) \mid u \in bn(\mathcal{S})\}, \forall m \in fn(Q), n \in \nu n(\mathcal{S})]$

  where $I_Q(m,n,u) = \begin{cases} \prod_{(1,5)} \mathit{sync}(\{1,4\}, A(u) \bullet f^\sharp_{\text{can}}(\mu, m, n)) \end{cases}$
- $\mathit{Escape}^\sharp = [m \mapsto \mathcal{E}(m),\ \forall m \in \nu n(\mathcal{S})]$

  where $\mathcal{E}(m) = \sqcup_1\{\prod_{(5)}(\mathit{sync}(\{(1,4)\}, A(u) \bullet (f^\sharp_{\text{can}}(\mu, x_k, m)))) \mid 1 \leqslant k \leqslant n\}$.

(a) Abstract spied communication

Let $(f^\sharp_{\text{can}}, f^\sharp_{\text{esc}})$ be an abstract configuration,
if there is $\lambda = y?^i[y_1, ..., y_n]Q$,
if $\sqcup_3\{A(u) \mid u \in \nu n(\mathcal{S})\} \neq \bot^\sharp_3$,
then $(f^\sharp_{\text{can}}, f^\sharp_{\text{esc}}) \rightsquigarrow_{\text{cfa}} \sqcup^\sharp \{(f^\sharp_{\text{can}}, f^\sharp_{\text{esc}}); \beta^\sharp(P, id^\sharp_P, E^\sharp_P)\}$ where

- $A(u) = \mathit{sync}(\{(2,3)\}, f^\sharp_{\text{can}}(\lambda, y, u) \bullet f^\sharp_{\text{esc}}(u))$,
- $id^\sharp_P = \sqcup_1\{\prod_{(1)}(A(u)) \mid u \in \nu n(\mathcal{S})\}$,
- $E^\sharp_P = [(m,n) \mapsto \sqcup_2\{I_P(m,n,u) \mid u \in \nu n(\mathcal{S})\}, \forall m \in fn(P), n \in \nu n(\mathcal{S})]$

  where $I_P(m,n,u) = \begin{cases} id^\sharp_P \bullet (f^\sharp_{\text{esc}}(n)) & \text{if } m = y_k \\ \prod_{(1,5)} \mathit{sync}(\{1,4\}, A(u) \bullet f^\sharp_{\text{can}}(\lambda, m, n)) & \text{otherwise.} \end{cases}$

(b) Abstract spoiled communication

Let $(f^\sharp_{\text{can}}, f^\sharp_{\text{esc}})$ be an abstract configuration,
if there is $\lambda = *y?^i[y_1, ..., y_n]Q$,
if $\sqcup_3\{A(u) \mid u \in \nu n\} \neq \bot^\sharp_3$,
then $(f^\sharp_{\text{can}}, f^\sharp_{\text{esc}}) \rightsquigarrow_{\text{cfa}} \sqcup^\sharp \{(f^\sharp_{\text{can}}, f^\sharp_{\text{esc}}); \beta^\sharp(P, id^\sharp_P, E^\sharp_P)\}$ where

- $A(u) = \mathit{sync}(\{(2,3)\}, f^\sharp_{\text{can}}(\lambda, y, u) \bullet f^\sharp_{\text{esc}}(u))$,
- $id^\sharp_P = \prod_{(1)}(\underset{(i,0)}{\mathit{push}}((\sqcup_1\{\prod_{(1)}(A(u)) \mid u \in \nu n(\mathcal{S})\}) \bullet t^\sharp \bullet \top^\sharp_{\mathcal{M}}))$,
- $E^\sharp_P = [(m,n) \mapsto \sqcup_2\{I_Q(m,n,u) \mid u \in \nu n(\mathcal{S})\}, \forall m \in fn(P), n \in \nu n(\mathcal{S})]$

  where

  $I_Q(m,n,u) = \begin{cases} (id^\sharp_P \bullet f^\sharp_{\text{esc}}(n)) & \text{if } m = y_k \\ \underset{(i,0)}{\mathit{push}}((\prod_{(1,6,5)} \mathit{sync}(\{1,4\}, A(u) \bullet f^\sharp_{\text{can}}(\lambda, m, n) \bullet t^\sharp))) & \text{otherwise.} \end{cases}$

(c) Abstract spoiled resource fetching

Fig. 12. Abstract semantics: external interactions

$\gamma_2(E^\sharp(m,n))]$, *we have:*

$$\forall F \subseteq \{t_n \mid n \in \mathbb{N}\},\ (\beta(P, id, E), \emptyset, F) \in \gamma(\beta^\sharp(P, id^\sharp, E^\sharp))$$

We now give some intuition about the abstract transition rules. For the sake of brevity, we focus on the abstract communication. Abstract communication between two syntactic components λ and μ simulates all the possible communications between instances of both λ and μ. These communications are first quantified by the channel used by the communication, and more precisely, by the name restriction $(\nu\ u)$ used in opening this channel. So, for each $u \in \nu n(\mathcal{S})$, we compute an abstraction $A(u)$ of the 4-tuples of markers $(id_\lambda, id_u, id_\mu, id'_u)$ such that id_λ, id_μ may be simultaneously the markers of an instance of λ and μ while id_u and id'_u may be the marker of the instance of the channel named u. We obviously enforce the synchronization condition between the second and the fourth components of those tuples, which allows us to detect relations between the first and the third ones. The abstract communication is allowed only if the set of 4-tuples is not empty. In that case, we add the new interactions caused by the continuation of both λ and μ. These continuations are calculated using the abstract extraction $\beta^\sharp$. Continuation marker descriptions are obtained by projecting $A(u)$ onto, respectively, its first and its third components. To calculate abstract environment, we have to pass interactions. An interaction is passed by abstractly concatenating it with each $A(u)$, enforcing the synchronization with the good agent marker, and then projecting the result in accordance to the marker of the agent it is passed. New open channels are dealt automatically by the abstract extraction.

Theorem 23 $(\mathcal{C}^\sharp, \sqsubseteq^\sharp, \bigsqcup^\sharp, \bot^\sharp, \gamma, C_0^\sharp, \rightsquigarrow, \nabla)$ *is an abstraction.*

7.2 Abstract domains

Various domains can be used to instantiate the family of parametric domains $(Id_n^\sharp)_{n\in\mathbb{N}}$, depending on the expected complexity and accuracy. We propose three particular instantiations. The first one abstracts away the information about markers. The result is a uniform control flow analysis. The second one keeps only the equality relations among markers, and gives an analysis which presents strong connections with group creation [4]. The third one allows algebraic comparisons of markers which is, to the best of our knowledge, out of the range of analyzes previously presented in literature.

7.2.1 Uniform Control Flow Analysis

Uniform control flow analysis consists in detecting the potential interactions between syntactic components, without keeping any information about markers. For each channel restriction $(\nu\ x)$, it will capture an upper-approximation of the set of the syntactic components name created by this restriction may be communicated to. Such an analysis can easily be obtained by instantiating all the elements of the family $(Id_n^\sharp)_{n\in\mathbb{N}}$ with the lattice $(\{\bot,\top\},\sqsubseteq)$. The domain $\{\bot,\top\}$ is related to $\wp(Id^n)$ by the concretization function γ_n defined by $\gamma_n(\bot)=\emptyset$ and $\gamma_n(\top)=Id^n$.

