Control Flow Analysis for the π-calculus

Chiara Bodei, Pierpaolo Degano,[1]
Flemming Nielson, Hanne Riis Nielson[2]

[1] Dipartimento di Informatica, Università di Pisa
Corso Italia 40, I-56100 Pisa, Italy
{chiara,degano}@di.unipi.it
[2] Computer Science Department, Aarhus University
Ny Munkegade, DK-8000 Aarhus C, Denmark
{fn,hrn}@daimi.aau.dk

Abstract. Control Flow Analysis is a static technique for predicting safe and computable approximations to the set of values that the objects of a program may assume during its execution. We present an analysis for the π-calculus that shows how names will be bound to actual channels at run time. The formulation of the analysis requires no extensions to the π-calculus, except for assigning "channels" to the occurrences of names within restrictions, and assigning "binders" to the occurrences of names within input prefixes.

The result of our analysis establishes a super-set of the set of names to which a given name may be bound and of the set of names that may be sent along a given channel. Applications of our analysis include establishing simple security properties of processes. One example is that P has *no leaks*, i.e. P offers communication through public channels only, and confines its secret names within itself.

1 Introduction

Program analysis aims at analysing properties of a program that hold in all executions – regardless of the actual data upon which the program operates and regardless of the specific environment in which it executes. Traditionally, program analysis has been used in compilers for "optimizing" the implementation of programming languages. Various classes of programming languages have then given rise to specific techniques. For example, Data Flow Analysis [11] was mainly developed for imperative languages but is also used for object-oriented languages, and Control Flow Analysis [18] was mainly developed for functional languages but can be used also for object-oriented languages [14] and languages with concurrency [9].

Program analysis provides automatic and decidable methods for analysing properties of programs. Since most of them implicitly involve questions about termination, the properties are intended to "err on the safe side". For each analysis an ordering is imposed on the properties, for example stipulating that a property is larger than another if more values satisfy the former than the latter. The properties are then interpreted in such a way that an analysis remains correct even when it produces a larger property than ideally possible. This corresponds to producing a valid inference in a program logic for partial correctness. However, program analysis is generally more efficient than program verification, and for that reason more approximate, because the focus is on the fully automatic processing of large programs.

More recently, program analysis has been used for validating security and safety issues for concurrent and distributed systems. To study these issues in a pure form we shall use the π-calculus which is a model of concurrent communicating processes based on naming. Our formulation of a Control Flow Analysis for the π-calculus only requires minor additions to the syntax: adding explicit "channels" to the names occurring in restrictions and adding explicit "binders" to the names occurring in input prefixes. This may be compared to the approach of [19] where a program analysis based on Abstract Interpretation demands the π-calculus expressions to be on a special form (called normal form).

The result of our Control Flow Analysis establishes a super-set of the set of names to which a given name may be bound and of the set of names that may be sent along a given channel. To study the security of systems, channels have been divided into "secret" and "public" ones: the idea then is that security is broken if secret information may be communicated over public channels. In other words, we obtain a static test for a given process having no leaks.

Overview. We begin by reviewing the syntax and early semantics of the π-calculus. Since program analysis studies more intensional properties than is normal for the π-calculus, we also introduce a new auxiliary notion (that of "free interesting names") and study its properties. Then we formulate the Control Flow Analysis as a specification of the correctness of a candidate solution; we show that best solutions always exist and we establish their semantic correctness (in the form of a subject-reduction result). We then present our treatment of security and demonstrate how a static condition obtained from the analysis can be used to guarantee the dynamic condition: that a process has no leaks. Finally, we conclude by discussing some other uses of Control Flow Analysis for guaranteeing security properties of π-calculus expressions. Because of lack of space, we only sketch the proofs of the main facts.

2 The π-calculus

In this section we briefly recall the π-calculus [12], a model of concurrent communicating processes based on the notion of *naming*. The formulation of our analysis requires no extensions to the π-calculus, except for assigning "channels" to the occurrences of names within restrictions, and assigning "binders" to the occurrences of names within input prefixes. The intuition of process constructors is therefore the standard one, apart from the usage of channels and binders, whose purpose will be explained in the next sections. Intuitively, these annotations will be used to compute a super-set of the actual links that a name can denote.

Definition 1. Let $\mathcal{N}$ be a countably infinite set of *names* ranged over by $a, b, \cdots, x, y, \cdots, w$ and let τ be a distinguished element such that $\mathcal{N} \cap \{\tau\} = \emptyset$. Also let $\mathcal{B}$ be a non-empty set of *binders* ranged over by $\beta, \beta', \cdots$; and let $\mathcal{C}$ be a non-empty set of *channels* ranged over by $\chi, \chi', \cdots$; moreover let $\mathcal{M} = \mathcal{B} \cup \mathcal{C}$ be the set of *markers*. Then *processes*, denoted by $P, P_1, P_2, Q, R, \cdots \in \mathcal{P}$ are built from names according to the syntax

$$P ::= \mathbf{0} \mid \mu.P \mid P + P \mid P|P \mid (\nu x^{\chi})P \mid [x = y]P \mid !P$$

where μ may either be $x(y^\beta)$ for *input*, or $\overline{x}y$ for *output* or τ for *silent* moves. Hereafter, the trailing $\mathbf{0}$ will be omitted (i.e. we write π instead of $\pi.\mathbf{0}$). We write $\mathcal{B}_P$ and $\mathcal{C}_P$ for the binders and for the channels occurring in P, respectively. $\diamondsuit$

The π-calculus can be equipped with an early as well as a late semantics; in this paper we consider the *early* operational semantics defined in SOS style. Here the labels of transitions are τ for silent actions, xy for free input, $\overline{x}y$ for free output, and $\overline{x}(y)$ for bound output. We will use μ as a metavariable for the labels of transitions (although it is formally distinct from the metavariable for prefixes with which it has a few cases in common). We recall the notion of free names $fn(\mu)$, bound names $bn(\mu)$, and names $n(\mu) = fn(\mu) \cup bn(\mu)$ of a label μ. Also two partial functions, *sbj* and *obj*, are defined giving, respectively, the subject x and the object y of input and output actions, i.e. the channel x on which y is transmitted.

