An Enhanced CFA for Security Policies *

Chiara Bodei[1] Pierpaolo Degano[1] Corrado Priami[2] Nicola Zannone[3]

[1] Dipartimento di Informatica, Università di Pisa
Via Filippo Buonarroti, 2, I-56127 Pisa, Italy – {chiara,degano}@di.unipi.it
[2] Dipartimento di Informatica e Telecomunicazioni, Università di Trento
Via Sommarive, 1438050 Povo (TN), Italy – priami@science.unitn.it
[3] Dipartimento di Informatica, Università di Verona
Ca' Vignal 2, Strada Le Grazie, 1, I-37134 Verona, Italy – zannone@students.univr.it

Abstract. We introduce a Control Flow Analysis, improving the one in [6], that statically approximates the dynamic behaviour of mobile processes, expressed in (a variant of) the π-calculus. Our analysis of a system is able to describe the behaviour of each sub-system, tracking where and between whom communications may occur. To identify each sub-system, we use a syntactic encoding of its position inside the abstract syntax tree. Furthermore, our analysis is general enough to safely approximate the behaviour of a system plugged in a larger and mainly unknown context, without explicitly analysing it. Quite a lot of possible properties fan out, among which some related to security policies.

1 Introduction

Systems of mobile processes are open systems: they are considered as part of larger and mainly unknown contexts. Their analysis is particularly challenging, because it should describe and predict the behaviour of systems and of their sub-systems, regardless of the possible hostile environment in which they run.

To abstractly represent and study systems, we use (a variant of) the π-calculus [14], which is a model of concurrent communicating processes based on name passing. Names may represent both data and channels that processes exchange. Protecting data and resources is crucial in any information management system for secrecy, for integrity and for availability. Enforcing protection requires to state which information flows are allowed and which are not, i.e. to enforce an access control policy w.r.t. principals and resources. In our framework resources are names (channels and data), while principals are sub-processes. We further assume that read and write are the only modes to use and access resources. We can then exploit the precision of the technique briefly outlined below to statically predict when processes respect a few security policies.

The analysis we present here approximates the behaviour of a system P and of its sub-systems, plugged in any environment E, without explicitly analysing E. In particular, we approximate the possible interactions of P with E, that may occur on the channels they share. More precisely, E can listen on these channels and can therefore acquire new names. Moreover E can send to P all the names it has acquired and exploit them as channels. This is reminiscent of the Dolev and Yao model [10]. A richer structure on data, e.g. that of the spi-calculus [1] does not affect deeply our approach and these extensions can be dealt with as in [7].

We use here a specific static technique, Control Flow Analysis (CFA), based on Flow Logic [15] and propose an analysis for the π-calculus that improves the one in [6]. The main idea is to exploit in its definition the notion of "logical" *addresses* of sub-processes, in the style of [9, 8]. Through them, our CFA can safely approximate the behaviour of each sub-system, tracking where and between whom communications may occur. More in detail, the result of our analysis, or *estimate*, is a triple (ρ, η, Φ).

* Work partially supported by EU-project DEGAS (IST-2001-32072) and the Progetto MIUR Metodi Formali per la Sicurezza (MEFISTO).

The first component, ρ, gives information about the set of channels to which names can be bound at run-time; the second component, η, gives information about the set of channels that can be communicated on given channels and about the *addresses* of participants in the communication. Additionally, an estimate establishes a super-set Φ of the knowledge of the environment. This component is therefore used to implicitly approximate the behaviour of E, by collecting all the names the environment initially knows, those it can acquire reading from, or more generally by communicating with P, as described above.

The analysis is carried on by taking into account the abstract syntax tree T of a process, built by considering parallel composition $|$ and nondeterministic choice $+$ as main operators. The path from the root of T – the whole process – to a node corresponding to a sub-process P we call *address* of P, in the style of the enhanced operational semantics [9, 8]. Addresses *implicitly* annotate sub-processes in a bijective manner. So, they also offer a way of determining the principals involved in a system. Consider the following three processes, where just for explanation, we make the annotation explicit:

$$P_1 = (\underbrace{a(y)}_{\|_0} | \underbrace{\overline{a}b}_{\|_1}) \qquad P_2 = (\underbrace{a(y)}_{+\!\!\!+_0} + \underbrace{\overline{a}b}_{+\!\!\!+_1}) \qquad P_3 = (a(y).\overline{a}b)$$

In our CFA, addresses are then used to selectively collect the information of interest, e.g. the values a variable y may assume at run-time. Clearly, these values may vary, depending on the different sub-processes where y occur. So, using addresses we obtain better estimates, *without explicitly* resorting to contexts.

In our example, the operational semantics of the π-calculus tells us that the variable y in P_1 can be bound to b, while in P_2 and in P_3 it cannot, unless E knows b and sends it along a. Our present analysis reflects this situation, by statically considering possible only synchronizations between sub-processes in parallel. A communication is predicted only if the receiving sub-process is in parallel with the sending sub-process; technically, if the two co-actions lie on different sides of the same $|$. Instead, the analysis proposed in [6] (safely, but imprecisely) predicts that variable y may be bound to b in all cases above, even if receiving and sending are mutually exclusive as in P_2, or reading must occur before sending b, as in P_3.