The abstract primitives are then defined as follows:

- $\varepsilon^\sharp=\top$;
- $\forall A\in\wp(\{\bot;\top\}),\ \bigsqcup_n(A)=\begin{cases}\bot & \mathit{if}\ \top\notin A\\ \top & \mathit{otherwise};\end{cases}$
- since there are no infinite sequence in $\{\bot;\top\}$ we can use the abstract union as a widening operator;
- $\forall a,b\in\{\bot,\top\},\ (a\bullet b)=\begin{cases}\bot & \mathit{if}\ a=\bot\ \mathit{or}\ b=\bot\\ \top & \mathit{if}\ a=\top\ \mathit{and}\ b=\top;\end{cases}$
- $\forall a\in\{\bot,\top\},\ \mathit{sync}(A,a)=a,\ \prod_X(a)=a\ \mathit{and}\ \underset{(i,j)}{\mathit{push}}(a)=a.$

The resulting analysis is always at least as precise as [2,3]. Nielson and al.'s analysis computes the least element of a Moore family defined as the solution family of a constraint system. Since our abstract union is exact, and we do not use widening operator, the result of our abstract semantics is the least fix-point of the abstract endomorphism induced by $C_0^\sharp$ and $\rightsquigarrow$. This least fix-point is also the least element of a Moore family defined as the set of the solutions of a constraint system. Then, comparing the constraints involved in the two analyses, it turns out that our constraint system implies Nielson and al.'s one. So, any solution of our system is also a solution of Nielson and al.'s one and the least solution of ours is more precise that Nielson and al.'s one.

Roughly speaking, Nielson and al.'s analysis does not take into account action sequentiality: the constructed system only depends on the syntactic action set. Furthermore, it cannot infer distinct interaction sets for two distinct occurrences of the same channel name.

7.2.2 Confinement

We now focus on the equality relations between markers. This allows us to analyze whether or not a channel name can be communicated only to the recursive instance which has created it.

For that purpose, we use a graph-based domain used to represent equality relations between tuple components. Each vertex describes a component; a path between two vertices expresses the fact that the two related components are always equal. We then lift that domain with an extra element in order to represent a non-satisfiable property. We first define the set $\mathcal{G}_n$ of all the non-oriented graphs having vertices in $[\![1;n]\!]$. The transitive closure of a graph $(\mathcal{G},\curvearrowright)$ is denoted by $(\mathcal{G},\curvearrowright^*)$. The pre-order, the concretization function and the abstract primitives are defined on $\mathcal{G}_n$ as follows:

- $\forall \curvearrowright_1,\curvearrowright_2 \in \wp([\![1;n]\!]^2), ([\![1;n]\!],\curvearrowright_1) \sqsubseteq_n ([\![1;n]\!],\curvearrowright_2) \iff \curvearrowright_2 \subseteq \curvearrowright_1$,
- $\forall([\![1;n]\!],\curvearrowright) \in \mathcal{G}_n,\ \gamma_n(([\![1;n]\!],\curvearrowright)) = \{(id_i)_{i\in[\![1;n]\!]} \mid k \curvearrowright l \implies id_k = id_l\}$;
- We also use $\bigsqcup_n$ as a widening operator since $\mathcal{G}_n$ is height bounded;
- $\varepsilon^\sharp = (\{1\},\emptyset)$;
- $\forall A \in \wp(\mathcal{G}_n),\ \bigsqcup_n(A) = ([\![1;n]\!];\curvearrowright_\cup)$
  where $i \curvearrowright_\cup j \iff \forall([\![1;n]\!],\curvearrowright) \in A, i \curvearrowright^* j$;
- $\forall a=([\![1;n]\!],\curvearrowright_a) \in \mathcal{G}_n, b=([\![1;m]\!],\curvearrowright_b) \in \mathcal{G}_m,\ (a \bullet b) = ([\![1;n+m]\!],\curvearrowright_\bullet)$
  where $i \curvearrowright_\bullet j \iff \begin{cases} i \curvearrowright_a j \ \mathit{if}\ i,j \in [\![1;n]\!] \\ (i-n) \curvearrowright_b (j-n)\ \mathit{if}\ i,j \in [\![n+1;n+m]\!]; \end{cases}$
- $\forall([\![1;n]\!],\curvearrowright) \in \mathcal{G}_n, A \in \wp([\![1;n]\!]^2),\ \mathit{sync}(A,([\![1;n]\!],\curvearrowright)) = ([\![1;n]\!], A \cup \curvearrowright)$;
- $\forall a = ([\![1;n]\!],\curvearrowright_a) \in \mathcal{G}_n,\ \prod_s(a) = ([\![1;p]\!],\curvearrowright_\Pi)$
  where $s = (s_k)_{k\in[\![1;p]\!]}$ and $i \curvearrowright_\Pi j \iff i,j \in [\![1;p]\!],\ s_i \curvearrowright_a^* s_j$;
- $\forall a \in \mathcal{G}_3,\ \underset{(i,j)}{\mathit{push}}(a) = ([\![1;2]\!],\emptyset)$;

Roughly speaking, partial order is the opposite of the constraint set inclusion because, the more constraints there are, the fewer solutions there are. The concretization of a graph is the set of all the tuples the components of which satisfy the equality relations described by the edges of this graph. The representation of the empty word is just a graph with one vertex. Gathering some abstract elements consists in intersecting the constraint sets they are described with. Abstract concatenation is just a non-relational union of two graphs, after a renaming of the vertices of the second one. Synchronization consists in adding new constraints. Projection consists in restricting the set of the vertices, keeping equality relations on those vertices. When fetching a resource, we know that the new instance marker is fresh, so it is distinct from all the markers of the channel it is passed. Before applying union and projection, we must closed graphs, least we should lose information.

We then lift each $\mathcal{G}_n$ by adding an extra element $\bot_n$. This element $\bot_n$ is the least element of $(\mathcal{G}_n \cup \{\bot_n\})$. The concretization of $\bot_n$ is the empty set.

Abstract union is lifted as follows:

$$\bigsqcup_n(\emptyset) = \bot_n$$
$$\bigsqcup_n(A \cup \{\bot_n\}) = \bigsqcup_n(A).$$

All other abstract primitives are lifted to be exact, which means that they return the good element in $(\bot_i)_{i\in\mathbb{N}}$ with respect to their image domain as soon as one of their argument is in $(\bot_i)_{i\in\mathbb{N}}$.

As in [4], this analysis can only prove that a channel name is confined inside the scope of the recursive instance which has declared it. It is unable to prove that a name which first exits this scope can then only be sent back to the recursive instance which had created it. The main problem is that we can only propagate equality relations. When a channel is communicated to an agent having a distinct marker, we have no information anymore. Then, if this channel is returned to its opening agent, we cannot infer the good relation. In order to do that we require an algebraic comparison between markers.

7.2.3 Non-Uniform Analysis with Algebraic Comparisons

We now abstract algebraic comparisons between markers. Following Prop. 5, we only abstract the right comb of each tree. We choose $m \in \{1;2\}$, in occordance to the choosen simplification function ϕ_1 or ϕ_2. We introduce Σ_m as the alphabet $\mathcal{L}^m$. We use the reduced-product of two abstractions. Our first abstraction consists in abstracting component-wise the shape of the markers associated to threads and their channel names. The second one infers a comparison between the Parikh vectors of the markers.

7.2.3.1 Regular approximation. We approximate the marker shape in regular languages. For the sake of efficiency, we only use the regular languages which may be described by a set of initial letters, a set of final letters and a succession relation between letters. The result is an efficient abstract domain of languages, the height of which is quadratic with respect to the cardinal of the alphabet Σ_m, and abstract primitives can be computed with a $\mathcal{O}(|\Sigma_m|^2)$ worst-case time cost.

We introduce the set Id^{Reg} of tuples (i, f, t, b) such that $i, f \in \wp(\Sigma_m)$, $t \in (\mathcal{F}(\Sigma_m \to \wp(\Sigma_m)))$ and $b \in \{0;1\}$. Each element $(i, f, t, b) \in Id^{\mathrm{Reg}}$ is related to a language on Σ_m via a concretization function γ^{Reg} defined as follows:

$$u \in \gamma^{\mathrm{Reg}}((i,f,t,b)) \Longleftrightarrow \begin{cases} |u| > 0 \implies u_1 \in i \\ |u| > 0 \implies u_{|u|} \in f \\ \forall i \in [\![1;|u|[\![,\ |u|_{i+1} \in t(u_i) \\ |u| = 0 \implies b = 1. \end{cases}$$

Roughly speaking, i is the set of the initial letters of the language words, f is the set of the final letters. $f(a)$ is the set of letters which may immediately follows an a. The boolean b is equal to 1 if the empty word belongs to the language.