Kind	μ	$fn(\mu)$	$bn(\mu)$	$sbj(\mu)$	$obj(\mu)$
Silent move	τ	$\emptyset$	$\emptyset$		
Free input and output	$xy, \overline{x}y$	$\{x, y\}$	$\emptyset$	x	y
Bound output	$\overline{x}(y)$	$\{x\}$	$\{y\}$	x	y

Functions fn, bn and n are extended in the obvious way to processes.

Below we shall need the *structural congruence* $\equiv$ on processes, defined as in [13] to be the least congruence satisfying:

- if P and Q are α-equivalent ($P =_\alpha Q$) then $P \equiv Q$; to be more precise: $(\nu x^\chi)P \equiv (\nu y^\chi)(P\{y/x\})$ if $y \notin fn((\nu x^\chi)P)$, and $x(y^\beta)P \equiv x(z^\beta)(P\{z/y\})$ if $z \notin fn(x(y^\beta)P)$;
- $(\mathcal{P}/_\equiv, +, \mathbf{0})$ is a commutative monoid; to be more precise: $\mathbf{0} + P \equiv P$, $P_1 + P_2 \equiv P_2 + P_1$, and $P_1 + (P_2 + P_3) \equiv (P_1 + P_2) + P_3$;
- $(\mathcal{P}/_\equiv, |, \mathbf{0})$ is a commutative monoid;
- $[x = x]P \equiv P$;
- $(\nu x^\chi)(\nu y^{\chi'})P \equiv (\nu y^{\chi'})(\nu x^\chi)P$, if $x \neq y$, $(\nu x^\chi)(P_1|P_2) \equiv (\nu x^\chi)P_1|P_2$ if $x \notin fn(P_2)$, and $(\nu x^\chi)P \equiv P$ if $x \notin fn(P)$;
- $!P \equiv P|!P$.

Note that we permit to exchange restrictions only if the restricted names are different, because otherwise $(\nu x^\chi)P \equiv (\nu x^{\chi'})P$.

To establish the correctness of our analysis we need to make certain assumptions about certain names occurring in the process being analysed. A difficulty here is that the congruence law $P \equiv [x = x]P$ allows[3] to introduce new names. For this reason we introduce the *free interesting names* $fin(P)$ of a process P to exclude names that occur only in $[- = -]$. The formal definition is as follows:

[3] This problem could be avoided by removing the law $P \equiv [x = x]P$ and by adding $[x = x]P \longrightarrow P$ in the semantics to be presented below. However, since this is a recurring problem in congruence-based semantics, we want to illustrate the techniques needed to deal with it.

$$fin(\mathbf{0}) = \emptyset$$
$$fin(\tau.P) = fin([x=y]P) = fin(!P) = fin(P)$$
$$fin(\overline{x}y.P) = fin(P) \cup \{x,y\}$$
$$fin(x(y^\beta).P) = (fin(P) \setminus \{y\}) \cup \{x\}$$
$$fin(P_1 + P_2) = fin(P_1|P_2) = fin(P_1) \cup fin(P_2)$$
$$fin((\nu x^\chi)P) = fin(P) \setminus \{x\}$$

So $fin([x=y]P) = fin(P)$, whereas $fn([x=y]P) = fn(P) \cup \{x,y\}$.

Table 1 shows the annotated *early* transition system of the π-calculus. A *variant* of $P \xrightarrow{\mu} Q$, is a transition which only differs in that P and Q have been replaced by structurally congruent processes. Rule Var ensures that all rules and axioms can also be used upon all its variants. Under the arrows of our transitions there is an additional label $\lambda \in \mathcal{C} \cup \{\epsilon\}$. The label ϵ is used in all cases except for extrusions or when input transitions have to be used as a premise of a $Close$ rule. In that case the label is χ and records the actual channel to be associated with the object of the input.

[tb]

$$Act : \mu.P \xrightarrow[\epsilon]{\mu} P, \;\; \mu \neq x(y^\beta) \qquad Ein : x(y^\beta).P \xrightarrow[\lambda]{xw} P\{w/y\} \quad Var : \frac{P' \equiv P \xrightarrow[\lambda]{\mu} Q \equiv Q'}{P' \xrightarrow[\lambda]{\mu} Q'}$$

$$Par : \frac{P_1 \xrightarrow[\lambda]{\mu} Q_1}{P_1|P_2 \xrightarrow[\lambda]{\mu} Q_1|P_2}, bn(\mu) \cap fn(P_2) = \emptyset \qquad Sum : \frac{P_1 \xrightarrow[\lambda]{\mu} Q_1}{P_1 + P_2 \xrightarrow[\lambda]{\mu} Q_1}$$

$$Res : \frac{P \xrightarrow[\lambda]{\mu} Q}{(\nu x^\chi)P \xrightarrow[\lambda]{\mu} (\nu x^\chi)Q}, x \notin n(\mu) \qquad Open : \frac{P \xrightarrow[\epsilon]{\overline{x}y} Q}{(\nu y^\chi)P \xrightarrow[\chi]{\overline{x}(y)} Q}, y \neq x$$

$$Close : \frac{P_1 \xrightarrow[\chi]{\overline{x}(y)} Q_1, P_2 \xrightarrow[\chi]{xy} Q_2}{P_1|P_2 \xrightarrow[\epsilon]{\tau} (\nu y^\chi)(Q_1|Q_2)}, y \notin fn(P_2) \qquad Com : \frac{P_1 \xrightarrow[\epsilon]{\overline{x}y} Q_1, P_2 \xrightarrow[\epsilon]{xy} Q_2}{P_1|P_2 \xrightarrow[\epsilon]{\tau} Q_1|Q_2}$$

Table 1. Early transition system for the π-calculus.