It is possible to establish the *semantic correctness* of our analysis in the form of a *subject reduction theorem* and to show that *least estimates always exist*. There is also a *constructive procedure* for obtaining the least solution, whose complexity we argue is low-polynomial.

We validate our proposal on a variety of security properties taken from the literature, in particular those based on access policies, thus showing it quite flexible and expressive. The general scheme consists in selecting a specific dynamic security property and defining a static check on estimates that implies the dynamic property, i.e. if a system passes a static security check, then all its computations do enjoy the dynamic property. The major point is that an estimate is generated *once for all*, while the properties are checked by performing different static tests on the *single* computed estimate. Since our analysis approximates the behaviour of each component of the system under consideration, we can therefore statically predict:

- which kind of actions (read or write) are possible at certain addresses,
- which principals can perform them,
- on which channels these actions may take place,
- which objects flow on which channels and
- with which partner communications take place.

The expressivity and generality sketched above make CFA a complementary approach to the more classical Type Systems (see e.g. [12, 13, 16]), where specific type systems are usually defined *ad hoc* for a particular policy. (For other static and related approaches see also [11], based on Abstract Interpretation.)

$$Tau : \tau.P \xrightarrow{\tau} P \qquad Out : \overline{a}b.P \xrightarrow{\overline{a}b} P \qquad Sel-Ein : a(y \in Y).P \xrightarrow{ab} P\{b/y\},\ b \in Y$$

$$Sum_0 : \frac{P_0 \xrightarrow{\mu} Q_0}{P_0 + P_1 \xrightarrow{\mu} Q_0 + \mathbf{0}} \qquad Sum_1 : \frac{P_1 \xrightarrow{\mu} Q_1}{P_0 + P_1 \xrightarrow{\mu} \mathbf{0} + Q_1}$$

$$Par_0 : \frac{P_0 \xrightarrow{\mu} Q_0}{P_0|P_1 \xrightarrow{\mu} Q_0|P_1}, bn(\mu) \cap fn(P_1) = \emptyset \qquad Res : \frac{P \xrightarrow{\mu} Q}{(\nu a)P \xrightarrow{\mu} (\nu a)Q}, a \notin n(\mu)$$

$$Open : \frac{P \xrightarrow{\overline{a}b} Q}{(\nu b)P \xrightarrow{\overline{a}(b)} Q}, b \neq a \qquad Close_0 : \frac{P_0 \xrightarrow{\overline{a}(b)} Q_0,\ P_1 \xrightarrow{ab} Q_1}{P_0|P_1 \xrightarrow{\tau} (\nu b)(Q_0|Q_1)}$$

$$Com_0 : \frac{P_0 \xrightarrow{\overline{a}b} Q_0,\ P_1 \xrightarrow{ab} Q_1}{P_0|P_1 \xrightarrow{\tau} Q_0|Q_1} \qquad Var : \frac{P' \equiv P \xrightarrow{\mu} Q \equiv Q'}{P' \xrightarrow{\mu} Q'}$$

Table 1. Early transition system for the π-calculus (symmetric rules are omitted).

2 The Process Calculus

Syntax In our approach, channels are interfaces and processes are interested in filtering received values. For this reason, we slightly change the standard π-calculus [14] obtaining an equally expressive variant of it.

The main difference with the standard calculus is the absence of a matching construct and the selection on the values received in communications. Values are accepted only if they are included in the selection set. In other words, inputs only succeed on values matching one of the names in the set. Furthermore, for the sake of simplicity, we refer to a version of the calculus without replication.

Processes are defined according to the following syntax, where Y is a finite set of names. We will use the standard input $x(y)$, when any value can be received on x.

$$P, Q, R ::= \mathbf{0} \mid \tau.P \mid \overline{x}y.P \mid x(y \in Y).P \mid (\nu a)P \mid P + P \mid P|P$$

Inputs and restrictions bind the names, the other names are free. The set of set of names $n(.)$, the set of free names $fn(.)$ and the one of bound names $bn(.)$ are defined in the standard way. Let Var_P be the set of the names occurring within input prefixes in a process P and Val_P be the set of the remaining names in P.

Semantics Our static approximations exploit the structure of processes, so we need to preserve it under this dynamic evolution. This is the main reason why our semantics slightly differs from the standard one.

In particular, our congruence has *no* rule for making the parallel and non-deterministic operators associative and commutative or for considering inaction as a neutral element. This way, the abstract syntax tree of a process always keeps its structure. The *structural congruence* $\equiv$ on processes is the least congruence satisfying:

- $P \equiv Q$ if P and Q are α-equivalent in the disciplined way discussed below;
- $(\nu\, x)(\nu\, y)P \equiv (\nu\, y)(\nu\, x)P$, $(\nu\, x)P \equiv_P P$ if $x \notin fn(P)$ and $(\nu\, x)(P \,|\, Q) \equiv (\nu\, x)P \,|\, Q$ if $x \notin fn(Q)$.

