Dependency Analysis of Mobile Systems*

Jrme Feret

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

Abstract. We propose an Abstract Interpretation-based static analysis for automatically detecting the dependencies between the names linked to the agents of a mobile system. We focus our study on the mobile systems written in the π-calculus. We first refine the standard semantics in order to restore the relation between the names and the agents that have declared them. We then abstract the dependency relations that are always satisfied by the names of the agents of a mobile system. That is to say we will detect which names are always pair-wisely equal, and which names have necessarily been declared by the same recursive instance of an agent.

1 Introduction

We are interested in analyzing automatically the behaviour of mobile systems of agents. Agent distribution in such systems may dynamically change during the computation sequences, which makes their analysis difficult. We address the problem of proving *non-uniform properties* about the interactions between the agents of a mobile system; such properties allow distinguishing between several recursive instances of a same agent. We especially intend to infer the *dependency relations* between the names communicated to each agent. This means that we will calculate whether they are pair-wisely equal (and / or) whether they have been pair-wisely declared by the same recursive instance of an agent.

Previous works. In previous articles [7, 8], we proposed several analyses for mobile systems expressed in the π-calculus. In [7], we already proposed a non-uniform abstraction of the interactions between the agents of a mobile system. This analysis takes into account the dynamic creation of both names and agents, which is an inherent feature of mobility: it assigns a unique marker to each agent instance and stamps each channel with the marker of the agent instance which has declared it. Group creation [3] allows proving that a channel name is confined inside the scope of a given recursive instance, but it can only infer equality or disequality constraints between the groups, and cannot prove any equality constraint by composing disequality constraints. In [7], the algebraic

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

properties of markers allow handling more complex properties: we can prove that a channel name is first sent to the next instance and then returned to the previous one; thus, we prove it is returned to the instance which has declared it.

Contribution. The main two drawbacks of [7] are that the properties it captures are very low level and are not easily understandable, and that the only calculated properties are those which involve a comparison between the marker of a channel name and the marker of the agent it is communicated to.

We propose here a more abstract parametric analysis which handles a wider class of properties. It can especially express some relations between names, even if there is no relation between their markers and the marker of the agent instance they are communicated to. Nevertheless, this raises some complexity problems we propose to solve by designing several domains: there is then a trade-off between information partitioning, and the accuracy of information propagation. Reduced product makes these domains collaborate. At this point, [7] can be seen as a particular instantiation of the parametric domain. We also propose two particular domains that aim at discovering and propagating explicit equality and disequality relations among channel names and among markers.

We briefly present, in Sect. 2, the standard semantics of the π-calculus. We recall the non-standard semantics of [7] in Sect. 3 and derive a new generic abstraction of it in Sect. 4. We design some domains in Sect. 5 and describe outlines for reduction in Sect. 6.

2 π-calculus

The π-calculus is a formalism well-fitted for describing the behaviour of mobile systems. It is based on the notion of name passing. We use a lazy synchronous version of the π-calculus which is inspired from [11, 1]. Let $\mathcal{N}$ be an infinite set of channel names; agents are built according to the following syntax:

$$\begin{array}{rl} P, Q ::= & \mathbf{0} \mid \text{action}.P \mid (P \mid Q) \mid (P + Q) \mid (\nu\ x)P \mid [x \diamond y]P \\ \text{action} ::= & c![\overline{x}] \mid c?[\overline{x}] \mid \ *c?[\overline{x}] \end{array}$$

where $c, x, y \in \mathcal{N}$, $\overline{x}$ is a tuple of channel names, and $\diamond \in \{=; \neq\}$. Input guards, replication guards and name restrictions are the only *name binders*, i.e, in $c?[x_1, ..., x_n]P$, $*d?[y_1, ..., y_n]Q$ and $(\nu\ x)R$, occurrences of x_1, ..., x_n in P, y_1, ..., y_p in Q and x in R are said to be bound. Usual rules about scoping, substitution and α-conversion apply. We denote by $fn(P)$ the set of the free names of P, i.e, the names which are not under the scope of any binder, and by $bn(P)$ the set of the bound names of P. The agent $P \mid Q$ denotes the parallel composition of two agents P and Q which performs P and Q simultaneously. The agent $P + Q$ denotes a non-deterministic choice between two agents P and Q which performs either P or Q: the choice is internal and does not depend on the other agents. $[x \diamond y]P$ denotes a matching guard: the agent P can be activated if the guard $[x \diamond y]$ is satisfied and it does not require that the agent P interacts with another agent of the system. We use a lazy version of replication: the agent $*c?[\overline{x}].P$ duplicates itself each time it communicates with another agent.

Example 1. We use, as an illustration, the following system $\mathcal{S}$ [12] throughout the paper:[1]

$$\begin{aligned}
&\textbf{Next} ::= *\text{make}?^0[\mathit{last}](\nu\ \mathit{next})(\text{edge}!^1[\mathit{last};\mathit{next}] \mid \text{make}!^2[\mathit{next}])\\
&\textbf{Last} ::= \text{make}?^3[\mathit{last}].\text{edge}!^4[\mathit{last};\text{first}]\\
&\textbf{Test} ::= \text{edge}?^5[x;y].[x =^6 y][x \neq^7 \text{first}]\text{ok}!^8[]
\end{aligned}$$

$$\mathcal{S} ::= (\nu\ \text{make})(\nu\ \text{edge})(\nu\ \text{first})(\nu\ \text{ok})(\textbf{Next} \mid \textbf{Last} \mid \textbf{Test} \mid \text{make}!^9[\text{first}]).$$

The system $\mathcal{S}$ creates a communication ring between several monitors. Each channel name created by the restrictions **(ν first)** and **(ν *next*)** denotes a monitor. The message **edge**![**x**; **y**] represents a connection between the monitors respectively denoted by **x** and **y**. The first monitor is created by the restriction (ν **first**). The resource **Next** can then be used to connect the last created monitor with a newly created one. The thread **Last** is used to connect the last created monitor with the first one. The thread **Test** is used to test whether a monitor is linked with itself and, in such a case, whether it is not the first monitor. We intend to prove that the first matching pattern of **Test** may be satisfied, although the second matching pattern can never be satisfied. This result is out of the range of any uniform analysis [2] which can give no more accurate result than the fact that monitors can be linked to each other. □

3 Non-standard Semantics

We refine the standard semantics in order to explicitly specify the link between channel names and the agent recursive instances that have declared them: we assign to each instance of an agent an unambiguous marker, and stamp each channel name with the marker of the instance of the agent which has declared it. We consider a closed mobile system $\mathcal{S}$ in the π-calculus, we may assume, without any loss of generality, that no one name occurs twice as an argument of an input guard, a replication guard or a name restriction. Let $\mathcal{L}$ be an infinite set of *labels*. We locate each syntactic component of $\mathcal{S}$ by labeling each action and each matching pattern with a distinct label. We describe each *configuration* of $\mathcal{S}$ by a set of *thread instances*. Each thread instance is a triplet composed of a *syntactic component* which, for the sake of simplicity, will often be denoted by its label, an unambiguous *marker* and an *environment* which specifies the semantic values of the syntactic channel names of the thread. An environment assigns to each syntactic channel name a pair composed of a channel name a and a thread marker *id*, meaning that the channel name has been declared by the name restriction ($\nu\ a$) of a thread the marker of which was *id*.