We now state a few consequences of the notion of free interesting names. The first item is just an analogue of an obvious property that holds for the free names.

The second item is more interesting because there is no analogue for the set of free names: clearly $[x = x]\mathbf{0} \equiv \mathbf{0}$ but $\mathit{fn}([x = x]\mathbf{0}) \neq \mathit{fn}(\mathbf{0})$.

Lemma 2. *We have the following:*

- $\mathit{fin}(P\{w/y\}) = \begin{cases} (\mathit{fin}(P) \setminus \{y\}) \cup \{w\} & \text{if } y \in \mathit{fin}(P) \\ \mathit{fin}(P) & \text{if } y \notin \mathit{fin}(P) \end{cases}$
- *If* $P \equiv Q$ *then* $\mathit{fin}(P) = \mathit{fin}(Q)$.

Finally we state a result that ensures that the set of free interesting names can only grow when performing bound outputs or inputs. Note that in the last three cases below $x \in \mathit{fin}(P)$ (and in the second case also $y \in \mathit{fin}(P)$); otherwise, the transition could not be deduced, because the side-condition of rule Res would be violated.

Lemma 3. *We have the following:*

- *If* $P \xrightarrow[\epsilon]{\tau} Q$ *then* $\mathit{fin}(P) \supseteq \mathit{fin}(Q)$*;*
- *If* $P \xrightarrow[\epsilon]{\overline{x}y} Q$ *then* $\mathit{fin}(P) \supseteq \{x, y\} \cup \mathit{fin}(Q)$*;*
- *If* $P \xrightarrow[\chi]{\overline{x}(y)} Q$ *then* $\mathit{fin}(P) \supseteq \{x\} \cup (\mathit{fin}(Q) \setminus \{y\})$*;*
- *If* $P \xrightarrow[\lambda]{xy} Q$*,* $\lambda \in \mathcal{C} \cup \{\epsilon\}$ *then* $\mathit{fin}(P) \supseteq \{x\} \cup (\mathit{fin}(Q) \setminus \{y\})$*.*

3 Control Flow Analysis

The result of our analysis for a process P in the π-calculus (with respect to an additional marker environment me for associating free names with markers) is a pair (ρ, κ): the abstract environment ρ gives information about which channels names can be bound to, and the abstract channel cache κ gives information about which channels can be communicated over channels. We now make this more precise.

3.1 Validation

To *validate* the correctness of a proposed solution (ρ, κ) we state a number of clauses operating upon judgments of the form:

$$(\rho, \kappa) \ \models_{me} P$$

The purpose of me, ρ and κ is clarified by:

- $me : \mathcal{N} \to (\mathcal{B} \cup \mathcal{C})$ is the *marker environment* that associates names (in particular the free names of a process) with the appropriate channel or binder where the name was introduced; so $me(x)$ will be the marker (in $\mathcal{B}$ or $\mathcal{C}$) where the current name x is bound;
- $\rho : \mathcal{B} \to \wp(\mathcal{C})$ is the *abstract environment* that associates binders with the set of channels that they can be bound to; more precisely, $\rho(\beta)$ must include the set of channels that β could evaluate to. We shall allow to regard the abstract environment as a function $\rho : (\mathcal{B} \cup \mathcal{C}) \to \wp(\mathcal{C})$ by setting $\forall \chi : \rho(\chi) = \{\chi\}$.
- $\kappa : \mathcal{C} \to \wp(\mathcal{C})$ is the *abstract channel environment* that associates channels with the set of channels that can be communicated over them; more precisely, $\kappa(\chi)$ must include the set of channels that can be communicated over the channel χ.

Note that we use a marker environment, because the identity of names is not preserved under α-conversions (see rule Ein). In particular, it would not suffice to "α-rename the program apart" because this property is not preserved under reduction (e.g. $x(y).(\nu w)\overline{w}(y) \xrightarrow{xw} ((\nu w)\overline{w}(y))\{w/y\}$).

The detailed definition is provided in Tab. 2. All rules for validating a compound process require that the components are validated. Moreover, the second conjunct of the rule for output requires that the set of channels that can be communicated along x (i.e. along each element of $\rho(me(x))$) includes the channels to which y can evaluate. Symmetrically for the case of input. The condition for matching says that P needs to be validated if there is at least one channel to which both x and y can evaluate (note that both can evaluate to $\emptyset$ and thus we need to check whether they actually denote the same channel by requiring $me(x) = me(y)$).

[tb]

$$
\begin{array}{lll}
(\rho,\kappa) \models_{me} \mathbf{0} & \textsf{iff} & \mathit{true} \\
(\rho,\kappa) \models_{me} \tau.P & \textsf{iff} & (\rho,\kappa) \models_{me} P \\
(\rho,\kappa) \models_{me} \overline{x}y.P & \textsf{iff} & (\rho,\kappa) \models_{me} P \wedge \\
 & & \quad \forall \chi \in \rho(me(x)) : \rho(me(y)) \subseteq \kappa(\chi) \\
(\rho,\kappa) \models_{me} x(y^{\beta}).P & \textsf{iff} & (\rho,\kappa) \models_{me[y \mapsto \beta]} P \wedge \\
 & & \quad \forall \chi \in \rho(me(x)) : \kappa(\chi) \subseteq \rho(\beta) \\
(\rho,\kappa) \models_{me} P_1 + P_2 & \textsf{iff} & (\rho,\kappa) \models_{me} P_1 \wedge (\rho,\kappa) \models_{me} P_2 \\
(\rho,\kappa) \models_{me} P_1 | P_2 & \textsf{iff} & (\rho,\kappa) \models_{me} P_1 \wedge (\rho,\kappa) \models_{me} P_2 \\
(\rho,\kappa) \models_{me} (\nu x^{\chi})P & \textsf{iff} & (\rho,\kappa) \models_{me[x \mapsto \chi]} P \qquad \text{(since } \{\chi\} = \rho(\chi)) \\
(\rho,\kappa) \models_{me} [x = y]P & \textsf{iff} & (\rho(me(x)) \cap \rho(me(y)) \neq \emptyset \vee me(x) = me(y)) \\
 & & \quad \Rightarrow (\rho,\kappa) \models_{me} P \\
(\rho,\kappa) \models_{me} !P & \textsf{iff} & (\rho,\kappa) \models_{me} P
\end{array}
$$

Table 2. Control flow analysis for the π-calculus.