To simplify the definition of the control flow analysis in Section 3, we discipline the α-renaming of bound names. To do it in a simple and "implicit" way, we assume that names are "stable", i.e. that for each name $a \in \mathcal{N}$ there is a canonical representative $\lfloor a \rfloor \in \lfloor \mathcal{N} \rfloor$: two names are α-convertible only when they have the same canonical names. In this way, we statically maintain the identity of names that may be lost by freely applying α-conversions. Hereafter, we shall simply write a for $\lfloor a \rfloor$. We also assume that all the bound names are kept distinct and are disjoint from the free names.

Tab. 1 displays the rules of the operational semantics of our calculus. As written above, our semantic rules slightly differ from the standard *early* ones, in particular input and summation are treated differently. The placeholder of the input is instantiated in the axiom for selective input $(Sel - Ein)$: $a(y \in Y).P \xrightarrow{ab} P\{b/y\}$, only if $b \in Y$. For any b and y, $P\{b/y\}$ denotes the operation of substituting the free occurrences of y in the process P by b. In particular, if P is on the form $c(z \in \{\ldots, y, \ldots\}).P'$, then $P\{b/y\} = c(z \in \{\ldots, b, \ldots\}).(P'\{b/y\})$. Here we are somehow more demanding that the standard semantics that allows everything to be read from outside (with a so-called free input). Intuitively, our selective input implements a filter on the messages that a process is willing to accept, much in the way we do with our e-mail.

Our selective input is rendered in standard π-calculus as an input followed by a choice among some matchings: $x(y \in \{y_1, \ldots, y_n\}).P$ behaves as the process $x(y)([y = y_1]P \ + \ \ldots + \ [y = y_n]P)$. Finally, $(\nu a)(\overline{a}x \mid a(y \in \{x\}).P)$ encodes the matching construct $[x = y]P$.

As for the *Sum* rules, the "structure" with respect to $+$ of the process is preserved leaving a $\mathbf{0}$ in the place of the process not chosen, i.e. if $P_0 \xrightarrow{\mu} Q_0$ then $P_0 + P_1 \xrightarrow{\mu} Q_0 + \mathbf{0}$. Note that this change does not affect at all the standard meaning of the nondeterministic operator.

Our peculiar rules for summation and congruence preserve the structure of a process, and in particular its abstract syntax tree, while the computation goes on. The next paragraph will show why this is useful for our analysis.

On the Structure of Processes To approximate the behaviour of sub-processes in a process, we need a way to distinguish among them. We do it by exploiting the syntax, i.e. by considering the abstract syntax trees of processes. They are built using the binary parallel composition $|$ and the non deterministic choice $+$ as the main operators. Given a process R, the nodes of its tree (see Fig. 1) correspond to the occurrences of $|$ and $+$ in R, and its leaves are the sequential components of R (roughly, those processes whose top-level operator is a prefix). Assume now that the left (resp. right) branches of such a tree denote the left (resp. right) component of a $|$ or of a $+$ operator, and label their arcs with tag $||_0$ and $+\!\!\!+_0$ (resp. $||_1$ and $+\!\!\!+_1$). Therefore, any sub-process P of R is uniquely identified by a string $\vartheta \in \mathcal{A} = \{||_i, +\!\!\!+_i \mid i = 0, 1\}^*$. The string ϑ, called *address* of P, corresponds to the path from the root, i.e. the whole process R, to P. For example, in Fig. 1, the address of P_3 is $||_1||_1 +\!\!\!+_0$. Conversely, given

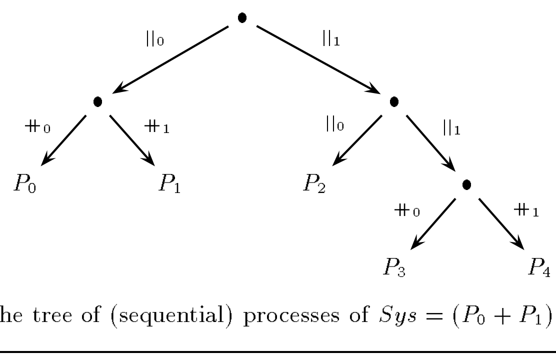

Fig. 1. The tree of (sequential) processes of $Sys = (P_0 + P_1)|(P_2|(P_3 + P_4))$.

any process R, it is possible to localize its sub-process P, if any, whose address is ϑ. This is done via the operator $@\vartheta$, defined below. In our example, e.g. $Sys@||_1||_1 +\!\!+_0 = P_3$.

The *localization operator* $@\vartheta$ is defined on processes by induction (ϵ is the empty address):

1. $P@\epsilon = P$;
2. $(P_0 \mid P_1)@||_i\vartheta = P_i@\vartheta$;
3. $(P_0 \; + \; P_1)@ +\!\!+_i\vartheta = P_i@\vartheta$;
4. $\pi.P@\vartheta = ((\nu a)P)@\vartheta = P@\vartheta$, for $\vartheta \neq \epsilon$.

The set $Addr(P) = \{\vartheta | \exists Q.P@\vartheta = Q\}$ collects all the addresses of sub-processes of P.