The abstract domain Id^{Reg} is then fitted with a complete lattice structure $(Id^{\mathrm{Reg}}, \perp^{\mathrm{Reg}}, \subseteq^{\mathrm{Reg}}, \bigcap^{\mathrm{Reg}}, \bigcup^{\mathrm{Reg}}, \perp^{\mathrm{Reg}}, \top^{\mathrm{Reg}})$ as follows:

- $\perp^{\mathrm{Reg}} = (\emptyset, \emptyset, [\lambda \mapsto \emptyset], 0)$;
- $\subseteq^{\mathrm{Reg}}$, $\bigcup^{\mathrm{Reg}}$ and $\bigcap^{\mathrm{Reg}}$ are defined component-wise from the usual set operations $\subseteq$, $\bigcup$ and $\bigcap$;
- $\top^{\mathrm{Reg}} = (\Sigma_m, \Sigma_m, [\lambda \mapsto \Sigma_m], 1)$;

Furthermore, the language containing only the empty word is described by $(\emptyset, \emptyset, [\lambda \mapsto \emptyset], 1)$ and is denoted by $\varepsilon^{\mathrm{Reg}}$. At last, we can define the primitive $push^{\mathrm{Reg}} : Id^{\mathrm{Reg}} \times \Sigma_m \to Id^{\mathrm{Reg}}$ which adds a letter at the end of all the words of a language as follows:

$$push^{\mathrm{Reg}}((i,f,t,b),\lambda) = (i',f',t',b') \text{ where } \begin{cases} i' = i \cup \{\lambda \mid \text{if } b = 1\} \\ f' = \{\lambda \mid \text{if } b = 1 \text{ or } i \neq \emptyset\} \\ t' = [a \mapsto t(a) \cup \{\lambda \mid a \in f\}] \\ b' = 0 \end{cases}$$

The emptyness test is given by the following equivalence:

$$\gamma^{\mathrm{Reg}}(i,f,t,b) \neq \emptyset \Longleftrightarrow b = 1 \text{ or } \exists a_0, ..., a_n, \begin{cases} a_0 \in i,\ a_n \in f, \\ \forall k \in [\![0;n[\![, a_{k+1} \in t(a_k) \end{cases}$$

We then introduce the set Id_n^{Reg} of the n-tuples of Id^{Reg}. It is related to $\wp(Idn)$ by the following concretization function:

$$\gamma_n((A_i)_{i\in[\![1;n]\!]}) = \{(id_i)_{i\in[\![1;n]\!]} \mid \forall i \in [\![1;n]\!],\ \phi_m(id_i) \in \gamma^{\mathrm{Reg}}(A_i)\}$$

Then the associated abstract primitives are defined as follows:

- $\varepsilon^\sharp = \varepsilon^{\mathrm{Reg}}$ is the language $\{\varepsilon\}$;

- $\bigsqcup_n$ applies $\bigsqcup^{\text{Reg}}$ component-wisely;
- $\bullet$ (resp. Π) is the classical concatenation (resp. projection) of tuples;
- $sync(\{(a,b)\} \cup A, Q)_i = \begin{cases} (sync(A,Q))_i & \text{if} i \notin \{a;b\} \\ (sync(A,Q))_a \sqcap^{\text{Reg}} (sync(A,Q))_b & \text{otherwise;} \end{cases}$

  and $sync(\emptyset, Q) = Q$.
- since there are no infinite increasing sequences,we define our widening operator ∇_n to be $\bigsqcup^n$;
- $\underset{\lambda}{push}(a,b,c) = push^{\text{Reg}}(b,\lambda)$.

7.2.3.2 Numerical approximation. Our second abstraction captures relational comparisons between occurrence number of each pattern inside sets of marker pairs. For each $n \in \mathbb{N}$, we introduce a set $\mathcal{V}_n$ of distinct variables $\{x_i^\lambda \mid i \in [\![1;n]\!],\ \lambda \in \Sigma_m\}$. The abstract domain $\wp(\mathbb{N}^{\mathcal{V}_n})$ is related to $\wp(Id^n)$ by the monotonic map γ_n:

$$\gamma_n(A) = \{(id_i)_{i\in[\![1;n]\!]} \mid \exists (n_t)_{t\in\mathcal{V}_n} \in A,\ \forall x_i^\lambda \in \mathcal{V}_n,\ n_{x_i^\lambda} = |\phi_m(id_i)|_\lambda\}.$$

The power set $\wp(\mathbb{N}^{\mathcal{V}_n})$ is then related to a numerical domain. Many relational numerical domains have been introduced in the literature [21,11,16,24]. We propose two choices in accordance to the expected trade-off between complexity and accuracy. They both use the complete lattice of the affine equality systems among a set of variables. This domain is described with its lattice operations in [21].

7.2.3.2.1 Component-wise affine comparison

The first choice for abstracting a word tuple set consists in abstracting for each λ in Σ_m the affine relations between the occurrence number of λ in each tuple component. This way, we introduce Id_n^{Rel} as the complete lattice $\mathcal{F}(\Sigma_m, \mathcal{K}_{\{x_i \mid i\in[\![1;n]\!]\}})$ defined point-wisely.

We then define abstract primitives as follows:

- $\varepsilon^\sharp = \left[\lambda \mapsto \left\{x_1 = 0 \right.\right]$;
- $\bigsqcup_n$ applies the affine hull point-wise;
- Π applies point-wise the affine projection. The affine projection of a system k onto a subset of variables A consist in using Gaussian reduction to compute all the constraints which are only involving the variables in A, and forgetting other constraints;
- $\forall f \in Id_n^{\text{Rel}}, g \in Id_m^{\text{Rel}},\ f \bullet g = [\lambda \mapsto f(\lambda) \sqcap_{\mathcal{K}} (g(\lambda)[x_i \mapsto x_{i+n}])]$;
- $sync(A, f) = [\lambda \mapsto f(\lambda) \sqcap_{\mathcal{K}} \{x_a = x_b,\ \forall (a,b) \in A]$;
- since there are no infinite increasing sequence, we define our widening operator ∇_n to be $\bigsqcup_n$;

- $\underset{\lambda}{push}(f)$ is obtained in by first replacing each occurrence of the variable x_2 in the constraints of the system $f(\lambda)$ by teh expression $x_2 - 1$, and then applying the abstract projection $\prod\limits_{(2,3)}$.

7.2.3.2.2 Global affine comparison

The second choice consists in abstracting globally all constraints. This way, we introduce for each $n \in \mathbb{N}$ the set of variables $\mathcal{V}_n = \{x_i^\lambda \mid i \in [\![1; n]\!],\ \lambda \in \Sigma_m\}$. Roughly speaking, the variable x_i^λ denotes the occurrence number of λ in the i-th component of marker tuples. We introduce Id^{Rel} as the complete lattice $\mathcal{K}_{\mathcal{V}_n}$ of the affine equality systems on the set of variables $\mathcal{V}_n$. Lattices operations are described in [21]. We describe the remaining primitives as follows:

- $\varepsilon^\sharp$ is given by the system $\{x_1^\lambda = 0,\ \forall \lambda \in \Sigma$;
- given $K \in Id_n^{\mathrm{Rel}}$ and $K' \in Id_m^{\mathrm{Rel}}$, we obtain the abstract concatenation of K and K', by renaming each variable x_i^λ to x_{i+n}^λ in K', and gathering all the constraints of the two systems;
- $sync(\{i_1 = j_1; ...; i_p = j_p\}, K)$ corresponds to inserting all the constraints of the form $x_{i_k}^\lambda = x_{j_k}^\lambda,\ \forall k \in [\![1; p]\!],\ \lambda \in \Sigma$ in K;
- $\prod\limits_{(i_1,\ldots,i_p)}(K)$ corresponds to collecting all the constraints involving only the variables $\{x_{i_k}^\lambda\}$ and then renaming each variable $x_{i_k}^\lambda$ into the variable x_k^λ;
- $\underset{(i,j)}{push}(K)$ is obtained by replacing first in each constraint each occurrence of the variable $x_2^{(i,j)}$ by the expression $x_2^{(i,j)} - 1$ and then applying the abstract projection $\prod\limits_{(2,3)}$.