As an example, consider the following process

$$P = a(x^{\beta_0}).(\nu b^{\chi_0})(\nu c^{\chi_1})((\overline{b}a.\overline{x}x.b(x^{\beta_1}).\overline{x}c \ + \ \overline{b}d.\overline{a}c) \mid b(x^{\beta_2}).\overline{b}x) \mid d(x^{\beta_3}),$$

the marker environment me such that $me(a) = \chi_2$ and $me(d) = \chi_3$, and the pair (ρ, κ) defined as follows, where $i \in \{0, 1, 2, 3, 4\}$

$$\rho(\beta_i) = \begin{cases} \{\chi_0, \chi_1, \chi_2, \chi_3, \chi_4\} & \text{if } i = 1, 2 \\ \{\chi_1, \chi_2, \chi_3, \chi_4\} & \text{if } i = 0, 3 \end{cases} \qquad \kappa(\chi_i) = \begin{cases} \{\chi_0, \chi_1, \chi_2, \chi_3, \chi_4\} & \text{if } i = 0 \\ \{\chi_1, \chi_2, \chi_3, \chi_4\} & \text{if } i \geq 1 \end{cases}$$

A simple check shows that $(\rho, \kappa) \models_{me} P$.

3.2 Existence of solution

So far we have only considered a procedure for validating whether or not a proposed solution (ρ, κ) is in fact acceptable. We now show that there always exists a least choice of (ρ, κ) that is acceptable in the manner of Tab. 2.

The set of proposed solutions can be partially ordered by setting $(\rho, \kappa) \sqsubseteq (\rho', \kappa')$ if and only if $\forall \beta \in \mathcal{B} : \rho(\beta) \subseteq \rho'(\beta)$ and $\forall \chi \in \mathcal{C} : \kappa(\chi) \subseteq \kappa'(\chi)$. It is immediate that this suffices for making the set of proposed solutions into a complete lattice; we write $(\rho, \kappa) \sqcup (\rho', \kappa')$ for the binary least upper bound (defined pointwise), $\sqcap \mathcal{I}$ for the greatest lower bound of a set $\mathcal{I}$ of proposed solutions (also defined pointwise), and $(\perp, \perp)$ for the least element (where $\perp$ maps everything to $\emptyset$).

A set $\mathcal{I}$ of proposed solutions is a *Moore family* if and only if it contains $\sqcap \mathcal{J}$ for all $\mathcal{J} \subseteq \mathcal{I}$ (in particular for $\mathcal{J} = \emptyset$ and $\mathcal{J} = \mathcal{I}$). This is related to what is sometimes called the model intersection property. When $\mathcal{I}$ is a Moore family it contains a greatest element ($\sqcap \emptyset$) as well as a least element ($\sqcap \mathcal{I}$). Then we have the following theorem that is proved by structural induction on P (since Tab. 2 is defined likewise).

Theorem 4. *For all me, P and $(\overline{\rho}, \overline{\kappa})$ the set $\{(\rho, \kappa) \mid (\rho, \kappa) \models_{me} P \wedge (\rho, \kappa) \sqsupseteq (\overline{\rho}, \overline{\kappa})\}$ is a Moore family.*

This result guarantees that there always is a least solution to the specification in Tab. 2; just take $(\overline{\rho}, \overline{\kappa}) = (\perp, \perp)$. There is also a constructive procedure for obtaining the least solution, which is cubic in the size of processes (see [9]). Essentially, establishing $(\rho, \kappa) \models_{me} P$ amounts to checking a number of individual constraints. In the full paper we define a function $\mathcal{G}_C[\![P]\!]_{me}$ for explicitly extracting these constraints, proceeding by induction on the structure of processes. For instance, in case of output $\overline{x}y.P$, the idea is to add the set $\{\{\chi\} \subseteq \rho(me(x)) \Rightarrow \rho(me(y)) \subseteq \kappa(\chi) \mid \chi \in \mathcal{C}\}$ to the existing constraints obtained for P. To be more precise, constraints will be in the form of syntax, as we are inducing on syntax.

3.3 Correctness

We state now some auxiliary results that help in establishing semantic correctness of our analysis; they are all independent of the semantics and only rely on Tab. 2; their proofs are all by induction.

Lemma 5. *Assume that $\forall x \in fn(P) : me_1(x) = me_2(x)$; then $(\rho, \kappa) \models_{me_1} P$ if and only if $(\rho, \kappa) \models_{me_2} P$.*

It is interesting to note that the previous result fails if we had used free interesting names instead of free names: let $me_1(x) = \chi_1 = me_1(y)$, $me_2(x) = \chi_1 \neq \chi_2 = me_2(y)$, $me_1(z) = \chi_3 = me_1(z)$, $(\rho, \kappa) = (\perp, \perp)$ and consider $P = [x = y]\overline{x}z$; then $(\rho, \kappa) \models_{me_2} P$ holds but $(\rho, \kappa) \models_{me_1} P$ does not.