From a semantic point of view, a communication can take place only if the two complementary actions are *compatible*, i.e. they lay on the *same* side of a $+$, if any, and on *different* sides of the same parallel composition operator $|$. If replication would be considered as well, then all the actions inside the replicated part of a process could be considered compatible each other and compatible with the ones in the rest of the process. Formally, we have the following definition.

Definition 1. *Given a process P and two addresses $\vartheta, \vartheta' \in Addr(P)$, ϑ and ϑ' are* compatible, *written as* $\mathrm{comp}_P(\vartheta, \vartheta')$, *or symmetrically as* $\mathrm{comp}_P(\vartheta', \vartheta)$, *if and only if:*

$$\vartheta = \vartheta_0||_i\vartheta_1 \quad \textit{and} \quad \vartheta' = \vartheta_0||_{1-i}\vartheta_1' \textit{ for } i \in \{0, 1\}.$$

Back to our example in Fig. 1, P_0 and P_3, i.e. $Sys@||_0 +\!\!+_0$ and $Sys@||_1||_1 +\!\!+_0$ (for which ϑ_0 is ϵ) may communicate and so do two parallel sub-components of P_0, if any (for which ϑ_0 is $||_0 +\!\!+_0$). On the contrary, P_0 and P_1 cannot communicate: their addresses $||_0 +\!\!+_0, ||_0 +\!\!+_1$ reflect the fact that only one out of P_0 and P_1 can be active.

Finally, to preserve the association between sub-processes and their relative addresses, we could have used a suitable updating of addresses, depending on the dynamic evolution of the computation, instead of changing the semantics.

3 Control Flow Analysis

We are interested in analyzing the behaviour of a process P plugged in any environment E, without explicitly analysing E. More precisely, our implicit analysis of the environment amounts to analysing the most powerful context sharing public (i.e. free) names with P. (Technically, in process algebras like π-calculus, a process Q cannot know the restricted names of another process P, unless P did not send them to Q.)

The result of analysing a process P is a triple (ρ, η, Φ), called estimate or solution, that, roughy, establishes a super-set of the set of (abstract) values to which the program objects can be bound to. In details, the estimate components are the following:

- $\rho : \lfloor \mathcal{N} \rfloor \to \wp(Val)$ is the *abstract environment* that associates a set of names with names; more precisely, $\rho(x)$ must include the set of values that x could assume at run-time. We will use $\rho(Y)$ as a shorthand for $\bigcup\{\rho(y_i) \mid y_i \in Y\}$.
- $\eta = \langle \eta_1, \eta_2 \rangle$ is the *abstract communication structure*:
  - $\eta_1 : \mathcal{A} \to (Val \to \wp(Val))$ associates names with the values that can be sent over them; more precisely, $\eta_1(\vartheta)(a)$ must include the set of values that can be sent over the channel a in the sub-process $P@\vartheta$ (ϑ represents the address of the sub-process w.r.t P). Note that for each process P, we have that $\mathcal{A} = Addr(P)$.

$$
\begin{array}{lll}
(\rho,\eta,\Phi) \models^{\vartheta} \mathbf{0} & \text{iff } \mathit{true} \\
(\rho,\eta,\Phi) \models^{\vartheta} \tau.P & \text{iff } (\rho,\eta,\Phi) \models^{\vartheta} P \\
(\rho,\eta,\Phi) \models^{\vartheta} \overline{x}y.P & \text{iff } (\rho,\eta,\Phi) \models^{\vartheta} P \wedge \\
& \quad \forall a \in \rho(x) : \rho(y) \subseteq \eta_1(\vartheta)(a) = J \wedge \\
& \quad (a \in \Phi \wedge J \neq \emptyset) \Rightarrow \begin{cases} \eta_1(\vartheta)(a) \subseteq \Phi \\ \eta_2(\vartheta)(a) \ni \vartheta_E \end{cases} \\
(\rho,\eta,\Phi) \models^{\vartheta} x(y \in Y).P & \text{iff } (\rho,\eta,\Phi) \models^{\vartheta} P \wedge \\
& \quad \forall a \in \rho(x), \forall \vartheta' : \mathit{comp}(\vartheta,\vartheta') : \\
& \quad J = (\eta_1(\vartheta')(a) \cap \rho(Y)), J' = (\Phi \cap \rho(Y)) \\
& \quad (J \neq \emptyset) \Rightarrow \begin{cases} \eta_1(\vartheta')(a) \cap \rho(Y)) \subseteq \rho(y) \\ \eta_2(\vartheta')(a) \ni \vartheta \end{cases} \\
& \quad (a \in \Phi \wedge J' \neq \emptyset) \Rightarrow \begin{cases} (\Phi \cap \rho(Y)) \subseteq \rho(y) \\ \eta_2(\vartheta_E)(a) \ni \vartheta \end{cases} \\
(\rho,\eta,\Phi) \models^{\vartheta} P_0 + P_1 & \text{iff } (\rho,\eta,\Phi) \models^{\vartheta +\!\!+_0} P_0 \wedge (\rho,\eta,\Phi) \models^{\vartheta +\!\!+_1} P_1 \\
(\rho,\eta,\Phi) \models^{\vartheta} P_0 | P_1 & \text{iff } (\rho,\eta,\Phi) \models^{\vartheta ||_0} P_0 \wedge (\rho,\eta,\Phi) \models^{\vartheta ||_1} P_1 \\
(\rho,\eta,\Phi) \models^{\vartheta} (\nu a)P & \text{iff } (\rho,\eta,\Phi) \models^{\vartheta} P
\end{array}
$$

Table 2. Control Flow Analysis for the π-calculus.