Remark 24 *Global affine comparison domain has not been implanted yet.*

7.2.4 Examples

We now describe results obtained on our examples.

Example 25 *In the* ftp-*server example , the analyzer proves that a channel name created by the restriction (ν **address**) can only be communicated to the variable **email** or to the variable **address**, and that a channel name created by the restriction (ν **request**) can only be communicated to the variable **request**, to the variable **data** or to the variable **rep**. More specifically, it discovers that each time an agent **email**![**rep**] is spawn, there exists n,m in $\mathbb{N}$ such that the agent marker is $(1,16).(1,5)^n.((2,4).(6,3).(2,12))^m.(2,4).(6,3)$; the variable **email** is linked to a channel name created by the restriction (ν **address**) of the instance the marker of which was $(1,16).(1,5)^n$; and the variable **rep** is linked to a channel name created by the restriction (ν **request**) of the instance the marker of which was $(1,16).(1,5)^n$. This is enough to prove that both variables*

email *and* ***data*** *were created by the same instance of the client resource and so the answer to a query can only be sent back to the good client.*

Example 26 *In the token-ring exmaple, the analyzer discovers that in each instance of the agent* **mon![left,right]**, *the variable* ***left*** *is either bound to a name created by an instance of the* (ν ***right***) *restriction or to a name created by the* (ν left0) *restriction, and the variable* ***right*** *is always bound to a name created by an instance of the* (ν ***right***) *restriction. More specifically, in the case where the variable* ***left*** *is bound to a name created by an instance of the* (ν ***right***) *restriction, it discovers that there exists* $n \in \mathbb{N}$ *such that:*

(1) the instance marker of mon![***left***, ***right***] *is* $(\mathbf{1},\mathbf{6})(\mathbf{1},\mathbf{3})^{\mathbf{n+1}}$,
(2) the marker associated to the channel name communicated to the variable ***left*** *is* $(\mathbf{1},\mathbf{6})(\mathbf{1},\mathbf{3})^{\mathbf{n}}$,
(3) and the marker associated to the channel name communicated to the variable ***right*** *is* $(\mathbf{1},\mathbf{6})(\mathbf{1},\mathbf{3})^{\mathbf{n+1}}$.

It is enough to prove that each process can only be linked to either the next one or to the first one.

Remark 27 *Our confinement analysis is not simply an abstraction of our non-uniform analysis since two distinct markers may be recognized by the same automaton while containing the same occurrence number of each pattern (i.e having the same Parikh vector [27]). The equality of the Parikh vectors implies the equality of the markers if they are recognized by an automaton composed of one acyclic path between an initial and a final state without embedded cycle, and such that the set of the Parikh vectors of the cycles of this automaton are linearly independent. Nevertheless, we may use the reduced product of both our confinement analysis and our non-uniform control flow analysis to solve this problem.*

8 Occurrence counting analysis

We propose to count the occurrence number of agents during computation sequences. This abstraction consists in abstracting away any information about markers and environments. This way, a configuration is just seen as a multiset of syntactic agents. In order to capture much more properties, we relate occurrence number of agent to the number of performed transitions.

We denote by $\mathcal{P}$ the set of all sub-processes of $\mathcal{S}$ and choose a set $\mathcal{T}$ of variables used to count performed transition. Since the number of transition labels is quadratic, we may want to quotient this set by an equivalence class, in order to deal with fewer variables. The set $(\mathcal{L}_{\text{used}} \cup \{Extlabel\})^2$ is related to $\mathcal{T}$ by an onto map. We propose three natural choices for $\mathcal{T}$ and ϕ in accordance with the expected trade-off between accuracy and efficiency:

(1) we can use one extra variable by assigning
$\mathcal{T} = \{0\}$ and $\forall i, j \in \mathcal{L}_{\text{used}} \cup \{0\}, \phi(i,j) = 0$,
(2) we can use a linear number of extra variables by assigning
$\mathcal{T} = \{(0,j) \mid j \in \mathcal{L}_{\text{used}} \cup \{0\}\}$ and $\forall i, j \in \mathcal{L}_{\text{used}} \cup \{0\},\ \phi(i,j) = (0,j)$,
(3) we can use a quadratic number of extra variables by assigning
$\mathcal{T} = (\mathcal{L}_{\text{used}} \cup \{0\})^2$ and $\forall i, j \in \mathcal{L}_{\text{used}} \cup \{0\},\ \phi(i,j) = (i,j)$.

Let V be the set $\mathcal{P} \cup \mathcal{T}$. We consider $\wp(\mathbb{N}^V)$, the complete lattice of the sets of natural number functions defined on V. The power set $\wp(\mathbb{N}^V)$ is related to our concrete domain via a concretization function $\gamma_{\mathbb{N}^V}$, where $\forall A^\sharp$, $\gamma_{\mathbb{N}^V}(A^\sharp)$ is defined as follows:

$$\left\{ (u,C) \;\middle|\; \exists f \in A^\sharp, \begin{cases} \forall t \in \mathcal{T}, \left(\sum_{\lambda \in \phi^{-1}(t)} |u|_\lambda \right) = f(t), \\ \forall v \in \mathcal{P},\ \mathrm{Card}(\{(P, id, E) \in C \mid v = P\}) = f(v) \end{cases} \right\}$$

We then introduce a generic pre-order $(\mathcal{N}_V, \sqsubseteq_{\mathcal{N}_V})$ to represent sets of natural number functions defined on V. It is related to $\wp(\mathbb{N}^V)$ via a monotonic concretization $\gamma_{\mathcal{N}_V}$. Furthermore, it is fitted with several generic primitives: a representation $\perp_{\mathcal{N}_V}$ of the empty set, an abstract union $\cup_{\mathcal{N}_V}$, a widening operator $\nabla_{\mathcal{N}_V}$, an abstract counterpart to the addition $+$, an abstract counterpart $-$ to the subtraction, an abstract synchronization $sync_{\mathcal{N}_V}$. We also require the abstraction of some elementary tuples: an abstract element $0_{\mathcal{N}_V}$ to represent the singleton containing the tuple of 0, and $\forall v \in V$, an abstract element $1_{\mathcal{N}_V}(v)$ to represent the singleton containing the tuple the components of which are all equal to 0 except the v component which is equal to 1, which can be denoted by δ^v.

These primitives shall satisfy the following properties:

(1) $\gamma_{\mathcal{N}_V}(\perp_{\mathcal{N}_V}) = \emptyset$,
(2) $\forall A \in \wp_{\text{finite}}(\mathcal{N}_V),\ \cup_{\mathcal{N}_V}(A) \in \mathcal{N}_V$ and $\bigcup_{a^\sharp \in A} \gamma_{\mathcal{N}_V}(a^\sharp) \subseteq \gamma_{\mathcal{N}_V}(\cup_{\mathcal{N}_V}(A))$,
(3) $\nabla_{\mathcal{N}_V}$ satisfies the definition 19.(7),
(4) $\forall a^\sharp, b^\sharp \in \mathcal{N}_V,\ (a^\sharp + b^\sharp) \in \mathcal{N}_V$ and

$\{(a_v + b_v)_{v \in \mathcal{V}} \mid a \in \gamma_{\mathcal{N}_V}(a^\sharp), b \in \gamma_{\mathcal{N}_V}(b^\sharp)\} \subseteq \gamma_{\mathcal{N}_V}(a^\sharp + b^\sharp)$,