Lemma 6. *$(\rho, \kappa) \models_{me[y \mapsto me(z)]} P$ if and only if $(\rho, \kappa) \models_{me} P\{z/y\}$.*

Corollary 7. *Assume $z \notin fn(P)$ and $\eta \in \mathcal{M} = \mathcal{B} \cup \mathcal{C}$; then $(\rho, \kappa) \models_{me[y \mapsto \eta]} P$ if and only if $(\rho, \kappa) \models_{me[z \mapsto \eta]} P\{z/y\}$.*

Lemma 8. *Assume that $P \equiv Q$; then $(\rho, \kappa) \models_{me} P$ if and only if $(\rho, \kappa) \models_{me} Q$.*

Lemma 9. *If $(\rho, \kappa) \models_{me} P$, $me(w) \in \mathcal{C}$ and $\rho(me(w)) \subseteq \rho(me(z))$; then $(\rho, \kappa) \models_{me} P\{w/z\}$.*

To establish the semantic soundness of our analysis we rely on the definition of the early semantics in Tab. 1 as well as on the analysis in Tab. 2. The result below applies to *all* solutions of the analysis, and hence in particular to the least. Note that item *(3b)* corresponds to "bound" input, mainly intended to be used to match a corresponding bound output in the rule *Close* of the semantics; therefore the name y read along link x must be fresh in P.

Theorem 10. *If* $me[\mathit{fn}(P)] \subseteq \mathcal{C}$, $(\rho, \kappa) \models_{me} P$ *and* $P \xrightarrow[\lambda]{\mu} Q$ *we have:*

(1) *if* $\mu = \tau$ *then* $\lambda = \epsilon$ *and*
$(\rho, \kappa) \models_{me} Q$ *and*
$me[\mathit{fn}(Q)] \subseteq \mathcal{C}$;

(2a) *if* $\mu = \overline{x}y$ *then* $\lambda = \epsilon$ *and*
$(\rho, \kappa) \models_{me} Q$,
$me[\mathit{fn}(Q)] \subseteq \mathcal{C}$, *and*
$\rho(me(y)) \subseteq \cap_{\chi' \in \rho(me(x))} \kappa(\chi')$;

(2b) *if* $\mu = \overline{x}(y)$ *then* $\lambda = \chi$ *for some* $\chi \in \mathcal{C}$ *and*
$(\rho, \kappa) \models_{me[y \mapsto \chi]} Q$,
$(me[y \mapsto \chi])[\mathit{fn}(Q)] \subseteq \mathcal{C}$, *and*
$\rho(\chi) \subseteq \cap_{\chi' \in \rho(me(x))} \kappa(\chi')$;

(3a) *if* $\mu = xy$, $\lambda = \epsilon$, $me(y) \in \mathcal{C}$ *and* $\rho(me(y)) \subseteq \cap_{\chi' \in \rho(me(x))} \kappa(\chi')$ *then*
$(\rho, \kappa) \models_{me} Q$, *and*
$me[\mathit{fn}(Q)] \subseteq \mathcal{C}$;

(3b) *if* $\mu = xy$, $\lambda = \chi$, $\rho(\chi) \subseteq \cap_{\chi' \in \rho(me(x))} \kappa(\chi')$, *and* $y \notin \mathit{fn}(P)$ *then*
$(\rho, \kappa) \models_{me[y \mapsto \chi]} Q$, *and*
$(me[y \mapsto \chi])[\mathit{fn}(Q)] \subseteq \mathcal{C}$.

Proof. A lengthy proof by induction on the construction of $P_1 \xrightarrow{\mu} P_2$ and with subcases depending on whether case *(1)*, *(2a)*, *(2b)*, *(3a)* or *(3b)* applies. The proof makes use of Lemmas 2, 3, 5, 6, 8 and 9.

The complex formulation of this result, in particular the cases *(3a)* and *(3b)*, mainly depend on the fact that we use an early semantics; however, we conjecture that a somewhat analogous result with a fairly simpler statement also holds for the late semantics. In fact, most of the inclusions occurring in the above items constrain the environment where the process under validation operates. When the environment is fixed, they become useless. In a sense, this is the case when we use the late semantics, and a fortiori when we only have closed systems, i.e. processes that only perform τ moves.

4 Security properties

Control Flow Analysis helps in statically detecting information on security. We consider below a static property ensuring that a channel, devised to be secret to a process P, is never communicated to an external user. Other properties statically checkable will be discussed briefly in the Conclusions.

The scoping rules of the π-calculus permit a responsible use of channels of processes. Two processes P and Q can communicate along a channel a if a is known to both of them, i.e. if $a \in \mathit{fn}(P) \cap \mathit{fn}(Q)$. For example, if a is a public link to some information on the web home page of P, Q can access this information through a, by performing a communication. Additionally, P can communicate to Q one of its private links, say x, previously introduced by a restriction (νx^{χ}). This fact is modelled by including Q in the scope of x. This requires P to perform a bound output, e.g. on channel a, of the form $\overline{a}(x)$ (also called *extrusion* of name x), to deduce a communication and then to re-establish the restriction, using rule *Close*. A process P could have the security requirement of keeping secret (some of) its private channels. A process matches this requirement if it never performs an extrusion of a secret channel (deduced with an *Open* rule), as made precise below. In the following, we assume that $\mathcal{S}$ is a set of *secret* channels given by the designer or by the user of a process P, who implicitly introduces also the set $\mathcal{P}$ of *public* channels as the complement $\mathcal{C} \setminus \mathcal{S}$ of $\mathcal{S}$. As expected, binders are neither public nor secret. We also assume that the markers of names coming from the external environment are all public.

Definition 11. The pair P, me is *admissible* for the set $\mathcal{S} \subseteq \mathcal{C}$ of *secret* channels, if $me[\mathit{fin}(P)] \subseteq \mathcal{C}$ and $me[\mathit{fin}(P)] \cap \mathcal{S} = \emptyset$. Then, the set of *public* channels is $\mathcal{P} = \mathcal{C} \setminus \mathcal{S}$. $\diamondsuit$

Note that the condition in Def. 11 is equivalent to $me[\mathit{fin}(P)] \subseteq \mathcal{P}$, i.e. all free names are public.

Now, we would like to characterize a process P that never discloses its secrets. We describe this property by saying that P *has no leaks*. Intuitively P enjoys this property, if, during its computations, neither it nor any of its derivatives can perform an extrusion of a name bound to a secret channel.

Actually, we use in the following a constrained notion of computation, that we call *censored*. Essentially, a computation is censored if no name y, with $me(y) \in \mathcal{S}$, can be *read* from the environment through a free input. This assumption amounts to saying that an external user cannot guess any secret channel.