Thread instances are created at the beginning of the system computation and when agents interact. In both cases, several threads are spawned, in accordance to which non-deterministic choices are made. The function β, defined below, applied to a labeled agent, its marker and its environment, returns the set of all possible combination sets of spawned thread instances:

[1] We have labeled each syntactic component, as explained in Sect. 3.

$$\begin{aligned}
\beta((\nu\ n)P, id, E) &= \beta\left(P, id, (E[n \mapsto (n, id)])\right)\\
\beta(\mathbf{0}, id, E) &= \{\emptyset\}\\
\beta(P+Q, id, E) &= \beta(P, id, E) \cup \beta(Q, id, E)\\
\beta(P \mid Q, id, E) &= \{A \cup B \mid A \in \beta(P, id, E),\ B \in \beta(Q, id, E)\}\\
\beta(action.P, id, E) &= \{\{(action.P, id, E_{|fn(action.P)})\}\}\\
\beta([x \diamond^i y]P, id, E) &= \{\{([x \diamond^i y]P, id, E_{|fn([x\diamond^i y]P)})\}\}
\end{aligned}$$

Markers are the history of the resource duplications which have led to the creation of the agent instances: they are binary trees the nodes of which are labeled with a pair of labels, and the leaves of which are unlabeled (ε). The markers of initial threads are ε, while new thread markers are calculated as follows: when a computation step does not involve fetching a resource, markers of computed threads are just passed to the threads of their continuations; when a resource is fetched, the marker of the new threads created from the continuation of the resource is $N((i,j), id_*, id_!)$ where i and id_* are the label and the marker of the resource, j and $id_!$ are the label and the marker of the message sender.

The set of initial configurations and the computation rules are given in Fig. 1. Standard and non-standard semantics are in *bisimulation*, provided that we restrict ourselves to the set of standard computations where all non-deterministic choices are made before other computation steps.

Example 2. Here is the non-standard configuration for our mobile system $\mathcal{S}$, reached after having replicated the resource **Next** twice, and after having made the last spawned agent labeled by 1 communicated with the thread **Test**:

$$\left\{\begin{array}{l}
\left(0, \varepsilon, \begin{cases}\text{make} \rightarrow (\text{make}, \varepsilon)\\ \text{edge} \rightarrow (\text{edge}, \varepsilon)\end{cases}\right)\\
\left(1, id_1, \begin{cases}\text{edge} \rightarrow (\text{edge}, \varepsilon)\\ last \rightarrow (\text{first}, \varepsilon)\\ next \rightarrow (next, id_1)\end{cases}\right)\\
\left(2, id_2, \begin{cases}\text{make} \rightarrow (\text{make}, \varepsilon)\\ next \rightarrow (next, id_2)\end{cases}\right)\\
\left(3, \varepsilon, \begin{cases}\text{make} \rightarrow (\text{make}, \varepsilon)\\ \text{edge} \rightarrow (\text{edge}, \varepsilon)\\ \text{first} \rightarrow (\text{first}, \varepsilon)\end{cases}\right)\\
\left(6, \varepsilon, \begin{cases}x \rightarrow (next, id_1)\\ y \rightarrow (next, id_2)\\ \text{first} \rightarrow (\text{first}, \varepsilon)\\ \text{ok} \rightarrow (\text{ok}, \varepsilon)\end{cases}\right)
\end{array}\right. \qquad \text{where} \begin{cases} id_1 = N((0,9), \varepsilon, \varepsilon)\\ id_2 = N((0,2), \varepsilon, id_1)\end{cases}$$

It turns out that there is no generic relation between the marker of the agent **6** and the markers of the names linked to the variables **x** and **y**, so both [12, 7] will fail to prove that the second matching pattern is not satisfiable. □

Moreover, in accordance with the following proposition, we can simplify the shape of the markers without losing marker allocation *consistency* which ensures

$$\mathcal{C}_0(\mathcal{S}) = \beta(\mathcal{S}, \varepsilon, \emptyset)$$

(a) Non-standard initial configurations.

$$\frac{E_?(y) = E_!(x),\ \ Cont_P \in \beta(P, id_?, E_{?[y_i \mapsto E_!(x_i)]}),\ Cont_Q \in \beta(Q, id_!, E_!)}{C \cup \{(y?^i[\overline{y}]P, id_?, E_?); (x!^j[\overline{x}]Q, id_!, E_!)\} \longrightarrow (C \cup Cont_P \cup Cont_Q)}$$

$$\frac{E_*(y) = E_!(x),\ \ Cont_P \in \beta(P, N((i,j), id_*, id_!), E_{*[y_i \mapsto E_!(x_i)]}),\ Cont_Q \in \beta(Q, id_!, E_!)}{C \cup \left\{\begin{array}{l}(*y?^i[\overline{y}]P, id_*, E_*);\\ (x!^j[\overline{x}]Q, id_!, E_!)\end{array}\right\} \longrightarrow \big(C \cup \{(*y?^i[\overline{y}]P, id_*, E_*)\} \cup Cont_P \cup Cont_Q\big)}$$

$$\frac{E(x) \diamond E(y),\ \ Cont_P \in \beta(P, id, E)}{C \cup \{([x \diamond^i y]P, id, E)\} \longrightarrow C \cup Cont_P}$$

(b) Non-standard transition system.

Fig. 1. Non-standard semantics.

that no one marker can be assigned twice to the same syntactic agent during a computation sequence.

Proposition 1. *Let ϕ_1 and ϕ_2 be the two following functions:*

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

Marker allocation remains consistent when replacing each marker with 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 Abstract Semantics

We denote by Id the set of all markers, by $\mathcal{E}(V)$ the set of all environments over the set of syntactic names V, by Σ the set $\mathcal{L}^2$ and by $\mathcal{C}$ the set of all non-standard configurations. We are actually interested in $C(\mathcal{S})$, the set of all the configurations a system $\mathcal{S}$ may take during any finite sequence of computation steps. This is the *collecting semantics* [4], which can be expressed as the least fix point of the following $\cup$-complete endomorphism $\mathbb{F}$ on the complete lattice $\wp(\mathcal{C})$:

$$\mathbb{F}(X) = \beta(\mathcal{S}, \varepsilon, \emptyset) \cup \left\{\overline{C} \in \mathcal{C} \ \middle|\ \exists C \in X,\ C \longrightarrow \overline{C}\right\}.$$

The least fix point of such an endomorphism is usually not decidable, so we use a relaxed version of the Abstract Interpretation framework [5] to compute a sound—but not necessary complete—approximation of it.