- $\eta_2 : \mathcal{A} \rightarrow (Val \rightarrow \wp(\mathcal{A}))$ associates names with the addresses of sub-processes that can receive and send values on them; more precisely, $\eta_2(\vartheta)(a)$ must include the set of addresses ϑ' such that $P@\vartheta'$ can receive values on a, sent by $P@\vartheta$. For instance, if $\vartheta \in \eta_2(\vartheta')(a)$, with $\vartheta, \vartheta' \neq \vartheta_E$, then an internal action on channel a having $P@\vartheta'$ as a sender and $P@\vartheta$ as a receiver is possible, where ϑ_E is the generic address of an unknown process harboured in E, by definition compatible with all the addresses in P.

– $\Phi \subseteq \lfloor \mathcal{N} \rfloor$ represents the *external environment* (see also [5]). This component establishes a super-set of the values the environment can send and receive on the channels it knows. According to the Dolev and Yao model [10, 2], (i) E initially knows a subset of the names of P; (ii) afterwards, E may increase its knowledge by communicating with P and (iii) E can use all the known values to communicate with P, possibly affecting its behaviour.

Once defined the form of analysis estimates, a Flow Logic for when estimates are acceptable takes the form of defining a number of clauses, much in the style of Type Systems. These clauses operate upon judgments of the form: $(\rho,\eta,\Phi) \models^{\vartheta} P$. An estimate for a process P is then a triple (ρ,η,Φ) s.t. $(\rho,\eta,\Phi) \models^{\epsilon} P$, where $fn(P) \subseteq \Phi$ and $comp = comp_P$. Our Control Flow Analysis is defined by the Flow Logic clauses in Tab. 2.

To understand how the processes are validated, consider the most crucial rules, those for parallel composition, for output, and for input. As all the rules for a compound process require that the components are validated, we omit these conjuncts below.

Consider the parallel composition rule. An estimate is valid for the sub-process $(P_0|P_1)$ with address ϑ with respect to the initial process P under analysis, if it is valid for each P_i $(i = 0, 1)$ with address $\vartheta||_i$, in fact if $(P_0|P_1) = P@\vartheta$ then, by definition, $P_i = (P_0|P_1)@\vartheta||_i$. (Note that in the clauses, addresses are only used and updated when the top-level operator is a $|$ or a $+$, i.e. in the parallel composition and in the summation rules.)

Consider now the output case. To validate the process $\overline{x}y.P$ the rule requires that

$\ldots \forall a \in \rho(x):$	for each name a that can be bound to x
$\rho(y) \subseteq \eta_1(\vartheta)(a) = J$	the set of names $\eta_1(\vartheta)(a)$ that can be sent at ϑ along each element a of $\rho(x)$, must include the names to which y can evaluate, i.e. $\rho(y)$.
$\wedge\ (a \in \Phi$	if a belongs to the environment knowledge and
$\wedge\ J \neq \emptyset)$	there is a possible flow of values on a, i.e. $\eta_1(\vartheta)(a) = J \neq \emptyset$, then
$\Rightarrow\ (\eta_1(\vartheta)(a) \subseteq \Phi \ \wedge$	the environment knows the values that flow on a, i.e. they are in Φ, and
$\eta_2(\vartheta)(a) \ni \vartheta_E))$	therefore, there is a possible communication between the sub-process at ϑ and E

The more demanding rule is the rule for input. For validating the process $x(y \in Y).P$, we require that

$\ldots \forall a \in \rho(x),$	for each name a that can be bound to x and
$\forall \vartheta' : comp(\vartheta, \vartheta'):$	for each compatible address ϑ', let
$J = (\eta_1(\vartheta')(a) \cap \rho(Y))$	J and J' be the intersections on the left
$J' = (\Phi \cap \rho(Y))$	Then,
$(J \neq \emptyset) \Rightarrow \begin{cases} \eta_1(\vartheta')(a) \cap \rho(Y)) \subseteq \rho(y) \\ \eta_2(\vartheta')(a) \ni \vartheta \end{cases}$	some of the values that can be sent on a at ϑ', i.e. $\eta_1(\vartheta')(a)$, are included in the set of names to which $y \in Y$ can evaluate, provided that they belong also to $\rho(Y)$, i.e. $J \neq \emptyset$, and there is a communication between the sub-process at ϑ' and the one at ϑ
$(a \in \Phi \wedge J' \neq \emptyset)\ \Rightarrow$	if a belongs to the environment knowledge,
$\begin{cases} (\Phi \cap \rho(Y)) \subseteq \rho(y) \\ \eta_2(\vartheta_E)(a) \ni \vartheta \end{cases}$	some of the values (in Φ) that can be sent on a by any process in E are included in the set of names to which $y \in Y$ can evaluate, provided that they belong also to $\rho(Y)$, i.e. $J' \neq \emptyset$, and there is a communication between the sub-process at ϑ' and the one in E