Definition 12. Given $P, me_P, \mathcal{S}$ a *censored step* is $(P, me_P) \overset{\mu}{\underset{\lambda}{\Longrightarrow}} (Q, me_Q)$, with $me_Q =$ $\begin{cases} me_P & \text{if } \lambda = \epsilon \\ me_P[obj(\mu) \mapsto \chi] & \text{if } \lambda = \chi \end{cases}$ and is defined whenever

1. $P \overset{\mu}{\underset{\lambda}{\longrightarrow}} Q$, and
2. if $\mu = xy$, then $\begin{cases} (a)\ me_P(y) \in \mathcal{P} & \text{if } \lambda = \epsilon \\ (b)\ \lambda \in \mathcal{P} \text{and } y \notin \mathit{fn}(P) & \text{if } \lambda = \chi. \end{cases}$

A *censored computation* $(P, me_P) \Longrightarrow^* (Q, me_Q)$ is made of censored steps. $\diamondsuit$

The following proposition shows that admissibility is preserved under censored computations, provided that there is no extrusion of a name marked secret. It also reveals the rôle played by the second condition of a censored step.

Proposition 13. *Let P, me_P be admissible for $\mathcal{S}$ and $(P, me_P) \overset{\mu}{\underset{\lambda}{\Longrightarrow}} (Q, me_Q)$ be such that $\mu = \overline{x}(y)$ implies $\lambda = \chi \in \mathcal{P}$. Then Q, me_Q are admissible for $\mathcal{S}$.*

We are ready to define our dynamic notion of security for P, me that are admissible for the set $\mathcal{S}$ of secret channels.

Definition 14. The process P has a *no leaks* with respect to $\mathcal{S}, me_P$ if and only if $(P, me_P) \Longrightarrow^* (Q, me_Q)$ implies that there is no pair (R, me_R) such that: $(Q, me_Q) \overset{\overline{x}(y)}{\underset{\chi}{\Longrightarrow}} (R, me_R)$, with $\chi \in \mathcal{S}$. $\diamondsuit$

Of course when P is stuck, it has no leaks. Note that if P has no leaks, with respect to $\mathcal{S}, me_P$, then for all Q such that $(P, me_P) \Longrightarrow^* (Q, me_Q)$, Q, me_Q are admissible for $\mathcal{S}$, due to Proposition 13.

The notion of no leaks above is dynamic. We now introduce a static notion, in order to predict at compile time if a process has no leaks. It is called *confinement*, and it is sufficient for a process to have no leaks.

Definition 15. Let P, me be admissible for given $\mathcal{S}$. A process P is *confined* with respect to $\mathcal{S}, me$ if and only if $\exists\ (\rho, \kappa)$ such that

$$(a)\ (\rho, \kappa)\ \models_{me} P \qquad \text{and} \qquad (b)\ \kappa(\chi) = \begin{cases} \mathcal{C} \text{ if } \chi \in \mathcal{S} \\ \mathcal{P} \text{ if } \chi \in \mathcal{P} \end{cases}$$

Hereafter, we will say that P is confined *via the confining* solution (ρ, κ). $\diamondsuit$

Intuitively, condition (b) above implies that only public information can be transmitted along a public channel, i.e. $\bigcup_{\chi \in \mathcal{P}} \kappa(\chi) \subseteq \mathcal{P}$; conversely, any channel, be it secret or public, can pass along a secret channel. In fact for part of our development we could use a weaker notion of confinement, where the equality in (b) is replaced by "$\subseteq$". However the notion chosen is easier to check automatically.

We now show that confinement is preserved by censored computations. Recall that, by Def. 12, me_Q below is $me_P[obj(\mu) \mapsto \chi]$, if $\lambda = \chi$ and is me_P otherwise.

Lemma 16 Subject reduction for confinement.

If P confined (with respect to $\mathcal{S}, me_P$), $(P, me_P) \overset{\mu}{\underset{\lambda}{\Longrightarrow}} (Q, me_Q)$ and $me_P[fin(P)] \subseteq \mathcal{P}$; then Q confined (with respect to $\mathcal{S}, me_Q$) and $me_Q[fin(Q)] \subseteq \mathcal{P}$.

Proof. If $\mu = \tau$ or $\overline{x}y$ or $\overline{x}(y)$, Theorem 10 and the hypotheses suffice to establish $(\rho, \kappa)\ \models_{me_Q} Q$. When $\mu = xy$, there are two cases. If $\lambda = \epsilon$, $me_P(y) = \chi \in \mathcal{P}$, because of condition (a) of censored step. Now, $\rho(\chi) = \{\chi\} \subseteq \kappa[\rho(me_P(x))] = \kappa[\rho(\chi')] = \kappa(\chi') = \mathcal{P}$, for some $\chi' \in \mathcal{C}$, because $x \in fin(P)$, otherwise the transition cannot be deduced (see comment before the Lemma 3). Hence $me_P(x) = \chi' \in \mathcal{P}$. Similarly for the case where $\lambda = \chi$.
By Lemma 3 and by the definition of censored step, we only need to check that if $\mu = \overline{x}(y)$ then $me_Q(y) \in \mathcal{P}$. We have that $me_Q(x) = me_P(x) = \chi' \in \mathcal{P}$ and since P is confined, also $\kappa(\chi') = \mathcal{P}$, hence $me_Q(y) \in \mathcal{P}$.

We are ready to show that confinement is sufficient to guarantee that P has no leaks.

Theorem 17. *P confined, with respect to $\mathcal{S}, me_P$, implies P has no leaks, with respect to $\mathcal{S}, me_P$.*

Proof. By Lemma 16, it is enough to prove that $(P, me_P) \xrightarrow[\chi]{\overline{x}(y)} (Q, me_Q)$, with $\chi \in \mathcal{S}$, never occurs. Suppose per absurdum that it does. Then $me_P(x) = \chi' \in \mathcal{S}$, because P is confined (otherwise, since $\chi \in \kappa(\chi')$ condition (b) of confinement would be violated). But $x \in \mathit{fin}(P)$, because otherwise the transition cannot be deduced (see comment before the Lemma 3). Contradiction, because of admissibility and because $me_P[\mathit{fin}(P)] \cap \mathcal{S} \supseteq \{\chi'\}$.