We assume we are given a family $(\mathcal{G}_V, \sqsubseteq_V, \bigsqcup_V, \bot_V)_{V \subseteq bn(\mathcal{S})}$ of abstract domains of properties. For each $V \subseteq bn(\mathcal{S})$, $\mathcal{G}_V$ is used for globally abstracting the marker and the environment of a thread instance that uses the set of free names V. The relation $\sqsubseteq_V$ is a pre-order which describes the relative amount of information between those properties. Each abstract property is related to $\wp(Id \times \mathcal{E}(V))$ by a monotonic concretization function γ_V. The operator $\bigsqcup_V$ maps each finite set of properties to a weaker property: for each finite $A \subseteq \mathcal{G}_V$, $\forall a \in A$, $a \sqsubseteq_V (\bigsqcup_V A)$. $\bot_V$ is the least element in $\mathcal{G}_V$ with respect to $\sqsubseteq_V$. We assume that γ_V is strict, that is to say, $\gamma_V(\bot_V) = \emptyset$. Then, our main abstract domain $(\mathcal{C}^\sharp, \sqsubseteq^\sharp, \bigsqcup^\sharp, \bot^\sharp)$ is the set of functions mapping each syntactic component P of $\mathcal{S}$ to an element of $\mathcal{G}_{fn(P)}$. The domain structure ($\sqsubseteq^\sharp$, $\bigsqcup^\sharp$ and $\bot^\sharp$) is defined pointwise. The abstract domain $\mathcal{C}^\sharp$ is related to $\wp(\mathcal{C})$ by the concretization function γ that maps each abstract property $f \in \mathcal{C}^\sharp$ to the set of configurations $C \in \mathcal{C}$ such that $\forall (p, id, E) \in C,\ (id, E) \in \gamma_{fn(p)}(f(p))$.

During a communication or resource fetching step, we have to describe the relations among the markers and the environments of two threads: a message receiver and a message sender. For that purpose, we also assume that we are given a family $(\mathcal{G}_{V_?}^{V_!})$ of abstract properties[2]. For any $V_?, V_! \in \wp(bn(\mathcal{S}))$, each property in $\mathcal{G}_{V_?}^{V_!}$ is related by a concretization function $\gamma_{V_?}^{V_!}$ to the elements of $\wp((Id \times \mathcal{E}(V_?)) \times (Id \times \mathcal{E}(V_!)))$ which satisfy this property.

Let V, $V_?$ and $V_!$ be three parts of $bn(\mathcal{S})$. We now introduce some primitives to handle elements in $\mathcal{G}_V$ and $\mathcal{G}_{V_?}^{V_!}$, and to relate $\mathcal{G}_{V_?}$ and $\mathcal{G}_{V_!}$ to $\mathcal{G}_{V_?}^{V_!}$:

- *initial environment abstraction:* $\varepsilon_\emptyset \in \mathcal{G}_\emptyset$ satisfies $\{(\varepsilon, \emptyset)\} \subseteq \gamma_\emptyset(\varepsilon_\emptyset)$;
- *abstract restriction:* $\forall x \in bn(\mathcal{S}) \setminus V,\ \nu_x : \mathcal{G}_V \to \mathcal{G}_{V \cup \{x\}}$ satisfies:
  $$\left\{(id, E) \in Id \times \mathcal{E}(V \cup \{x\}) \;\middle|\; \begin{array}{l}(id, E_{|V}) \in \gamma_V(A), \\ E(x) = (x, id)\end{array}\right\} \subseteq \gamma_{V \cup \{x\}}(\nu_x(A));$$
- *abstract extension:* $\forall X \subseteq bn(\mathcal{S}) \setminus V,\ new_X^\top : \mathcal{G}_V \to \mathcal{G}_{V \cup X}$ satisfies:
  $\{(id, E) \in Id \times \mathcal{E}(V \cup X) \mid (id, E_{|V}) \in \gamma_V(A)\} \subseteq \gamma_{V \cup X}(new_X^\top(A))$;
- *abstract garbage collection:* $\forall X \subseteq V,\ gc_X : \mathcal{G}_V \to \mathcal{G}_X$ satisfies:
  $\{(id, E_{|X}) \in Id \times \mathcal{E}(X) \mid (id, E) \in \gamma_V(A)\} \subseteq \gamma_X(gc_X(A))$;
- *abstract matching:* $match : ((V \times \{=; \neq\} \times V) \times \mathcal{G}_V) \to \mathcal{G}_V$ satisfies:
  $\{(id, E) \in Id \times \mathcal{E}(V) \mid (id, E) \in \gamma_V(A),\ E(x) \diamond E(y)\} \subseteq \gamma_V(match(x \diamond y, A))$.
- *abstract product:* $\bullet : \mathcal{G}_{V_?} \times \mathcal{G}_{V_!} \to \mathcal{G}_{V_?}^{V_!}$ satisfies:
  $\gamma_{V_?}(A_?) \times \gamma_{V_!}(A_!) \subseteq \gamma_{V_?}^{V_!}(A_? \bullet A_!)$;
- *abstract projections:* $fst^\sharp : \mathcal{G}_{V_?}^{V_!} \to \mathcal{G}_{V_?}$ and $snd^\sharp : \mathcal{G}_{V_?}^{V_!} \to V_!$ satisfy:
  $fst\left(\gamma_{V_?}^{V_!}(A)\right) \subseteq \gamma_{V_?}(fst^\sharp(A))$ and $snd\left(\gamma_{V_?}^{V_!}(A)\right) \subseteq \gamma_{V_!}(snd^\sharp(A))$;
- *abstract synchronization:* $sync : \wp(V_? \times V_!) \times \mathcal{G}_{V_?}^{V_!} \to \mathcal{G}_{V_?}^{V_!}$ satisfies:
  $\{((id_?, E_?), (id_!, E_!)) \in \gamma_{V_?}^{V_!}(A) \mid \forall (a, b) \in S,\ E_?(a) = E_!(b)\} \subseteq \gamma_{V_?}^{V_!}(sync(S, A))$;

[2] The abstract domain $\mathcal{G}_{V_?}^{V_!}$ is not assumed to be a pre-order, because it is only used to make intermediary calculi, and not to make iterations.

– *abstract marker allocation:* $fetch : \mathcal{L}^2 \times \mathcal{G}_{V_?}^{V_!} \to \mathcal{G}_{V_?}^{V_!}$ satisfies:
$$\left\{((id_*, E_?), (id_!, E_!)) \;\middle|\; \begin{array}{l} ((id_?, E_?), (id_!, E_!)) \in \gamma_{V_?}^{V_!}(A) \\ id_* = N((i,j), id_?, id_!) \end{array}\right\} \subseteq \gamma_{V_?}^{V_!}(fetch((i,j),A)).$$