(5) $\forall a^\sharp, b^\sharp \in \mathcal{N}_V,\ (a^\sharp \sqsubseteq_{\mathcal{N}_V} b^\sharp) \in \mathcal{N}_V$ and
$\{(a_v - b_v)_{v \in \mathcal{V}} \mid a \in \gamma_{\mathcal{N}_V}(a^\sharp), b \in \gamma_{\mathcal{N}_V}(b^\sharp),\ \forall v \in V,\ a_v \geqslant b_v\} \subseteq \gamma_{\mathcal{N}_V}(a^\sharp - b^\sharp)$,

(6) $\forall a^\sharp \in \mathcal{N}_V,\ I \in \wp(\mathcal{L}),\ sync_{\mathcal{N}_V}(I, a^\sharp) \in \mathcal{N}_V$ and
$\{a \mid a \in \gamma_{\mathcal{N}_V}(a^\sharp), \forall i \in I,\ a(i) \geqslant 1\} \subseteq \gamma_{\mathcal{N}_V}(sync_{\mathcal{N}_V}(I, a^\sharp))$,

(7) $0_{\mathcal{N}_V} \in \mathcal{N}_V$ and $(0)_{i \in V} \in \gamma_{\mathcal{N}_V}(0_{\mathcal{N}_V})$,

(8) $\forall v \in V,\ 1_{\mathcal{N}_V}(v) \in \mathcal{N}_V$ and $(\delta_i^v)_{i \in V} \in \gamma_{\mathcal{N}_V}(1_{\mathcal{N}_V}(v))$.

Roughly speaking, $+$ is an abstract counterpart to the component-wise addition, $-$ is the abstract counterpart to the component-wise subtraction, $sync_{\mathcal{N}_V}$ is used to extract from the abstract value the representation of configurations which simultaneously contain all the processes required by a computation step.

We now define an abstract transition system $(C_0^{\mathcal{N}_V}, \rightsquigarrow_{\mathcal{N}_V})$ in Fig. 13. It uses an abstract extraction function $\beta_{\mathcal{N}_V}$ which computes an approximation of the syntactic agents which are spawned when launching a continuation or the initial system. Internal choice $\oplus$ and parallel composition are just abstracted by their respective abstract counterpart $\cup_{\mathcal{N}_V}$ and $+$. The initial abstract configuration is obtained by applying $\beta_{\mathcal{N}_V}$ to the initial system. An abstract communication between two syntactic components labelled i and j from an abstract state $C^\sharp$ first consists in computing a description of the set of the configurations described in $C^\sharp$ in which the two syntactic components may occur simultaneously. If this set is empty, the components are mutually exclusive and the abstract communication is disabled. Otherwise, the result of the communication is obtained by translation: decreasing the occurrence number of the communicating agents which are not a resource, and adding the description of the two continuations. External interactions are dealt the same way, but only one agent is involved. We do not bother about the control flow, since we use the reduced product between this analysis and our control flow analysis.

Theorem 28 $(\mathcal{N}_V, \sqsubseteq_{\mathcal{N}_V}, \cup_{\mathcal{N}_V}, \perp_{\mathcal{N}_V}, \gamma_{\mathcal{N}_V}, C_0^{\mathcal{N}_V}, \rightsquigarrow_{\mathcal{N}_V}, \nabla_{\mathcal{N}_V})$ *is an abstraction.*

8.2 Abstract domain

We only need to define an abstract domain to approximate sets of tuples of natural numbers in which abstract primitives can be precisely and efficiently implemented. The primitive $sync_{\mathcal{N}_V}$ needs to express the property that several variables may be greater than one, which is a relational information. The shape of the abstract computation rule suggests that the domain should be closed under abstract addition. This problem is very similar to abstracting the occurrence number of ambients [17,26], or even approximating the collecting semantics of a Petri net. We reject the use of usual numerical domains. We are unlikely to design a precise primitive $sync_{\mathcal{N}_V}$ in non-relational domains.

$$
\begin{aligned}
\beta_{\mathcal{N}_V}((\nu\ n)P) &= \beta_{\mathcal{N}_V}(P)\\
\beta_{\mathcal{N}_V}(\mathbf{0}) &= 0_{\mathcal{N}_V}\\
\beta_{\mathcal{N}_V}(P \oplus Q) &= \beta_{\mathcal{N}_V}(P) \cup_{\mathcal{N}_V} \beta_{\mathcal{N}_V}(Q)\\
\beta_{\mathcal{N}_V}(P \mid Q) &= \beta_{\mathcal{N}_V}(P) + \beta_{\mathcal{N}_V}(Q)\\
\beta_{\mathcal{N}_V}(aP) &= 1_{\mathcal{N}_V}(aP)
\end{aligned}
$$

(a) Abstract extraction

$$C_0^{\mathcal{N}_V} = \beta_{\mathcal{N}_V}(\mathcal{S})$$

(b) Initial configuration

Let $v^\sharp$ be an abstract configuration,
if there is $i, j \in \mathcal{L}_{\text{used}} \cup \{0\}$,
such that $\mathit{sync}_{\mathcal{N}_V}(\{\mathit{agent}(l) \mid l \in \{i,j\} \setminus \{0\}\}, v^\sharp) \neq \bot_{\mathcal{N}_V}$,
then
$v^\sharp \overset{(i,j)}{\rightsquigarrow}_{\mathcal{N}_V} (\mathit{sync}_{\mathcal{N}_V}(\{\mathit{agent}(l) \mid l \in \{i,j\} \setminus \{0\}\}, v^\sharp) + \mathit{Act}(i) + \mathit{Act}(j) + 1_{\mathcal{N}_V}(\phi(i,j))$,
where

- $\forall l \in \mathcal{L}_{\text{used}} \setminus \{0\},\ \mathit{agent}(l) \in \mathcal{P} \subset V$ is the syntactic agent labelled with l,
- $\mathit{Act}(l) = \begin{cases} 0_{\mathcal{N}_V} & \textit{if } l = 0 \\ \beta_{\mathcal{N}_V}(P) - 1_{\mathcal{N}_V}(aP) & \begin{array}{l}\textit{if agent}(i) = aP \\ \textit{and a matches with } y?^i[\overline{y}] \textit{ or } x!^i[\overline{x}]\end{array} \\ \beta_{\mathcal{N}_V}(P) & \textit{if agent}(i) \textit{ matches with } {*y?^i}[\overline{y}]P. \end{cases}$

(c) Abstract transition rule

Fig. 13. Occurrence counting analysis

Disjunctive completion can be used to lift a non-relational domain into a relational one: each configuration is abstracted in a finite non-relational domain, and then the abstraction of a configuration set is given by the collection of all the abstractions of its elements. Nevertheless, disjunctive completion often leads to a lack of accuracy and an exponential explosion. We cannot afford the domain of linear inequalities among a finite set of variables [11] because we deal with too many variables.

We propose the use of a reduced product of two domains: the interval lattice and the linear equalities lattice described in [21]. The first domain is based on the use of the interval lattice and is used for expressing properties of interest. This domain can represent all the information we need to express non-exhaustion of resources, but it cannot calculate them precisely without being refined. The second domain is based on the use of linear equalities between variables [21] and is used for expressing more complex properties, such as mutual exclusion for instance, which allows for more precise calculations in the first domain. The power of our analysis directly comes from the use of an unexpensive algorithm, straightforwardly adapted from Linear Constraint Programming, to calculate an approximated reduction between these two domains.

