Information and Computation **165**, 68–92 (2001)
doi:10.1006/inco.2000.3020, available online at http://www.idealibrary.com on IDEAL®

Static Analysis for the π-Calculus with Applications to Security

Chiara Bodei and Pierpaolo Degano

Dipartimento di Informatica, Università di Pisa, Corso Italia, 40, I-56125 Pisa, Italy
E-mail: {chiara, degano}@di.unipi.it

and

Flemming Nielson and Hanne Riis Nielson

Informatics and Mathematicse Modelling, The Technical University of Denmark, Richard Petersens Plads, blg 321, DK 2800 Kongens Lyngby, Denmark
E-mail: nielson@imm.dtu.dk, riis@imm.dtu.dk

October 14, 1998; received January 15, 2000

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 result of our analysis establishes a super-set of the set of channels to which a given name may be bound and of the set of channels that may be sent along a given channel. Besides a set of rules that permits one to validate a given solution, we also offer a constructive procedure that builds solutions in low polynomial time. Applications of our analysis include establishing two simple security properties of processes. One example is that P has *no leaks*: P offers communication to the external environment through public channels only and confines its secret channels within itself. The other example is connected to the *no read-up/no write-down* property of Bell and LaPadula: once processes are given levels of security clearance, we check that a process at a high level never sends channels to processes at a lower level. © 2001 Academic Press

1. INTRODUCTION

Program analysis aims at verifying 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. More recently, program analysis has been used for validating security and safety issues for concurrent and distributed systems.

Program analysis provides automatic and decidable methods for analysing properties of programs. Since most properties implicitly involve questions about termination, the methods 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.

We wish to study these issues for concurrent languages. To investigate them in a pure form we shall use the π-calculus which is a model of concurrent communicating processes based on name passing. Names may represent both data and channels that processes exchange. For example, if a is the name of a link to some information on the web home page of a user, then another user can access this information through a, by performing a communication. We propose in Section 3 a Control Flow Analysis for the π-calculus that requires only minor additions to the syntax: assigning explicit "channels" to the names occurring in restrictions and assigning explicit "binders" to the names occurring in input prefixes. This may be compared to the approach of [41], where processes are required to be on a special form. Roughly, channels can be seen as representatives of semantic values that names may have, and binders as the actual placeholders in input prefixes. We review in Section 2 the syntax and the early semantics of the π-calculus, and we introduce our annotations.

The result of our control flow analysis establishes a super-set of the set of channels to which a given name may be bound and of the set of channels that may be sent along a given channel. These super-sets give rise to *solutions* (ρ, κ) and we formulate the control flow analysis as a specification of the correctness of a candidate solution. This takes the form of a Flow Logic with judgements $(\rho, \kappa) \models_{me} P$ (where *me* is an auxiliary function that associates channels or binders with the free names of the process P), and a set of clauses that operate on them. We show that best solutions always exist and we establish the semantic correctness of solutions in the form of a subject-reduction result. In Section 4 we then present a procedure that generates solutions by inducing on the structure of processes, and operates in $O(N^5)$ time with respect to the size N of the process under analysis.

We apply our analysis for statically checking two simple security properties. The first property is in Section 5 and considers channels as divided into “secret” and “public” channels. Then, the dynamic security requirement is that secret information may only be communicated over secret channels; in other words, a process has no leaks of secret information. With simple checks on a solution, we obtain a static test (called *confinement*) for a given process having no leaks, and we prove it safe with respect to the dynamic notion (called *no leaks*).

The second property presented in Section 6 is the *simple security* property that is part of the multi-level security property (“no read-up/no write-down”) of Bell and LaPadula [5]. Processes are given levels of security clearance, and the dynamic property demands that those at high level never send information to those at low level, while communication in any other direction is permitted. A little extension to our machinery is sufficient to define a static check (called *discreetness*) for when a process respects the classification hierarchy, and to prove it safe with respect to the dynamic notion (called *nru/nwd*).

Finally, we briefly discuss in Section 7 some related work on static analysis and security properties, and in Section 8 we discuss some other uses of control flow analysis for concurrent processes.

2. THE π-CALCULUS

Syntax

In this section we briefly recall the π-calculus [29], a model of concurrent communicating processes based on the notion of *name passing*.

DEFINITION 2.1. Let $\mathcal{N}$ be an infinite set of names ranged over by $a, b, \dots, x, y, \dots$ and let τ be a distinguished element such that $\mathcal{N} \cap \{\tau\} = \emptyset$. Processes are built from names according to the syntax

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

where μ may either be $x(y)$ 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 μ instead of $\mu.\mathbf{0}$). We assume that $+$ has lower precedence than $|$ which again has lower precedence than the other operators.