The abstract semantics is then given by an initial abstract element $\mathcal{C}_0^\sharp(\mathcal{S}) \in \mathcal{C}^\sharp$ and an abstract transition relation $\longrightarrow_\sharp$ in Fig. 2. Their definitions use the following abstract extraction function:

$$\begin{aligned}
\beta^\sharp((\nu\ n)P, A) &= \beta^\sharp\,(P, \nu_n(A)) \\
\beta^\sharp(\mathbf{0}, A) &= \bot^\sharp \\
\beta^\sharp(P + Q, A) &= \bigsqcup\nolimits^\sharp\{\beta^\sharp(P, A); \beta^\sharp(Q, A)\} \\
\beta^\sharp(P \mid Q, A) &= \bigsqcup\nolimits^\sharp\{\beta^\sharp(P, A); \beta^\sharp(Q, A)\} \\
\beta^\sharp(action.P, A) &= \{action.P \mapsto gc_{fn(action.P)}(A)\} \\
\beta^\sharp([x \diamond^i y]P, A) &= \{[x \diamond^i y]P \mapsto gc_{fn([x\diamond^i y]P)}(A)\}
\end{aligned}$$

$$\mathcal{C}_0^\sharp(\mathcal{S}) = \beta^\sharp(\mathcal{S}, \varepsilon_\emptyset)$$

(a) Abstract initial configuration.

$$\frac{\begin{cases} \lambda = y?^i[\overline{y}]P,\ \mu = x!^j[\overline{x}]Q, \\ E_s = syn_x^y\begin{pmatrix} C^\sharp(\lambda) \leftarrow C^\sharp(\mu) \\ \overline{y} \quad \leftarrow \quad \overline{x} \end{pmatrix} \end{cases}}{C^\sharp \longrightarrow_\sharp \bigsqcup\nolimits^\sharp\{C^\sharp; \beta^\sharp(P, fst^\sharp(E_s)); \beta^\sharp(Q, snd^\sharp(E_s))\}}$$

$$\frac{\begin{cases} \lambda = *y?^i[\overline{y}]P,\ \mu = x!^j[\overline{x}]Q, \\ E_s = fetch\left((i,j), \left(syn_x^y\begin{pmatrix} C^\sharp(\lambda) \leftarrow C^\sharp(\mu) \\ \overline{y} \quad \leftarrow \quad \overline{x} \end{pmatrix}\right)\right) \end{cases}}{C^\sharp \longrightarrow_\sharp \bigsqcup\nolimits^\sharp\{C^\sharp; \beta^\sharp(P, fst^\sharp(E_s)); \beta^\sharp(Q, snd^\sharp(E_s))\}}$$

$$\frac{\diamond \in \{=; \neq\},\ \lambda = [x \diamond^i y]P}{C^\sharp \longrightarrow_\sharp \bigsqcup\nolimits^\sharp\{C^\sharp; \beta^\sharp(P, match(x \diamond y, C^\sharp))\}}$$

(b) Abstract transition system.

Fig. 2. Abstract semantics.

Furthermore, abstract communication and resource fetching require the following synchronization function, which merges the abstract environments of two communicating syntactic components, and mimics the communication of a part of the sender environment.

$$syn_x^y\begin{pmatrix} A_? \leftarrow A_! \\ \overline{y} \ \leftarrow \ \overline{x} \end{pmatrix} = sync\left(\{(y;x)\} \cup \left(\bigcup\{(y_i;x_i)\}\right), (new^\top_{\{y_1;\ldots;y_n\}}(A_?)) \bullet A_!\right)$$

Theorem 1. $C_0^\sharp$ *and* $\longrightarrow_\sharp$ *satisfy the following soundness property:*

1. $\mathcal{C}_0(\mathcal{S}) \subseteq \gamma(\mathcal{C}_0^\sharp(\mathcal{S}))$;
2. $\forall C^\sharp \in \mathcal{C}^\sharp,\ \forall C \in \gamma(C^\sharp),\ \forall \overline{C} \in \mathcal{C},\ C \longrightarrow \overline{C} \Longrightarrow \exists \overline{C}^\sharp \in \mathcal{C}^\sharp, \begin{cases} C^\sharp \longrightarrow_\sharp \overline{C}^\sharp \\ \overline{C} \in \gamma(\overline{C}^\sharp). \end{cases}$

So, the abstract counterpart $\mathbb{F}^\sharp$ to $\mathbb{F}$ defined by :

$$\mathbb{F}^\sharp(C^\sharp) = \bigsqcup^\sharp \left(\{\mathcal{C}_0^\sharp(\mathcal{S})\} \cup \left\{\overline{C}^\sharp \mid C^\sharp \longrightarrow_\sharp \overline{C}^\sharp\right\}\right)$$

satisfies the soundness condition $\forall C^\sharp \in \mathcal{C}^\sharp, \mathbb{F} \circ \gamma(C^\sharp) \subseteq \gamma \circ \mathbb{F}^\sharp(C^\sharp)$. Using Kleene's theorem, we obtain the soundness of our analysis:

Theorem 2. $\mathit{lfp}_\emptyset \mathbb{F} \subseteq \bigcup_{n \in \mathbb{N}} \left[\gamma \circ \mathbb{F}^{\sharp^n}\right] (\mathcal{C}_0^\sharp(\mathcal{S}))$.

A widening operator [6] may be then used to compute a sound and decidable approximation of the abstract semantics.

5 Analyses

We propose in this section five abstract domains of properties. The first two domains describe properties of interest which can directly be understood by the user. The last three domains represent complex properties which are used to complete the two first domains by reduction.

5.1 Dependencies Among Agent Names

5.1.1 Abstract Domain of Equality and Disequality Relations

We introduce an abstract domain for describing equality and disequality relations among a finite set of variables. Let $\mathcal{V}$ be a set of variables, we introduce for all finite subsets of $V \subseteq \mathcal{V}$ the abstract domain T_V of all non-oriented graphs (G, R) such that G is a partition of V. Given a finite subset V of $\mathcal{V}$ and $X = (G, R) \in T_V$, we introduce two binary relations over V as follows:

- $a =_X b \overset{\Delta}{\Longleftrightarrow} \exists \mathcal{X} \in G, \{a, b\} \subseteq \mathcal{X}$,
- $a \neq_X b \overset{\Delta}{\Longleftrightarrow} \exists \mathcal{X} \in G,\ \exists \mathcal{Y} \in G,\ a \in \mathcal{X},\ b \in \mathcal{Y}, (\mathcal{X}, \mathcal{Y}) \in R$.

The domain T_V is partially ordered by the $\leqslant_V$ relation defined as follows:

$$\forall X, Y \in T_V,\ X \leqslant_V Y \overset{\Delta}{\Longleftrightarrow} \begin{cases} \forall x, y \in V,\ x =_Y y \Rightarrow x =_X y \\ \forall x, y \in V,\ x \neq_Y y \Rightarrow x \neq_X y. \end{cases}$$

For every set I, each element A in T_V is related to the set of functions $\gamma_V^I(A)$, defined as follows:

$$\gamma_V^I(A) = \left\{ f \in \mathcal{F}(V, I) \;\middle|\; \forall x,\ y \in V, \begin{cases} x =_A y \implies f(x) = f(y) \\ x \neq_A y \implies f(x) \neq f(y) \end{cases} \right\}.$$

We also define some primitives over our abstract domain as follows:
let (G_X, R_X) be an element in T_V,