8.2.1 Interval domain

The complete lattice $(\mathcal{I}, \sqsubseteq_{\mathcal{I}}, \sqcup_{\mathcal{I}}, \bot_{\mathcal{I}}, \sqcap_{\mathcal{I}}, \top_{\mathcal{I}})$, where $\mathcal{I} = \mathcal{F}(V \rightarrow \mathbb{N} \times \mathbb{N} \cup \{+\infty\})$ is the functional domain of the natural number intervals, where lattice operations are defined point-wisely. A family $(\nabla^n_{\mathcal{I}})$ of widening operators on $\mathcal{I}$ is defined as follows:

$$[f \nabla^n_{\mathcal{I}} g](x) = f(x) \nabla^n g(x)$$

$$\text{where} \begin{cases} [\![a;b]\!] \ \nabla^n \ [\![c;d]\!] = [\![min\{a;c\};+\infty|[\ \text{if} d > max\{b;n\} \\ \quad I \quad \nabla^n \quad J \quad = \quad I \cup J \quad \text{otherwise} \end{cases}$$

We can easily define abstract primitives in $\mathcal{I}$ as follows:

- $(f +_{\mathcal{I}} g) = [x \rightarrow f(x) + g(x)]$,
- $(f -_{\mathcal{I}} g) = [x \rightarrow f(x) - g(x)] \cap [\![0;+\infty[\![$,
- $sync_{\mathcal{I}}(I, f) = \left[x \rightarrow \begin{cases} f(x) \cap [\![1;+\infty[\![& \text{if } x \in I \\ f(x) & \text{otherwise} \end{cases} \right]$,
- $0_{\mathcal{I}} = [x \rightarrow [\![0;0]\!] \ ]$,
- $1_{\mathcal{I}}(v) = \left[\begin{cases} x \rightarrow [\![1;1]\!] & \text{if } x = v \\ x \rightarrow [\![0;0]\!] & \text{otherwise} \end{cases} \right]$.

8.2.2 Linear equality domain

The complete lattice $(\mathcal{K}, \sqsubseteq_{\mathcal{K}}, \cup_{\mathcal{K}}, \top_{\mathcal{K}}, \cap_{\mathcal{K}}, \bot_{\mathcal{K}})$ of linear equality systems between the finite set of variables V is described with its lattice operations in [21]. This domain uses Gauss reduction in order to normalize systems. Moreover, since there are no infinite increasing chain [21], we can choose $\cup_{\mathcal{K}}$ as a widening operator. To define the addition and the subtraction of two systems, we compute a particular solution of each system O_1 and O_2, and a linear direction $\overrightarrow{H_1}$ and $\overrightarrow{H_2}$. Then we use the following equalities:

$$(O_1 + \overrightarrow{H_1}) +_{\mathcal{K}} (O_2 + \overrightarrow{H_2}) = (O_1 + O_2) + (\overrightarrow{H_1} \cup_{\mathcal{K}} \overrightarrow{H_2}),$$
$$(O_1 + \overrightarrow{H_1}) -_{\mathcal{K}} (O_2 + \overrightarrow{H_2}) = (O_1 - O_2) + (\overrightarrow{H_1} \cup_{\mathcal{K}} \overrightarrow{H_2}).$$

Such a decomposition can be extracted from the normal form in linear time, so the cost of affine addition and subtraction is cubic.

Other primitives are defined as follows:

- $sync_{\mathcal{K}}(I, K) = K$,
- $0_{\mathcal{K}} = \{x = 0,\ \forall x \in V$,
- $1_{\mathcal{K}}(v) = \begin{cases} x = 0 \text{ if } x \in V \setminus \{v\} \\ x = 1 \text{ otherwise} \end{cases}$

Roughly speaking, synchronization cannot be directly checked in $\mathcal{K}$. So we define it as the identity. Linear constraints will therefore be used to prove that synchronization interval constraints are incompatible, by reduction. Other primitive definitions are straightforward.

8.2.3 Reduced product

Our numerical domain is the product $(\mathcal{I} \times \mathcal{K})$. It is partially ordered by the pair-wise order $\sqsubseteq_{\mathcal{I}} \times \sqsubseteq_{\mathcal{K}}$. It is related to $\wp(\mathbb{N}^V)$ by the concretization function $\gamma_{\mathcal{N}_V}$ where $\gamma_{\mathcal{N}_V}(i, s)$ is defined as $\gamma_{\mathcal{I}}(i) \cap \gamma_{\mathcal{K}}(s)$. Generic primitives are expressed as follows:

- $\perp_{\mathcal{N}_V}$, $\cup_{\mathcal{N}_V}$, $\nabla_{\mathcal{N}_V}$, $+$, $\sqsubseteq_{\mathcal{N}_V}$, $0_{\mathcal{N}_V}$, $1_{\mathcal{N}_V}$ are defined pair-wisely,
- $sync_{\mathcal{N}_V}(A, (i, s)) = \rho(i', s)$,

  where $\begin{cases} i'(x) = i(x) \cap [\![1; +\infty|[& \forall x \in A \\ i'(x) = i(x) & \forall x \in \mathcal{P} \setminus A, \end{cases}$

  and ρ satisfies $\forall a \in \mathcal{I} \times \mathcal{K},\ \gamma_{\mathcal{N}_V}(a) \sqsubseteq_{\mathcal{N}_V} \gamma_{\mathcal{N}_V}(\rho(a))$.

The definition of $sync_{\mathcal{N}_V}$ uses a reduction operator ρ which satisfies the following property:

$$\forall a \in \mathcal{I} \times \mathcal{K},\ \gamma_{\mathcal{N}_V}(a) \sqsubseteq_{\mathcal{N}_V} \gamma(\rho(a)).$$

Roughly speaking, the operator ρ is a reduction, it simplify constraints without losing any solutions. We now present a reduction [8] ρ between $\mathcal{I}$ and $\mathcal{K}$. It consists in taking into account linear constraints in order to narrow the domain of interval variables. For instance, the system of constraints $\{x + y = 12,\ x \in [\![3; 15]\!],\ y \in [\![4; 19]\!]$ can be reduced to the system $\{x + y = 12,\ x \in [\![3; 8]\!],\ y \in$

$[\![4;9]\!]$. Linear constraints are likely to be combined, via Gauss reduction, in order to give new linear constraints which will allow for further reductions. Therefore, generating the whole set of such combinations is likely to require an exponential time of execution.

We propose a two-step polynomial algorithm for solving this problem. The first step aims at narrowing infinite intervals into finite ones. It uses Gauss reduction to obtain a positive representation of systems of linear equalities, that is to say, an equivalent system of equations such that if a variable occurs with a strictly negative coefficient in an equation, then this variable occurs with a negative coefficient in each equation. Positive representations contain only a few undefined forms, which allows narrowing infinite intervals into finite ones with a $\mathcal{O}(|V|^3)$ worst-case. The second step is inspired by [5]: it consists in obtaining a triangular system of constraints of the form $a_1.x_1 + ... + a_n.x_n \in I$ where I is an interval. This system is then used to propagate unidirectionally intervals from non-diagonal to diagonal variables. The result is a good reduction with a $\mathcal{O}(|V|^4)$ worst-case.

8.2.3.1 Solving undefined forms Let V_{inf} be a subset of V and K a finite system of linear constraints on the variables V. We denote by K the system of equations:

$$\left\{ \sum_{v \in V} a_i^v.v = b_i, \forall i \in [\![1;n]\!] \right. .$$

We first define a positive form with respect to V_{inf} as follows:

Definition 29 *K is in positive form with respect to V_{inf} if and only if $\forall v \in V$, $\exists i_1, i_2 \in [\![1;n]\!]$ such that $a_{i_1}^v < 0 < a_{i_2}^v \implies v \notin V_{\mathrm{inf}}$.*

This way, in the matrix describing K, the variables which may occur with both a positive and a negative coefficient are known to be bounded[4]. Such a form can be computed by using the Gaussian reduction with a $\mathcal{O}(|V|^3)$ worst-case time cost.

A positive form contains only few undefined forms. For each constraint in which all variables with an infinite range occurs with the same sign, we can narrow their range into finite intervals. A dynamic resolution of such a system leads to a reduction step in $\mathcal{O}(|V|^2)$.

8.2.3.2 Narrowing finite intervals The second step is inspired by [5]: it consists in obtaining a triangular system of constraints of the form $a_1.x_1 +$

[4] V_{inf} denotes the set of the variables which are not proved to be bounded.