Consider again the process P validated before sub-section 3.2:

$$P = a(x^{\beta_0}).(\nu b^{\chi_0})(\nu c^{\chi_1})((\overline{b}a.\overline{x}x.b(x^{\beta_1}).\overline{x}c \; + \; \overline{b}d.\overline{a}c) \mid b(x^{\beta_2}).\overline{b}x) \mid d(x^{\beta_3}).$$

If we take as secret channels $\mathcal{S} = \{\chi_0\}$ (hence $\mathcal{P} = \{\chi_1, \chi_2, \chi_3, \chi_4\}$), then P is confined via the solution (ρ, κ) provided there. Indeed, κ satisfies condition *(b)* in Def. 15. Notice that $(\rho, \kappa) \models_{me'} \overline{x}c$, for $me' = (me[x \mapsto \beta_0])[x \mapsto \beta_1, b \mapsto \chi_0, c \mapsto \chi_1]$, because

$$\forall \chi \in \rho(me'(x)) = \rho(\beta_1) = \{\chi_0, \chi_1, \chi_2, \chi_3, \chi_4\} : \rho(me'(c)) = \{\chi_1\} \subseteq \kappa(\chi)$$

On the contrary, if we take as secret channels the set $\mathcal{S}' = \{\chi_0, \chi_1\}$ (and $\mathcal{P}' = \{\chi_2, \chi_3, \chi_4\}$), then P is *not* confined. In fact, the new κ' imposed by condition *(b)* in Def. 15 is such that $\kappa'(\chi_i) = \mathcal{P}$ for $i \neq 0, 1$, hence $\mathcal{S}' \ni \chi_1 \notin \kappa'(\chi_i)$, for $i \neq 0, 1$. For example $(\rho, \kappa') \not\models_{me'} \overline{x}c$. Actually, $(\rho', \kappa') \not\models_{me} P$, for any choice of ρ'. The analysis detects that $\overline{x}c$ is dangerous, because x can be bound to a at run time, possibly giving rise to the term $((\nu b^{\chi_0})(\nu c^{\chi_1})\overline{a}c) \mid d(x^{\beta_3})$, which can extrude c.

It is immediate to see that confinement is not a necessary condition for P having no leaks. For instance the process $(\nu x^{\chi})\overline{x}y.\overline{y}x$ has no leaks but it is not confined with respect to $\mathcal{S} = \{\chi\}$. Indeed not all deadlocks can be detected by the static analysis. So the extrusion of the name x along channel y is considered a possible violation of secrecy.

We now determine the extent to which confinement is preserved by syntactic operators.

Proposition 18. *Let P be confined via (ρ, κ) with respect to $\mathcal{S}, me$. Then*

1. *$\tau.P$ and* **2.** *$!P$ are confined with respect to $\mathcal{S}, me$;*

3. *$\overline{x}y.P$ is confined with respect to $\mathcal{S}, me[x \mapsto \chi, y \mapsto \chi']$, for all $\chi, \chi' \notin \mathcal{S}$, provided that $x \in \mathit{fn}(P)$ implies $me(x) = \chi$ and $y \in \mathit{fn}(P)$ implies $me(y) = \chi'$;*

4. *$(\nu x^{\chi})P$ is confined with respect to $\mathcal{S}$, me for all $\chi \in \mathcal{C}$, provided that $x \in \mathit{fn}(P)$ implies $me(x) = \chi$;*

5. *$[x = y]P$ is confined with respect to $\mathcal{S}, me[x \mapsto \chi, y \mapsto \chi']$ for all $\chi, \chi' \in \mathcal{C}$, provided that $x \in \mathit{fn}(P)$ implies $me(x) = \chi$ and $y \in \mathit{fn}(P)$ implies $me(y) = \chi'$.*

Furthermore, let Q be confined via (ρ, κ) with respect to $\mathcal{S}, me$, then

6. *$P + Q$ and* **7.** *$P|Q$ are confined with respect to $\mathcal{S}, me$.*

Proof. **1.** $\tau.P$, **2.** $!P$, **4.**$(\nu x^{\chi})P$, and **5.** $[x = y]P$ are trivially confined;

3. $\overline{x}y.P$. We prove that $(\rho, \kappa) \models_{me'} \overline{x}y.P$, with $me' = me[x \mapsto \chi, y \mapsto \chi']$. The second conjunct in Tab. 2 is trivially satisfied, because by hypothesis $\kappa(\chi) = \mathcal{P}$, thus $\kappa(\chi) \supseteq \rho(me'(y)) = \{\chi'\}$. For proving that $(\rho, \kappa) \models_{me'} P$, it is sufficient to see that $\rho(me'(z)) = \rho(me(z))$, for all $z \in fn(P)$ and then use Lemma 3.

6. $[P + Q]$ and **7.** $[P|Q]$ follow immediately by rules in Tab. 2.

Confinement would seem not to be fully compositional, in particular, in the case of input. Again, this is because early input is very permissive, in that it allows any value to be read from the environment. We reduced this freedom somehow by considering censored steps only, i.e. by assuming that the environment cannot guess secrets. We conjecture that confinement is fully compositional for closed processes, or when a late operational semantics is used in place of the early one.

5 Conclusions and Future Work

We presented a Control Flow Analysis for the π-calculus that statically predicts how names will be bound to actual channels at run-time. The only extensions made to the syntax of processes are that a channel χ is explicitly assigned to a restricted name, and that an input action has the form $x(y^{\beta})$, making explicit the rôle of the placeholder y. The result of our analysis for a process P is a solution (ρ, κ). The abstract environment ρ gives information about which channels a binder β may be bound to, by means of communication. The abstract channel cache κ says which channels may pass along a given channel. Both ρ and κ approximate the actual solution, because they may give a super-set of the corresponding actual values.