- given $(G_Y, R_Y) \in T_V$, $(G_X, R_X) \Cup (G_Y, R_Y) = (G', R') \in T_V$ where
  - $G' = \{x \cap y \mid x \in G_X,\ y \in G_Y,\ x \cap y \neq \emptyset\}$
  - $R' = \{(x_1 \cap y_1, x_2 \cap y_2) \in (G')^2 \mid (x_1, x_2) \in R_X \textit{ and } (y_1, y_2) \in R_Y\}$
- given $V' \subseteq \mathcal{V} \setminus V$, and $(G_Y, A_Y) \in T_{V'}$, $(G_X, A_X) \Cap (G_Y, A_Y) \in T_{V \cup V'}$, and $((G_X, A_X) \Cap (G_Y, A_Y)) = (G_X \cup G_Y, A_X \cup A_Y)$;
- given $V' \subseteq V$, $proj_{V'}(G_X, R_X) = (G', R') \in T_{V'}$ where
  - $G' = \{A \cap V' \mid A \in G_X,\ A \cap V' \neq \emptyset\}$,
  - $R' = \{(A \cap V', A' \cap V') \in (G')^2 \mid (A, A') \in R\}$;
- given $x \in bn(\mathcal{S}) \setminus V$, $fresh_x(G_X, R_X) \in T_{V \cup \{x\}}$ and $fresh_x(G_X, R_X) = (G_X \cup \{\{x\}\}, R_X \cup (\{\{x\}\} \times G_X) \cup (G_X \times \{\{x\}\}))$;
- given $V' \subseteq bn(\mathcal{S}) \setminus V$, $new_{V'}(G_X, R_X) \in T_{V \cup V'}$ and $new_{V'}(G_X, R_X) = (G_X \cup \{\{v'\} \mid v' \in V'\}, R_X) \in T_{V \cup V'}$;
- given $x, y \in V$, and $\diamond \in \{=; \neq\}$, $set(x \diamond y, (G_X, R_X)) \in T_V$ and:
  - $set(x = y, (G_X, R_X)) = ((G_X)_{/\sim}, (R_X)_{/\sim})$, if not $x \neq_{(G_X, R_X)} y$, where $\forall X, Y \in G_X,\ X \sim Y \overset{\Delta}{\Longleftrightarrow} X = Y$ or $(x, y) \in (X \times Y) \cup (Y \times X)$,
  - $set(x \neq y, (G_X, R_X)) = (G_X, R_X \cup \{(A, B); (B, A)\})$, if not $x =_{(G_X, R_X)} y$, where $A, B \in G_X$ such that $x \in A$ and $y \in B$,
  - $set(x \diamond y, (G_X, R_X))$ is undefined, otherwise.

The union $\Cup$ gives the lowest upper bound of two properties; the intersection $\Cap$ gathers the description of two disjoint sets of variables; the projection *proj* restricts the set of the variables; the operator *fresh* adds a new variable the value of which is assumed to be distinct from the value of the other variables; the operator *new* adds a set of variables without any assumption about their values; the operator *set* simply adds a new constraint.

Then, we lift T_V by adding an extra element $\bot_V$. Its concretization $\gamma_V^I(\bot_V)$ is the empty set of functions. Abstract union is lifted by $A \Cup \bot_V = \bot_V \Cup A = A$, abstract intersection by $A \Cap \bot_{V'} = \bot_V \Cap A = \bot_{V \cup V'}$. Other primitives are defined to be strict, that is to say:

$$fresh_{\_}(\bot_V) = proj_{\_}(\bot_V) = set(\_, \bot_V) = \bot_V$$

Furthermore, undefined images in *set* are identified with $\bot_V$.

5.1.2 Equality and Disequality Relations Among Channel Names

We first abstract the equality and the disequality relations between the channel names of each agent. We set, for that purpose, $\mathcal{V} = bn(\mathcal{S})$ and we introduce, for each $V \subseteq bn(\mathcal{S})$, the abstract domain $\mathcal{G}_V = T_V \cup \{\bot_V\}$. $\mathcal{G}_V$ is related to $\wp(Id \times \mathcal{E}(V))$ by the following concretization function:

$$\gamma_V(A) = \{(id, E) \mid E \in \gamma_V^{(bn(\mathcal{S}) \times Id)}(A)\},\ \text{when } A \neq \bot_V.$$

The same way, we describe relations between the names of two agents: we introduce $\mathcal{V}' = \{x_i \mid x \in bn(\mathcal{S}),\ i \in \{?, !\}\}$, and we define for each subset pair $(V_?, V_!)$ of $bn(\mathcal{S})^2$, $\mathcal{G}_{V_?}^{V_!}$ as being the set $T_{V'} \cup \{\perp_{V'}\}$ where $V' = \{x_i \mid i \in \{?, !\},\ x \in V_i\}$.

We now give primitive definitions: we set $\varepsilon_\emptyset = (\emptyset, \emptyset)$; we set $\nu x = \mathit{fresh}_x$ since, at its creation, a fresh name cannot have been communicated to any other variables; we define the extension $\mathit{new}_X^\top$ (resp. the garbage collection gc_X) by new_X (resp. proj_X); we set $\mathit{match}(x \diamond y, A) = \mathit{set}(x \diamond y, A)$; product $A_? \bullet A_!$ is obtained by first renaming the variable x into $x_?$ (resp. $x_!$) in $A_?$ (resp. in $A_!$) and then intersecting the two results using $\Cap$; projection $\mathit{fst}^\sharp(A)$ (resp. $\mathit{snd}^\sharp(A)$) is obtained by first using the projection $\mathit{proj}_{V_?}$ (resp. $\mathit{proj}_{V_!}$) and then renaming the variable $x_?$ (resp. $x_!$) into x; synchronization is defined by $\mathit{sync}(\{x = y\} \cup X, A) = \mathit{sync}(X, \mathit{set}(x = y, A))$ and $\mathit{sync}(\emptyset, A) = A$[3]; we set $\mathit{fetch}((i,j), A) = A$ since the allocation of new markers does not change the marker of channel names.

5.1.3 Equality and Disequality Relations Among Markers

We then abstract the equality and the disequality relations between the marker of an agent and the markers of its names. We set $\mathcal{V} = bn(\mathcal{S}) \uplus \{\text{agent}\}$[4], and we introduce, for each $V \subseteq bn(\mathcal{S})$, the abstract domain $\mathcal{G}_V = T_{V'} \cup \{\perp_{V'}\}$ where $V' = V \cup \{\text{agent}\}$. $\mathcal{G}_V$ is related to $\wp(\mathit{Id} \times \mathcal{E}(V))$ by the following concretization function:

$$\gamma_V(A) = \left\{ (id, E) \;\middle|\; \left(\begin{cases} V \cup \{\text{agent}\} & \to \mathit{Id} \\ v \in V & \mapsto \text{snd}(E(v)) \\ \text{agent} & \mapsto id \end{cases} \right) \in \gamma^{\mathit{Id}}_{V \cup \{\text{agent}\}}(A) \right\},$$

when $A \neq \perp_V$.

The same way, we describe relations between the markers of two agents: we introduce $\mathcal{V}' = \{x_i \mid x \in bn(\mathcal{S}) \uplus \{\text{agent}\},\ i \in \{?, !\}\}$, and define, for each subset pair $(V_?, V_!)$ of $bn(\mathcal{S})^2$, $\mathcal{G}_{V_?}^{V_!}$ as being the set $T_{V'} \cup \{\perp_{V'}\}$ where $V' = \{x_i \mid i \in \{?, !\},\ x \in V_i \uplus \{\text{agent}\}\}$.