... $+ a_n.x_n \in I$ where I is an interval. This system is then used to propagate unidirectionally intervals from non-diagonal to diagonal variables. The result is a good reduction with a $\mathcal{O}(|V|^4)$ worst-case time cost.

We use three kinds of reduction:

(1) Gaussian reduction:

$$\begin{cases} x+y+z=1 \\ x+y+t=2 \end{cases} \implies \begin{cases} x+y+z=1 \\ t-z=1, \end{cases}$$

(2) interval propagation:

$$\begin{cases} x+y+z=3 \\ x \in [\![0;5]\!] \\ y \in [\![0;6]\!] \\ z \in [\![0;8]\!] \end{cases} \implies \begin{cases} x+y+z=3 \\ x \in [\![0;\mathbf{3}]\!] \\ y \in [\![0;6]\!] \\ z \in [\![0;8]\!], \end{cases}$$

(3) redundancy introduction:

$$\begin{cases} x+y-z=3 \\ x \in [\![1;2]\!] \end{cases} \implies \begin{cases} x+y-z=3 \\ y-z \in [\![1;2]\!] \\ x \in [\![1;2]\!], \end{cases}$$

We first use Gaussian reduction to get a normal form of the linear constraint system, we then use interval propagation to narrow the range of the interval of the pivot of each constraint. Then, we forget about the pivot using redundancy introduction. We then get a new system involving only variables which were not pivot in the previous one. We then proceed recursively with it until constraints contain some variables. We then consider all the constraints we have computed which form a triangular system. We propagate the information we have collected on the interval ranges backward, by applying interval propagation from non-diagonal variables to diagonal ones, starting from the last constraints and ending with the first one.

8.2.4 Examples

We now propose some examples of mobile systems analyzed with our prototype. For the sake of brevity and simplicity, we do not present linear constraints. They are not of interest in the result as they are used internally to improve the interval information. Interval constraints are tagged with the actions of syntactic agents.

Example 30 *We give in Fig. 14 the result of occurrence analysis of the* ftp-

server. This result ensures that no more than three syntactic instances of the syntactic agent ***deal***!*[**data**] can simultaneously occur. So we can conclude that no more than three connection can be active at the same time. This constraint is proved using the linear constraint which proves that the sum of the number of available ports* port![] *and activated sessions* ***deal***!*[**data**] is always equal to three.*

Example 31 *We give in Fig. 15 the occurrence analysis result for the* token-ring. *This result ensures that only one at any time syntactic instance of the agent* ***crit***!*[] may occur. So only one process can proceed its critical section at same time.*

Remark 32 *The analysis cannot succeed in proving that the occurrence number of the syntactic agent* **mon**![***left***, **left0**] *is less than or equal to* 1 *without counting the number of performed transitions. On the other hand, only a linear number of extra variables are required to prove this property.*

9 Complexity and benchmark

9.1 About the complexity of our analyses

We shortly describe the complexity in the worst case of our analyses in Figs. 16 and 17. We denote by n the number of syntactic components in the inital system. The first line denotes the cost to scan all the possible syntactic redex. The second line denotes the maximum number of stages in the abstract iteration. The third line denotes the cost to perform an abstract transition, it includes both abstract operation cost and information propagation. The worst-case time complexity of an analysis is then the product of those three partial worst-case time complexities.

In the third column of Fig. 16, m is a parameter of our analysis which denotes analysis precision. We have $m \in \{1; 2\}$ and in the case when $m = 1$ each transition label is abstracted by its second component, while the two components are kept when $m = 2$. In Fig. 17, i is also a parameter of our analysis such that the number of variables is in $\mathcal{O}(n^i)$. It is equal to 0 when no occurrence analysis is done; it is equal to 1 when only syntactic agents are counted; it is also equal to 1 when we also count the number of send message; it is equal to 2 when we count performed transitions number.

To obtain the worst-case time complexity of the product of two analyses, we add pair-wisely the complexities due to redex detection, the maximum number of iteration, and the cost of an abstract transition and then make the

$$
\begin{array}{l}
((\nu\ \mathrm{make})(\nu\ \mathrm{server})(\nu\ \mathrm{port}) \\
((*\mathrm{make}?^{1}[\,](\nu\ \mathit{address})(\nu\ \mathit{request}) \\
\qquad\qquad (\\
\qquad\qquad\ (*\mathit{address}?^{\llbracket 0;+\infty\llbracket}[\,]\mathrm{server}!^{\llbracket 0;+\infty\llbracket}[\mathit{address},\mathit{request}]) \\
\qquad\qquad | \\
\qquad\qquad\ \mathit{address}!^{\llbracket 0;+\infty\llbracket}[\,] \\
\qquad\qquad | \\
\qquad\qquad\ \mathrm{make}!^{\llbracket 0;1\rrbracket}[\,] \\
\qquad\qquad)) \\
| \\
\ (*\mathrm{server}?^{1}[\mathit{email},\mathit{data}](\nu\ \mathit{deal}) \\
\qquad\qquad (\\
\qquad\qquad\ \mathrm{port}?[\,]^{\llbracket 0;+\infty\llbracket} \\
\qquad\qquad\qquad (\\
\qquad\qquad\qquad\ \mathit{deal}!^{\llbracket 0;3\rrbracket}[\mathit{data}] \\
\qquad\qquad\qquad | \\
\qquad\qquad\qquad\ \mathit{deal}?^{\llbracket 0;3\rrbracket}[\mathit{rep}](\mathit{email}!^{\llbracket 0;+\infty\llbracket}[\mathit{rep}] \mid \mathrm{port}!^{\llbracket 0;3\rrbracket}[\,]) \\
\qquad\qquad\qquad) \\
\qquad\qquad \oplus \\
\qquad\qquad\ \mathit{email}!^{\llbracket 0;+\infty\rrbracket}[\,] \\
\qquad\qquad)) \\
\mid \mathrm{port}!^{\llbracket 0;1\rrbracket}[\,] \\
\mid \mathrm{port}!^{\llbracket 0;1\rrbracket}[\,] \\
\mid \mathrm{port}!^{\llbracket 0;1\rrbracket}[\,] \\
\mid \mathrm{make}!^{\llbracket 0;1\rrbracket}[\,])) \\
)
\end{array}
$$

Fig. 14. The *ftp*-server occurrence analysis

$$
\begin{array}{l}
((\nu\ \mathrm{make})(\nu\ \mathrm{mon})(\nu\ \mathrm{left0}) \\
(\\
\ (\ (*\mathrm{make}?^{1}[\mathit{left}](\nu\ \mathit{right})(\mathrm{mon}!^{\llbracket 0;+\infty\llbracket}[\mathit{left},\mathit{right}] \mid \mathrm{make}!^{\llbracket 0;+\infty\llbracket}[\mathit{right}])) \\
\ \mid (*\mathrm{make}?^{1}[\mathit{left}](\mathrm{mon}!^{\llbracket 0;1\rrbracket}[\mathit{left},\mathrm{left0}])) \\
\ \mid \mathrm{make}!^{\llbracket 0;1\rrbracket}[\mathrm{left0}]) \\
| \\
\ ((*\mathrm{mon}?^{1}[\mathit{prev},\mathit{next}] \\
\qquad\qquad (*\mathit{prev}?^{\llbracket 0;+\infty\llbracket}[\,](\nu\ \mathit{crit})(\mathit{crit}?^{\llbracket 0;1\rrbracket}[\,]\mathit{next}!^{\llbracket 0;1\rrbracket}[\,] \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \mid \mathit{crit}!^{\llbracket 0;1\rrbracket}[\,]))) \\
\ \mid \mathrm{left0}!^{\llbracket 0;1\rrbracket}[\,])))
\end{array}
$$

Fig. 15. The token-ring occurrence analysis

	0-CFA	confinement	non-uniform comp.-wise $m \in \{1;2\}$	non-uniform global $m \in \{1;2\}$
scan	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
iteration height	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^{2\times m+3})$	$\mathcal{O}(n^{2\times m+3})$
transition complexity	$\mathcal{O}(n^4)$	$\mathcal{O}(n^4)$	$\mathcal{O}(n^{2\times m+4})$	$\mathcal{O}(n^{3\times m+4})$
total cost	$\mathcal{O}(n^9)$	$\mathcal{O}(n^9)$	$\mathcal{O}(n^{4\times m+9})$	$\mathcal{O}(n^{5\times m+9})$

Fig. 16. CFA-complexity

	occurrence counting analysis $i \in \{0;1;2\}$
scan	$\mathcal{O}(n^2)$
iteration height	$\mathcal{O}((n^i)^2)$
transition complexity	$\mathcal{O}((n^i)^4)$
total cost	$\mathcal{O}(n^{6\times i+2})$

Fig. 17. Occurrence counting analysis complexity

product of these three complexities. So, for instance, the analysis obtained as the product of non-uniform global analysis with $m = 2$, with occurrence counting analysis with $i = 2$, has a $\mathcal{O}(n^2 \times n^7 \times n^{10})$ worst case time cost.