Lack of space prevented us from presenting a constructive procedure for generating solutions, which we proved to exist in any case. We only defined judgments of the form $(\rho, \kappa) \models_{me} P$ and a set of clauses that operate on them and validate the correctness of the solution. The additional marker environment me binds the free names of P to actual channels. The other names occurring in P either have assigned an explicit channel, if they are restricted, or have a binder β to which ρ assigns a set of channels. When considering only closed systems or using the late semantics, our analysis and the statement of its correctness considerably simplify.

We described how our analysis can be used to establish a simple property of security, that we called confinement. Given a set $\mathcal{S}$ of channels to be kept secret, a process confined will never communicate any channels in $\mathcal{S}$ to the environment. The dynamic notion is akin to having no leaks, because an external process never reaches a secret information. We did not present here an extension to this two-level notion of confinement that consists of a hierarchy of classification levels associated with channels. A static property similar to confinement then ensures that a high level information is never transmitted on a channel with a lower level of classification.

In the full paper we will present another security property, in the style of [3]. This requires to extend the syntax to assign classification levels also to processes, which can only read data classified at or below their own level, and can only write data classified at or above their own level. The extension to syntax calls for an extension to the analysis, and judgments will have the form $(\rho, \kappa, \sigma) \models^{l}_{me} P$. The additional label

l records the classification level of P, and σ gives information about the directionality of channels. Variants of this property have been studied in a dynamic setting (see, e.g. [6, 7, 10], to cite only a few).

Closer to our proposal is the use of type systems made in [1, 15, 17, 8, 2, 5, 16, 20, 4] in order to guarantee security properties. In particular, Abadi's paper [1] studies secrecy of channels and of encrypted messages in the spi-calculus. This is a more ambitious goal than ours, because in the π-calculus there is no notion of encryption. As for the disciplined use of channels, Abadi's and our aims are very close, as well as for the assumptions made (except that Abadi has a further notion of *Any*, besides those of *Public* and *Secret*). We conjecture that the two approaches are comparable in precision. Indeed our solution (ρ, κ) can be seen as an explicit type annotation of processes. Our notion of validation corresponds to validate the type annotation. However, whereas we have established that a best solution always exists (via Moore families), [1] does not establish the existence of principal types, and it is unclear whether a sound and complex typing algorithm does exists. The semantic correctness is checked against two different dynamic notions (testing equivalence versus no leaks).

Acknowledgments
We wish to thank the referees for their precise comments and suggestions, and Alan Mycroft for the word "censored". The first two authors have been partially supported by the CNR Progetto Strategico *Modelli e Metodi per la Matematica e l'Ingegneria* and by the MURST Progetto *Tecniche Formali per la Specifica, l'Analisi, la Verifica, la Sintesi e la Trasformazione di Sistemi Software*. The last two authors have been partially supported by the DART project (Danish Science Research Council).

References

1. M. Abadi. Secrecy by typing in security protocols. In *Proceedings of Theoretical Aspects of Computer Software, Third International Symposiuoum, LNCS 1281*, pages 611–638. Springer-Verlag, 1997.
2. R.M. Amadio. An asynchronous model of locality, failure, and process mobility. In *Proceedings of COORDINATION'97, LNCS 1282*. Springer-Verlag, 1997.
3. D.E. Bell and L.J. La Padula. Secure computer systems: Unified exposition and multics interpretation. Technical Report ESD-TR-75-306, Mitre C., 1976.
4. M. Dam. Proving trust in systems of second-order processes: Preliminary results. Technical Report LOMAPS-SICS 19, SICS, Sweden, 1997.
5. R. De Nicola, G. Ferrari, and R. Pugliese. Coordinating mobile agents via blackboards and access rights. In *Proceedings of COORDINATION'97, LNCS 1282*. Springer-Verlag, 1997.
6. R. Focardi. Comparing two information flow security properties. In *Proceedings of* 9^{th} *IEEE Computer Science Security Foundation W/S*, pages 116–122, 1996.
7. R. Focardi and R. Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*. To appear.
8. C. Fournet, C. Laneve, L. Maranget, and D. Remy. Implicit typing à la ML for the join calculus. In *Proceedings of CONCUR'97, LNCS*. Springer-Verlag, 1997.

9. K.L.S. Gasser, F. Nielson, and H.R. Nielson. Systematic realisation of control flow analysis for CML. In *Proceedings of ICFP'97*, pages 38–51. ACM Press, 1997.
10. J.A. Goguen and J. Meseguer. Security policy and security models. In *Proceedings of the 1982 IEEE Symposioum on Security and Privacy*, pages 11–20, 1982.
11. T.J. Marlowe and B.J. Ryder. Properties of data flow frameworks - a unified model. *Acta Informatica*, 28(2):121–163, 1990.
12. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (I and II). *Information and Computation*, 100(1):1–77, 1992.
13. R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *TCS*, 114:149–171, 1993.
14. J. Palsberg and M.J. Schwartzbach. *Object-Oriented Type Systems*. Wiley, 1994.
15. B.C. Pierce and D. Sangiorgi. Typing and sub-typing for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996.
16. J. Riely and M. Hennessy. A typed language for distributed mobile processes. In *Proceedings of POPL'98*. ACM Press, 1998.
17. P. Sewell. Global/local subtyping for a distributed π-calculus. Technical Report 435, Computer Laboratory, University of Cambridge, 1997.
18. O. Shivers. Control flow analysis in Scheme. In *Proceedings of PLDI'88*, volume 7(1). ACM SIGPLAN Notices, 1988.
19. A. Venet. Abstract interpretation of the π-calculus. In *Analysis and Verification of Multiple-Agent Languages, LNCS*, volume 1192. Springer-Verlag, 1997.
20. D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4:4–21, 1996.