Primitives are all defined as in the previous domain, except for the primitives $\varepsilon_\emptyset$, ν, *match* and *fetch*: we set $\varepsilon_\emptyset = (\{\text{agent}\}, \emptyset)$ since it describes an agent with an empty environment; we set $\nu x(A) = \mathit{match}(\text{agent} = x, \mathit{new}_{\{x\}}(A))$ because the marker of a newly created name is the marker of the thread which has declared it; we define $\mathit{match}(x = y, A) = \mathit{set}(x = y, A)$ and $\mathit{match}(x \neq y, A) = A$, since two different names do not necessarily have distinct markers; we set $\mathit{fetch}((i,j), A) = \mathit{fresh}_{\text{agent}_?}(\mathit{proj}_{\mathcal{V}' \setminus \{\text{agent}_?\}}(A))$ since, when duplicating a resource, the marker is fresh and distinct from any other existing marker.

5.2 Marker Analysis

We aim at describing the markers associated with an agent and its channel names. For the sake of simplicity, we use Prop. 1 and approximate every tree

[3] This primitive is well-define due to an associativity criterion.

[4] $A \uplus B$ denotes the disjoint union of A and B.

marker id by the word $\phi_1(id)$ of Σ^* written along its right comb. Then, we want to compute, for each agent P, an approximation of the set $Int(P)$, defined as follows:

$$\left\{ \left(id, \begin{cases} fn(P) & \to (\Sigma \cup bn(\mathcal{S}))^* \\ y & \to (a_y.\phi_1(b_y)) \end{cases} \right) \;\middle|\; \begin{array}{l} \exists C \in C(\mathcal{S}),\ \exists id \in Id,\ \exists E \in \mathcal{E}(fn(P)), \\ \begin{cases} (P, id, E) \in C \\ \forall y \in fn(P),\ E(y) = (a_y, b_y) \end{cases} \end{array} \right\}.$$

We will first describe the general shape of markers, and then infer some relational algebraic properties on them. By reduction, we will use this information to synthesize equality and disequality relations between channel names and between markers.

5.2.1 Shape Analysis

Shape analysis consists in distinctly abstracting, for each agent P, the set of markers which can be associated to an instance of P and to each of its channel name. Our abstraction is built upon the lattice Reg of regular languages over the alphabet $(bn(\mathcal{S}) \cup \Sigma)$. For each $V \subseteq bn(\mathcal{S})$, we define $\mathcal{G}_V = Reg \times \mathcal{F}(V, Reg)$. For all (A, f) in $\mathcal{G}_V$, $\gamma_V(A, f) \subseteq Id \times \mathcal{E}(V)$ is the set of all the elements (id, E) which satisfy: $\phi_1(id) \in A$ and $\forall x \in V, y.\phi_1(id_x) \in f(x)$ where $(y, id_x) = E(x)$.

Abstract union is defined point-wisely. Since the domain is not relational, we define, for each $V_?$, $V_!$ in $bn(\mathcal{S})$, the domain $\mathcal{G}_{V_?}^{V_!}$ as the Cartesian product $\mathcal{G}_{V_?} \times \mathcal{G}_{V_!}$, and the primitives $\bullet$, $first^\sharp$, $second^\sharp$ as the canonical pair construction and projection functions. Other abstract primitives are defined as follows:

- $\varepsilon_\emptyset = (\varepsilon, \emptyset)$;
- $\nu x(id^\sharp, f^\sharp) = \left(id^\sharp, f^\sharp[x \mapsto x.id^\sharp]\right)$;
- $new_X^\top(id^\sharp, f^\sharp) = \left(id^\sharp, f^\sharp[x \mapsto bn(\mathcal{S}).\Sigma^*,\ \forall x \in X]\right)$;
- $gc_X(id^\sharp, f^\sharp) = \left(id^\sharp, f^\sharp_{|X}\right)$;
- $match(x = y, (id^\sharp, f)) = \begin{cases} [\_ \to \emptyset] & \text{if } f(x) \cap f(y) = \emptyset \\ f[x, y \mapsto f(x) \cap f(y)] & \textit{otherwise}; \end{cases}$
- $match(x \neq y, A) = A$;
- $sync(\{x = y\} \cup X, A) = sync(X, match(x = y, A))$ and $sync(\emptyset, A) = A$;
- $fetch((i, j), ((id_?, f_?), (id_!, f_!))) = ((id_!.(i, j), f_?), (id_!, f_!))$.

There can be infinite increasing sequences in Reg, so we need a widening operator to ensure the convergence of our analysis in a finite amount of time.

5.2.2 Global Numerical Abstraction

Numerical abstraction captures the relations between the markers which are associated to an agent and to its channel names. This abstraction is built upon the lattice of the affine relations among a set of numerical variables [9]. Each word is first approximated by its Parikh vector [10], then we abstract the relations between occurrence numbers of letters in markers. For each $V \subseteq bn(\mathcal{S})$, we denote by $\mathcal{X}_V$ the set of variables $\{p^\lambda \mid \lambda \in \Sigma\} \cup \{c^{(\lambda,v)} \mid \lambda \in \Sigma \cup bn(\mathcal{S}),\ v \in V\}$. The variable p^λ is used to describe the number of occurrences of λ in the right comb of the markers of the instances of the agent p, while $c^{(\lambda,v)}$ is used to count the number of occurrences of λ in the marker associated to the syntactic name v when $\lambda \in \Sigma$, or in determining whether v is bound to a channel created by an instance of the restriction $(\nu\ \lambda)$ when $\lambda \in bn(\mathcal{S})$. $\mathcal{G}_V$ is the set of affine equality relations among the variables of $\mathcal{X}_V$. For all $\mathcal{K}$ in $\mathcal{G}_V$, $\gamma_V(\mathcal{K}) \subseteq Id \times \mathcal{E}(V)$ is the set of the elements (id, E) such that the assignment

$$\{p^\lambda \rightarrow |id|_\lambda,\ c^{(\lambda,v)} \rightarrow |y.id_c|_\lambda\ \textit{where}\ (y, id_c) = E(c)\}$$

is a solution of $\mathcal{K}$.

In the same manner, for each $V_?, V_! \subseteq bn(\mathcal{S})$, we denote by $\mathcal{X}_{V_?}^{V_!}$ the set of variables $\{p_i^\lambda \mid \lambda \in \Sigma,\ i \in \{?, !\}\} \cup \{c_i^{(\lambda,v)} \mid \lambda \in \Sigma, i \in \{?, !\}, v \in V_i\}$. We also define $\mathcal{G}_{V_?}^{V_!}$ as the set of affine equality relations among the variables of $\mathcal{X}_{V_?}^{V_!}$. For all $\mathcal{K}$ in $\mathcal{G}_{V_?}^{V_!}$, $\gamma_{V_?}^{V_!}(\mathcal{K}) \subseteq Id \times \mathcal{E}(V)$ is the set of elements $((id_?, E_?), (id_!, E_!))$ such that the assignment:

$$\{p_i^\lambda \rightarrow |id_i|_\lambda,\ c_i^{(\lambda,v)} \rightarrow |y.id_c|_\lambda\ \textit{where}\ (y, id_c) = E_i(c)\}$$

is a solution of $\mathcal{K}$.