9.2 Analysis time

We report in Figs. 18 and 19 CPU-time used in analyzing our examples on a Sun UltraSPARC-V 333 MHz CPU with 128 Mb Ram. We write time in boldface when the analysis has succeeded in proving all the properties we were interested in.

10 Conclusion

We have designed an automatic analysis for proving some properties of mobile systems. This analysis relies on a non-standard semantics which explicitly encodes the link between agents and the channels they have opened. This semantics handles the full π-calculus and allows us to describe the behaviour of

ftp-server $n = 16$	0-CFA	non-uniform comp.-wise $m = 1$	non-uniform comp.-wise $m = 2$
$i = 0,\ \lvert V\rvert = 0$	0.12s	0.18s	0.16s
$i = 1,\ \lvert V\rvert = n$	1.11s	**1.16s**	**1.12s**
$i = 1,\ \lvert V\rvert = 2 \times n$	2.32s	**2.67s**	**2.68s**
$i = 2,\ \lvert V\rvert = n^2 + n$	2.34s	**2.70s**	**2.68s**

Fig. 18. *ftp*-server analysis time.

Token-ring $n = 12$	0-CFA	non-uniform comp.-wise $m = 1$	non-uniform comp.-wise $m = 2$
$i = 0,\ \lvert V\rvert = 0$	0.08s	0.11s	0.13s
$i = 1,\ \lvert V\rvert = n$	0.41s	0.58s	0.59s
$i = 1,\ \lvert V\rvert = 2 \times n$	1.11s	**1.21s**	**1.25s**
$i = 2,\ \lvert V\rvert = n^2 + n$	1.43s	**1.89s**	**1.89s**

Fig. 19. The token ring analysis time.

a system part, ignoring its context. We have approximated this non-standard semantics, in order to focus on the properties which rely on both control flow and agent occurrence number.

Acknowledgements

We deeply thank Patrick, Radhia Cousot and Arnaud Venet, for their insightful comments and discussions. We are also very grateful to Yves Verhoeven and Antoine Miné who helped me in correcting previous versions of this work.

References

[1] C. Bodei, P. Degano, F. Nielson, and H. R. Nielson. Static analysis of processes for no read-up and no write-down. In Wolfgang Thomas, editor, *Proc. FoSSaCS'99*, volume 1578 of *LNCS*, pages 120–134. Springer, 1999.

[2] C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Static analysis for the π-calculus with applications to security. *Information and Computation*, 165, 2000.

[3] C. Bodei, P. Degano, F. Nielson, and H.R Nielson. Control flow analysis for the π-calculus. In *Proc. CONCUR'98*, Lecture Notes in Computer Science. Springer-Verlag, 1998.

[4] L. Cardelli, G. Ghelli, and A. D. Gordon. Secrecy and group creation. In Catuscia Palamidessi, editor, *Proc. CONCUR'00*, volume 1877 of *LNCS*, pages 365–379. Springer, August 2000.

[5] C.K. Chiu and J.H.M. Lee. Interval linear constraint solving using the preconditioned interval Gaus-Seidel method. In *Proceedings of the Twelfth International Conference on Logic Programming*, Logic Programming, pages 17–32. The MIT Press, 1995.

[6] 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.

[7] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. POPL'77*, pages 238–252, Los Angeles, California, U.S.A., 1977.

[8] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. POPL'79*. ACM Press, 1979.

[9] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of logic and computation*, 2(4):511–547, 1992.

[10] P. Cousot and R. Cousot. Comparing the Galois connection and widening-narrowing approaches to abstract interpretation. In *Proc. PLILP'92*, LNCS, pages 269–295. Springer, 1992.

[11] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proc. POPL'78*. ACM Press, 1978.

[12] N. G. de Bruijn. Lambda-calculus notation with nameless dummies: a tool for automatic formula manipulation with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34(5):381–392, 1972.

[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/dea.ps`.

[14] J. Feret. Confidentiality analysis of mobile systems. In *Proc. SAS'00*, LNCS, pages 135–154. Springer, 2000.

[15] J. Feret. Occurrence counting analysis for the π-calculus. *Electronic Notes in Theoretical Computer Science*, 39(2), 2001.

[16] P. Granger. Static analysis of linear congruence equalities among variables of a program. In *Proc. TAPSOFT'91*, number 493 in LNCS, pages 169–192. Springer-Verlag, 1991.

[17] R. R. Hansen, J. G. Jensen, F. Nielson, and H. Riis Nielson. Abstract interpretation of mobile ambients. In *Proc. SAS'99*, LNCS, pages 134–148. Springer, 1999.

[18] M. Hennessy and J. Riely. Resource access control in systems of mobile agents. In Uwe Nestmann and Benjamin C. Pierce, editors, *Proc. HLCL'98*, volume 16.3 of *ENTCS*, pages 3–17. Elsevier Science Publishers, 1998. Full version as CogSci Report 2/98, University of Sussex, Brighton.

[19] A. Igarashi and N. Kobayashi. A generic type system for the π-calculus. In *Proc. POPL'01*, pages 128–141. ACM, January 2001.

[20] J.-P. Jouannaud. Rewrite proofs and computations. In Helmut Schwichtenberg, editor, *Proof and Computation*, volume 139 of *Computer and Systems Sciences*, pages 173–218. Springer-Verlag, 1995.

[21] M. Karr. Affine relationships among variables of a program. *Acta Informatica*, pages 133–151, 1976.

[22] R. Milner. The polyadic π-calculus: a tutorial. In *Proceedings of the International Summer School on Logic and Algebra of Specification*. Springer Verlag, 1991.

[23] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, pages 1–77, 1992.

[24] A. Miné. The octagon abstract domain. In *Proc. AST'01*, IEEE. IEEE Press, 2001. (to appear).

[25] F. Nielson and H. Seidl. Control-flow analysis in cubic time. In *Proc. ESOP'01*, number 2028 in Lecture Notes in Computer Science, pages 252–268. Springer-Verlag, 2001.

[26] H. Riis Nielson and F. Nielson. Shape analysis for mobile ambients. In *Proc. POPL'00*, pages 142–154. ACM Press, 2000.

[27] R. J. Parikh. On context-free languages. *Journal of the ACM*, 13:570–581, 1966.

[28] S. K. Rajamani and J. Rehof. A behavioral module system for the π-calculus. In *Proc. SAS'01*, LNCS. Springer, 2001.

[29] D. N. Turner. *The Polymorphic π-Calculus: Theory and Implementation*. PhD thesis, Edinburgh University, 1995.

[30] A. Venet. Abstract interpretation of the π-calculus. In Mads Dam, editor, *Analysis and Verification of Multiple-Agent Languages (Proceedings of the Fifth LOMAPS Workshop)*, volume 1192 of *LNCS*, pages 51–75. Springer, 1997.

[31] A. Venet. Automatic determination of communication topologies in mobile systems. In *Proc. SAS'98*, LNCS, pages 152–167. Springer, 1998.