Note that whenever we test for emptiness, we address the reachability issue; indeed the condition $\eta_1(\vartheta)(a) = J = \emptyset$ in the rule for output amounts to saying that an output action in that position, on channel a, is not reachable (e.g the output action $\overline{a}x$ in the process $(\nu b)b(x \in \{c\}).\overline{a}x$). Similarly for input conditions: if $J = (\eta_1(\vartheta')(b) \cap \rho(Y)) = \emptyset$ and $J' = \Phi \cap \rho(Y) = \emptyset$ then the input action on b is not reachable, as the action $b(x \in \{c\})$ in the example above.

Example 1. To give the flavour of the analysis, we use the following example.

$$R = P \mid Q = (\underbrace{\overline{a}d.\overline{c}d}_{||_0 +\!\!+_0} + \underbrace{\overline{a}b.c(z)}_{||_0 +\!\!+_1} \ \mid\ \underbrace{a(w \in \{b\}).\overline{c}w.}_{||_1}$$

The analysis should provide us with the following information:

- which are the values that can be bound to each variable. In terms of the CFA, we need to know $\rho(z)$ and $\rho(w)$;
- the set of values that can be sent over the channels a and c, i.e. $\eta_1(\vartheta)(a)$ and $\eta_1(\vartheta)(c)$ for $\vartheta \in \{||_0 +\!\!+_0, ||_0 +\!\!+_1, ||_1\}$;
- the set of addresses of sub-processes that can send and receive values on the channels, i.e. $\eta_2(\vartheta)(a)$ and $\eta_2(\vartheta)(c)$ for $\vartheta \in \{||_0 +\!\!+_0, ||_0 +\!\!+_1, ||_1\}$;
- finally, the contribute Φ of the environment. Initially, the environment knows all the free names $\{a, b, c, d\}$ of the process R.

To establish $(\rho, \eta, \Phi) \models^{\epsilon} R$ it is necessary to establish $(\rho, \eta, \Phi) \models^{||_0} P$ and $(\rho, \eta, \Phi) \models^{||_1} Q$, where $R@||_0 = P$ and $R@||_1 = Q$, and, in turn, to establish $(\rho, \eta, \Phi) \models^{||_0} P$ it is necessary to establish $(\rho, \eta, \Phi) \models^{||_0 +\!\!+_0} \overline{a}d.\overline{c}d$ and $(\rho, \eta, \Phi) \models^{||_0 +\!\!+_1} \overline{a}b.c(z)$. We can compute an estimate giving the following results:

- $\rho(z) \supseteq \{b, \mathbf{a}, \mathbf{c}, \mathbf{d}\}$ and $\rho(w) \supseteq \{b\}$
- $\eta_1(||_0 +\!\!+_0)(a) = \{d\}$, $\eta_1(||_0 +\!\!+_1)(a) = \{b\}$, $\eta_1(||_1)(a) = \emptyset$ and $\eta_1(||_0 +\!\!+_i)(c) = \emptyset$, $\eta_1(||_1)(c) \supseteq \{b\}$
- $\Phi \supseteq \{a, b, c, d\}$

Note that we write in boldface some of the names in $\rho(z)$, to stress the contribute of the environment. As a matter of fact, in the process in isolation, the only value that could be bound to z would be b, the value passed from the only compatible action $\overline{c}w$. In particular the action $\overline{c}d$, being not compatible, could not contribute. Nevertheless, since $(\Phi \cap \lfloor \mathcal{N} \rfloor) = \Phi \subseteq \rho(z)$, then all the names in Φ belong to $\rho(z)$, and in particular d. Dynamically, this amounts to putting R in parallel with an environment including the sub-process $a(x).\overline{c}x$. Furthermore, $\rho(w)$ only includes $\{b\}$, that coincides with the non-empty intersection $(\eta_1(||_0 +\!\!+_0)(a) \cup \eta_1(||_0 +\!\!+_1)(a) \cup \Phi) \cap \{b\}$.

As described above, our CFA is formalized as a specification of a candidate estimate. It is possible to prove the existence of a least choice of (ρ, η, Φ), acceptable according to the rules in Tab. 2, In fact, estimates can be partially ordered and arranged in a Moore family and a Moore family always contains a least element. Furthermore, we can prove the semantic correctness of our analysis, in the form of a subject reduction result. Intuitively, if an estimates validates a process P and P evolves in Q, then the estimate validates Q as well. Finally, it is possible to define a constructive procedure for obtaining the least solution, whose complexity we argue is low-polynomial.

Note that the same results can be obtained with the standard π-calculus. Nevertheless, our variant is more immediate in representing open systems and helps in simplifying the definition of the CFA.

4 CFA & Security Properties

Our analysis statically approximates the behaviour of a process P under consideration, in particular it tracks where and between whom communications may occur. This is essentially done through its component η_2. Actually, exploiting the soundness of our analysis, we can prove, among others, the following three basic facts. Suitable combinations of them enable us to define and check many security properties. Essentially, they are centered on access control and prescribe which processes can communicate each other and in which direction.