Most primitives can be encoded using affine operators described in [9]. Positive abstract matching and synchronization are simply defined by adding new affine constraints in the system. These constraints specify that the Parikh's vectors coordinates of the synchronized channel names are pair-wisely equal; negative abstract matching is the identity function. Abstract resource fetching $\textit{fetch}((i, j), \mathcal{K})$ is obtained using an affine projection to keep only constraints not involving variables of the form $p_?^\lambda$, then adding the constraints $p_?^\lambda = p_!^\lambda$ for all $\lambda \in \Sigma$, and last replacing each occurrence of the variable $p_?^{(i,j)}$ by the expression $p_?^{(i,j)} - 1$.

Example 3. In our example, the analysis detects that:

- in each agent labeled **1**, the variable ***next*** is linked to a name created by the (ν ***next***) restriction, while the variable ***last*** is linked to a name, either created by an instance of the (ν ***next***) restriction, or by an instance of the (ν **first**) restriction. We also detect that, in the case where the variable ***last*** is linked to a name created by the (ν ***next***) restriction, this variable is linked to the name created by the previous recursive instance of the one which has created the name communicated to the variable ***next***;
- in each agent labeled **4**, the variable **first** is linked to a name created by the (ν **first**) restriction, while the variable ***last*** is linked to a name either created by an instance of the (ν ***right***) restriction or by an instance of the (ν ***first***) restriction.

These properties are deduced from the following invariants:

$$\begin{cases} f(1) \text{ satisfies } \begin{cases} c^{next,next} = 1 \\ c^{\text{first},last} + c^{next,last} = 1 \\ c^{(0,2),next} = c^{(0,2),last} + c^{\text{next},last} \end{cases} \\ f(4) \text{ satisfies } \begin{cases} c^{next,last} + c^{\text{first},last} = 1 \\ c^{\text{first},\text{first}} = 1 \end{cases} \end{cases}$$

where f denotes the result of the analysis.

Nevertheless, our abstract domain is not expressive enough to merge these two environments, and detects no insightful information for the agent labeled **6**. That is why we introduce a partitioned domain. □

5.2.3 Partitioned Numerical Abstraction

We propose to partition the set of the interactions between channels and agents in order to get more accurate results. To avoid complexity explosion, we do not globally abstract environments, we only compare pair-wisely the right comb of the markers. Let ψ[5] be a linear form defined on $\mathbb{Q}^{\Sigma}$. We introduce the set $\mathcal{G}_V$ of functions which map $(V \times bn(\mathcal{S})) \uplus ((V \times bn(\mathcal{S}))^2)$ onto the set of affine subspaces of $\mathbb{Q}^2$. For all $f \in \mathcal{G}_V$, $\gamma_V(f) \subseteq Id \times \mathcal{E}(V)$ is the set of the elements (id, E) such that:

- $\forall x \in V$, such that $E(x) = (c_x, id_x)$.
  $(\psi([\lambda \to |\phi_1(id)|_\lambda]), \psi([\lambda \to |\phi_1(id_x)|_\lambda])) \in f(x, c_x)$;
- $\forall x \in V, \forall y \in V$, such that $E(x) = (c_x, id_x)$ and $E(y) = (c_y, id_y)$,
  $(\psi([\lambda \to |\phi_1(id_x)|_\lambda]), \psi([\lambda \to |\phi_1(id_y)|_\lambda])) \in f((x, c_x), (y, c_y))$.

Because of the precision of the partitioning, we cannot afford much calculi in this domain. This domain will only be used to propagate information we got from the global numerical abstraction.

We now give primitive definitions. For $V_?, V_! \subseteq bn(\mathcal{S})$, we define $\mathcal{G}_{V_?}^{V_!}$ as the Cartesian product $\mathcal{G}_{V_?} \times \mathcal{G}_{V_!}$. The concretization $\gamma_{V_?}^{V_!}$ is defined pair-wise. Pair construction and projection functions are the canonical ones. Other abstract primitives are defined as follows:

- abstract union is defined by applying the affine union component-wise;
- we define $\nu_x(f)$ to be the following element:

$$f[(x, x) \to \{(n, n) \mid n \in (Q)\}, ((x, x), (\_, \_)) \to \mathbb{Q}^2, ((\_, \_), (x, x)) \to \mathbb{Q}^2];$$

- $match(x = y, f) = f \begin{bmatrix} ((x, c), (y, c)) \to f((x, y), (c, y)) \cap \{(n, n) \mid n \in \mathbb{Q}\} \\ ((x, c), (y, d)) \to \emptyset, \ if\ c \neq d \end{bmatrix}$;
- $match(x \neq y, f) = f$;

[5] This abstraction must be done with several linear forms chosen according to a pre-analysis and Thm. 3.

– $gc_X(f) = f_{|\{((x,\_),(y,\_))\ |\ x,y \in X\}}$;

– $new^{\top}_X(f) = f\begin{bmatrix} x, c \to \mathbb{Q}^2, x \in X; \\ (x,c),(\_,\_) \to \mathbb{Q}^2, x \in X; \\ (\_,\_),(x,c) \to \mathbb{Q}^2, x \in X \end{bmatrix}$;

– $sync(\{y_k = x_k \mid k \in K\}, (f,g)) = (f', g)$ where
$f' = f[((y_i,c),(y_j,d)) \to f((y_i,c),(y_j,d)) \cap g((x_i,c),(x_j,d)), \forall i,j \in K]$;

– $fetch((i,j),(f,g)) = (f[(y,c) \to \mathbb{Q}^2], g)$.

We shall notice that no information is calculated by abstract name creation and abstract marker allocation. Nevertheless, during synchronization, the description of the communicated names is copied to the description of the receiver environment. A complete reduction of properties would lead to a time complexity explosion. We use a partial reduction. On the first hand, we use thread markers as pivots and replace each abstract element f with the following element:

$$f\left[((x,c),(y,d)) \to f((x,c),(y,d)) \cap \left\{ (x,y) \;\middle|\; \exists z, \begin{cases} (z,y) \in f(y,d), \\ (z,x) \in f(x,c) \end{cases} \right\}\right],$$

and on the other hand, we always perform reductions between global numerical abstraction and partitioned numerical abstraction: global numerical analysis is used to collect all the information, which is then projected onto each case of the partition.

Example 4. For Ψ, we choose the linear form which maps each vector to the sum of its components. Our analysis succeeds in proving that the second pattern matching, in the example, is not satisfiable. Along the abstract iteration, the analyzer proves that in agent **6**, the names linked to the variables **x** and **y** have been respectively declared:

1. by the action (ν **first**) of a thread with a **0** marker length and the action (ν **first**) of a thread with a **0** marker length;
2. or by the action (ν **first**) of a thread with a **0** marker length and by the action (ν ***next***) with a **1** marker length;
3. or by the action (ν ***next***) of a thread t_1 and by the action (ν ***next***) of a thread t_2 such that the length of the marker of t_2 is equal to the length of the marker of t_1 plus **1**;
4. or by the action (ν ***next***) of a thread the length of the marker of which was arbitrary and by the action (ν **first**) of a thread the length of the marker of which was **0**.

Then it detects that the matching pattern ([**x** = **y**]) can only be satisfied for the case 1 and it discovers that in agent **7**, all the syntactic channel names **x**, **y** and **first** are bound to a channel created by the action (ν **first**) the thread marker of which is ε and concludes that the second pattern matching ([**x** $\neq$ **first**]) cannot be satisfied. □

6 Reduced Product

We run our analysis with a reduced product of these five domains. There are several kinds of reductions: we use the information about equality of channel names and equality of markers to refine marker analysis; conversely, we use the information obtained in marker analysis to infer information about disequalities of channel names and disequalities of markers; we also make reductions between the global and partitioned numerical abstractions; we use the results obtained in marker analysis to prove equality relations between name markers. For the sake of the brevity, we only describe this last kind of reduction, which is the most difficult one. Marker analysis may discover that two markers are recognized by the same automaton $\mathcal{A}$ and have the same Parikh's vector, but these two conditions do not ensure that these two markers are the same. We give in Thm. 3 a decidable sufficient condition on the automaton $\mathcal{A}$ to ensure the equality of such markers.

Let Σ be a finite alphabet, ϕ be a linear function from the vector space $\mathbb{R}^{\Sigma}$ into the vector space $\mathbb{R}^m$, and $\mathcal{A}$ be an automaton $(Q, \rightarrow, i, f)$ such that the set Q is finite, the relation $\rightarrow$ is a part of $Q \times \Sigma \times Q$ and i and f are parts of Q.

Definition 1. *We define the set $Path(\mathcal{A})$ of acyclic derivation sequences in $\mathcal{A}$ as the set of sequences $q_0 \overset{\lambda_1}{\rightarrow} .... \overset{\lambda_n}{\rightarrow} q_n$ such that $q_0 \in i$, $q_n \in f$, for all $i, j \in [\![0; n]\!]$, $i \neq j \implies q_i \neq q_j$, and for all $i \in [\![1; n]\!], (q_{i-1}, \lambda_i, q_i) \in \rightarrow$.*

Definition 2. *Let q be a particular state in Q. We define the set $Cycle(\mathcal{A}, q)$ of elementary cycles of $\mathcal{A}$ stemming from the state q, as the set of sequences $q = q_0 \overset{\lambda_0}{\rightarrow} q_1 ... q_n \overset{\lambda_n}{\rightarrow} q_{n+1} = q$ such that for all $i \in [\![1; n]\!]$, $q_i \neq q$, and for all $i \in [\![0; n]\!]$, $(q_i, \lambda_i, q_{i+1}) \in \rightarrow$.*

Definition 3. *Let $q_0 \overset{\lambda_1}{\rightarrow} .... \overset{\lambda_n}{\rightarrow} q_n$ be a sequence in $\mathcal{A}$, we define its affine description $\mathcal{P}(q_0 \overset{\lambda_1}{\rightarrow} .... \overset{\lambda_n}{\rightarrow} q_n)$ as the vector $\phi([\lambda \rightarrow |\lambda_1 ... \lambda_n|_\lambda])$.*

Definition 4. *Let $a = q_0 \overset{\lambda_1}{\rightarrow} .... \overset{\lambda_n}{\rightarrow} q_n$ be an acyclic derivation sequence in $Path(\mathcal{A})$, we define the family $\mathcal{F}(a)$ of affine descriptions of the cycle of the derivation a as the family $(\mathcal{P}(c))_{c \in \bigcup \{ Cycle(q_i) \mid i \in [\![0;n]\!] \}}$.*

Theorem 3. *If*

1. *for all $q \in Q$, $Card(Cycle(\mathcal{A}, q)) \leqslant 1$,*
2. *for all $a \in Path(\mathcal{A})$, $\mathcal{F}(a)$ is linearly independent in $\mathcal{R}^m$,*
3. *for all distinct acyclic derivations $a, a' \in Path(\mathcal{A})$, the two affine sets $\mathcal{P}(a) + Vect(\mathcal{F}(a))$ and $\mathcal{P}(a') + Vect(\mathcal{F}(a'))$ are disjoint,*

then

$$\forall u, v \in \Sigma^*, \ [u, v \text{ recognized by } \mathcal{A} \text{ and } \phi[\lambda \rightarrow |u|_\lambda] = \phi[\lambda \rightarrow |v|_\lambda]] \implies u = v.$$

Roughtly speaking, the first condition ensures that the automaton $\mathcal{A}$ contains no embedded cycle. Then, from the description of a word Parikh's vector, we can deduce which main acyclic derivation is to be used to recognize this word (third condition), and how many times each cycles are to be used (second condition).

7 Conclusion

We have proposed a new parametric framework for automatically inferring the description of the dependency between the channel names used by the agents of a mobile system. We claim that this framework is very generic since [2, 12, 7, 3] may all be seen as a particular use of it.

We have proposed several abstract domains to deal with this framework. They allowed us to prove some properties which cannot be obtained with [2, 12, 7, 3].

Acknowledgments. We deeply thank anonymous referees for their significant comments on an early version of this paper. We wish also to thank Patrick and Radhia Cousot, Arnaud Venet, Antoine Min, Francesco Logozzo and Xavier Rival for their comments and discussions.

References

1. G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.
2. C. Bodei, P. Degano, F. Nielson, and H.R Nielson. Control flow analysis for the π-calculus. In *Proc. CONCUR'98*, LNCS. Springer-Verlag, 1998.
3. L. Cardelli, G. Ghelli, and A. D. Gordon. Secrecy and group creation. In *Proc. CONCUR'00*, LNCS. Springer-Verlag, 2000.
4. 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.
5. P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of logic and computation*, 2(4):511–547, August 1992.
6. P. Cousot and R. Cousot. Comparing the Galois connection and widening--narrowing approaches to abstract interpretation. In *Proc. PLILP'92*, LNCS. Springer-Verlag, 1992.
7. J. Feret. Confidentiality analysis of mobile systems. In *Proc. SAS'00*, LNCS. Springer-Verlag, 2000.
8. J. Feret. Occurrence counting analysis for the π-calculus. *ENTCS*, 39.2, 2001. Workshop on GEometry and Topology in COncurrency theory, PennState, USA, August 21, 2000.
9. M. Karr. Affine relationships among variables of a program. *Acta Informatica*, pages 133–151, 1976.
10. R. J. Parikh. On context-free languages. *Journal of the ACM*, 13:570–581, 1966.
11. D. N. Turner. *The Polymorphic Pi-Calculus: Theory and Implementation*. PhD thesis, Edinburgh University, 1995.
12. A. Venet. Automatic determination of communication topologies in mobile systems. In *Proc. SAS'98*, LNCS. Springer-Verlag, 1998.