- **[no output for $P@\vartheta'$ on a]** The process $P@\vartheta'$ cannot use the channel a to send data to the environment, when $\vartheta_E \notin \eta_2(\vartheta')(a)$, (with $\vartheta' \neq \vartheta_E$).
- **[no input for $P@\vartheta$ on a]** The process $P@\vartheta$ cannot use the channel a to receive data from the environment, when $\vartheta \notin \eta_2(\vartheta_E)(a)$, (with $\vartheta \neq \vartheta_E$).
- **[no communication between $P@\vartheta$ and $P@\vartheta'$ on a]** The process $P@\vartheta'$ cannot communicate with $P@\vartheta$ on channel a, when $\vartheta \notin \eta_2(\vartheta')(a)$, (with $\vartheta, \vartheta' \neq \vartheta_E$), i.e. no exchange of information occurs on a, from site ϑ' to site ϑ within the system.

The first fact can be made more precise, by exploiting the sub-component η_1 of an estimate. It permits to establish suitable tests for when a sub-process $P@\vartheta'$ cannot send a selected value b to the environment. If b is considered a sensible data, its *secrecy* is indeed guaranteed. Analogously, one can refine the second fact: a sub-process $P@\vartheta'$ can ascertain that a specific value b comes only from inside the system, so establishing a form of *authentication*.

Our last fact provides us with the basis for statically checking whether a process follows a particular security policy.

As a first example of these policies, consider the following variant of the *no read-up/no write-down* property by Bell and LaPadula [3] (see also [12]). Processes receive clearance levels of security and a process with a high level cannot write any value to a process at low level, while the converse is allowed; symmetrically a process at low level cannot read data from one at a high level. In other words, the dynamic property requires that there is no communication between the low-level process $P@\vartheta$ and the high-level process $P@\vartheta'$. More precisely, it requires that $P@\vartheta'$ does not perform an output, i.e. a write down to $P@\vartheta$ (see the simple example below). The corresponding static test is straightforward: $\vartheta \notin \eta_2(\vartheta')$.

Consider now the dual policy for integrity, based on the Biba model [4]. It requires that a sub-process $P@\vartheta$ with a low clearance level cannot write any value to a sub-process $P@\vartheta'$ at a higher level, while the converse is allowed. Again the point is that there should not be any communication between $P@\vartheta$ and $P@\vartheta'$ in the wrong sense. The static test also turns out to be easy: $\vartheta' \notin \eta_2(\vartheta)$.

Some simpler policies focus on polarity (see also [16, 13]). These policies prescribe for each channel if a particular process can use it either for reading or for writing or in both directions. Checking that a process obeys a given policy is again straightforward. For instance, say that channel a cannot be used for output by a sub-process $P@\vartheta'$. This property is guaranteed by the following simple check: $\eta_1(\vartheta')(a) = \emptyset$.

5 An example

For lack of space, we just present a small example, to simply illustrate our technique.

Assume that within an organization, *Alice* (A), a top manager, creates a file *trend* containing an analysis of the organization trend, i.e. a collection of important and sensitive data. This data should not be disclosed to anybody besides *Alice*. Consider now *Bob* (B), one of *Alice*'s subordinates, who wants to acquire some information about the file to exploit it within a competitor organization. *Bob* modifies a web consultant application generally used by *Alice*, to include one hidden operation: a write operation on the channel *hidden* (he uses it as a Trojan horse). Then he gives this application to *Alice*. The application receives through the channel *line* the information on the trend of the organization and if it is negative, i.e. if it is *bad*, then sends Alice some suggestion. Otherwise, it sends an acknowledgement of the communication, through channel *ack*. When *Alice* will execute the application, also the hidden write operation can be performed. As a result, *Bob* will acquire the desired information, through the channel *hidden* and he can use it to decide, e.g. to sell his actions, through the channel *sell*. Recall that in the π-calculus, the restriction operator acts as a static binder, i.e. the restricted name is private of the process prefixed by the restriction: for instance, the restricted name *good* is not known outside A, until it is explicitly sent, as in the example with the action $\overline{line}\langle good\rangle$. Below, we present a specification, in which A' and B' represent two suitable continuations of the processes A and B.

$$\begin{aligned}
System &= (\nu\ line)(\underbrace{A}_{||_0||_0} \mid \underbrace{WebConsultant)}_{||_0||_1}) \mid \underbrace{B}_{||_1}\\
A &= (\nu\ good)(\nu\ bad)(\overline{line}\langle good\rangle.ack(x_1).A' \mid \overline{line}\langle bad\rangle.line(x_2).A'')\\
WebConsultant &= (line(z_{trend} \in \{good\}).\overline{ack}\langle ok\rangle) \mid\\
&\quad\ (line(y_{trend} \in \{bad\}).\overline{line}\langle suggestion\rangle.\overline{hidden}\langle SideEffect\rangle)\\
B &= hidden(w).\overline{sell}\langle actions\rangle.B'
\end{aligned}$$

Assign now a high security level H to A ($||_0||_0$) and to her sub-components ($||_0||_0||_i$), as well as to the WebConsultant application ($||_0||_1$) and to its sub-components ($||_0||_1||_i$); assign instead low security level L to B ($||_1$). The following is the relevant part of an acceptable estimate.

- $\eta_1(||_0||_0||_0)(line), \rho(z_{trend}) \ni good$ and $||_0||_1||_0 \in \eta_2(||_0||_0||_0)(line)$;
- $\eta_1(||_0||_0||_1)(line), \rho(y_{trend}) \ni bad$ and $||_0||_1||_1 \in \eta_2(||_0||_0||_1)(line)$;
- $\eta_1(||_0||_1||_0)(ack), \rho(x_1) \ni ok$ and $||_0||_0||_0 \in \eta_2(||_0||_1||_0)(ack)$;
- $\eta_1(||_0||_1||_1)(line), \rho(x_2) \ni suggestion$ and $||_0||_0||_1 \in \eta_2(||_0||_1||_1)(line)$;
- $\eta_1(||_0||_1||_1)(hidden), \rho(w) \ni SideEffect$ and $||_1 \in \eta_2(||_0||_1||_1)(hidden)$;
- $\eta_1(||_1)(sell) \ni actions$ and $\vartheta_E \in \eta_2(||_1)(sell)$.

It is immediate to check that our system violates the *no read-up/no write-down* policy. In fact, the estimate is such that $||_1 \in \eta_2(||_0||_1||_1)(hidden)$, with $||_1 \in L$ and $||_0||_1||_1 \in H$. This corresponds to the fact that the high-level sub-process $System@||_0||_1||_1$ performs a *write down* to the low sub-process $System@||_1$ (that, in turn, performs a *read up*), that corresponds to a undesired flow of information from high to low. The same system respects instead the dual property: the integrity of data of high-level processes is indeed preserved.

6 Concluding Remarks

Our present analysis gains precision with respect to the one in [6]. Technically, this is done by giving a static interpretation to the main operators of the calculus $(|, +, .)$, which is closer to the dynamic interpretation. This results in a better approximation of the whole behaviour of the system under consideration, in terms of communications: some of the communications that in [6] were wrongly considered possible (false positives) are discarded. Moreover, our CFA is able to approximate the local behaviour of sub-systems, e.g. we can approximate what happens on a particular channel a in a particular location. An estimate, computed *once and for all*, allows for a variety of static checks, that imply many security properties, besides those considered here, based on confidentiality and access control. This gives evidence that our approach offers a uniform, flexible and expressive framework for analysing security properties.

References

1. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols - The Spi calculus. *Information and Computation 148*, 1:1–70, January 1999.
2. M. Abadi, Security protocols and specifications, In *Proc. FoSSaCS'99*, LNCS 1578, pp. 1–13, Springer, Berlin, 1999.
3. D.E. Bell and L.J. LaPadula. Secure computer systems: Unified exposition and Multics interpretation. Technical Report ESD-TR-75-306, Mitre C., 1976.
4. K.J. Biba. Integrity Considerations for Secure Computer Systems. US Airforce Electronic Systems Division, 1977.
5. C. Bodei, P. Degano, R. Focardi, R. Gorrieri, F. Martinelli. Techniques for security checking: Non-Interference vs Control Flow Analysis *Proc. of the Final Workshop Tosca 2001*, ENTCS 62, 2001.
6. C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Static analysis for the π-calculus with their application to security. *Info. & Co.*168: 68-92, 2001.
7. C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Flow Logic for Dolev-Yao Secrecy in Cryptographic Processes. *Future Generation Computer Systems* 18 (6): 747-756, 2002, Elsevier.
8. C. Bodei, P. Degano, and C. Priami. "Names of the π-Calculus Agents Handled Locally". *TCS*, 253(2):155–184, 2001.
9. P. Degano and C. Priami. "Enhanced Operational Semantics.". *ACM Computing Surveys*, 33, 2 (June 2001), 135-176.
10. D. Dolev and A.C. Yao, On the security of public key protocols, IEEE TIT, IT-29(12) (1983) 198–208.
11. J. Feret. Confidentiality Analysis of Mobile Systems. In *Proc. of SAS'00*, LNCS 1824, pp. 135–154. Springer, 2000.
12. M. Hennessy and J.Riely. Information Flow vs. Resource Access in the Asynchronous Pi-Calculus. In *Proc. of ICALP'00*, LNCS 1853, pp. 415–427, Springer, 2000.
13. B. König. Analysing Input/Output-Capabilities of Mobile Processes with a Generic Type System. In *Proc. of ICALP'00*, LNCS 1853, pp. 403–414, Springer, 2000.
14. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (I and II). *Info. & Co.*, 100(1):1–77, 1992.
15. F. Nielson and H. Riis Nielson. Flow logics and operational semantics. *ENTCS*, 10, 1998.
16. B.C. Pierce, and D. Sangiorgi. Typing and Sub-typing for Mobile Processes. *MSCS* 6(5), pp.409-454,1996.