The prefix μ is the first atomic action that the process $\mu.P$ can perform. The input prefix $x(y)$ binds the name y in the prefixed process. Intuitively, some name y is received along the link named x. The output prefix $\overline{x}y$ does not bind the name y which is sent along x. The silent prefix τ denotes an action which is invisible to an external observer of the system. Summation denotes nondeterministic choice, so $P + Q$ behaves either as P or as Q. The operator $|$ describes parallel composition of processes. Intuitively, P and Q in $P \mid Q$ act independently and can also communicate when one performs an input and the other an output on the same common link. The restriction operator $(\nu x)P$ acts as a declaration for the name x in the process P that it prefixes. In other words, x is a unique name in P which is different from all the external names. The agent $(\nu x)P$ behaves as P except that sending and receiving along $\overline{x}$ and x is blocked. A distinguished feature of the π-calculus is to allow for an enlargement of the scope of a restriction; we will expand on this below. Matching $[x = y]P$ is an if–then operator: process P is activated if $x = y$. Finally, replication $!P$ behaves as $P|P| \cdots$ as many times as needed.

The formulation of our analysis requires only a minor extension to the syntax of the π-calculus, namely annotating the binding occurrences of names within restrictions with "channels" χ, and the binding occurrences of names within input prefixes with "binders" β. These syntactic extensions are needed because of the α-conversion allowed by the structural congruence. They do not affect the dynamic semantics of the π-calculus; however, they heavily influence the way in which our static analysis of Section 3 operates. From the point of view of the analysis, annotations place all the names of a process in a finite set of equivalence classes. Through them, the analysis computes (a super-set of) the actual links that a name can denote.

Definition 2.2. Let $\mathcal{B}$ be a nonempty set of binders ranged over by $\beta, \beta', \ldots$; and let $\mathcal{C}$ be a non emptyset of channels ranged over by $\chi, \chi', \ldots$ such that $\mathcal{B} \cap \mathcal{C} = \emptyset$; moreover call $\mathcal{B} \cup \mathcal{C}$ the set of markers. Then (annotated) processes, denoted by $P, P_1, P_2, Q, R, \ldots \in \mathit{Proc}$ are built as in Definition 2.1, where the (annotated) input prefix $x(y^\beta)$ replaces $x(y)$ and the (annotated) restriction $(\nu x^\chi)P$ replaces $(\nu x)P$.

Semantics

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, because it is emerging as a standard for transitional semantics and appears to be more suitable for the security issues studied in the next sections. We follow [30], in particular for the distinction between free and bound input.

The labels of transitions are τ for silent actions, xy for free input, $\overline{x}y$ for free output, $x(y^\chi)$ for bound input and $\overline{x}(y^\chi)$ for bound output. Roughly speaking, the effect of a bound output is moving a (νx^χ) operator from a process to a label, as in $Q = (\nu y^\chi)\overline{x}y.P \xrightarrow{\overline{x}(y^\chi)} P$. The intuition behind this operation is to make the name y, which is private to Q, available to the external environment. The bound output then enlarges the scope of the declaration, and for this reason it is sometimes referred to as *scope extrusion* in the literature. When coupled with a bound input $x(y^\chi)$, the extrusion originates a communication and reestablishes the removed restriction.

As usual, 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. The *subject* of an input or output action is the channel (x) used for the communication and the *object* is the entity (y) being transmitted.

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

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

Congruence

Below we shall need the *structural congruence* $\equiv$ on processes, defined as in [30] 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 $z \notin fn(x(y^\beta)P)$;
- $(\mathit{Proc}/_\equiv, +, \mathbf{0})$ and $(\mathit{Proc}/_\equiv, |, \mathbf{0})$ are commutative monoids;
- $(\nu x^\chi)(\nu y^{\chi'})P \equiv (\nu y^{\chi'})(\nu x^\chi)P$, if $x \neq y$, $(\nu x^\chi)(P_1 \mid P_2) \equiv (\nu x^\chi)P_1 \mid P_2$ if $x \notin fn(P_2)$, and $(\nu x^\chi)P \equiv P$ if $x \notin fn(P)$;
- $!P \equiv P \mid !P$.

TABLE 1

Early Transition System for the π-Calculus

Tau: $\tau.P \xrightarrow{\tau} P$	*Out*: $\bar{x}y.P \xrightarrow{\bar{x}y} P$
FreeIn: $x(y^{\beta}).P \xrightarrow{xw} P\{w/y\}$	*Bound In*: $x(y^{\beta}).P \xrightarrow{x(y^{x})} P$
Par: $\dfrac{P_1 \xrightarrow{\mu} Q_1}{P_1 \mid P_2 \xrightarrow{\mu} Q_1 \mid P_2}, bn(\mu) \cap fn(P_2) = \emptyset$	*Sum*: $\dfrac{P_1 \xrightarrow{\mu} Q_1}{P_1 + P_2 \xrightarrow{\mu} Q_1}$
Res: $\dfrac{P \xrightarrow{\mu} Q}{(\nu x^{\chi})P \xrightarrow{\mu} (\nu x^{\chi})Q}, x \notin n(\mu)$	*Open*: $\dfrac{P \xrightarrow{\bar{x}y} Q}{(\nu^{\chi})P \xrightarrow{\bar{x}(y^{\chi})} Q}, y \neq x$
Close: $\dfrac{P_1 \xrightarrow{\bar{x}(y^{\chi})} Q_1, P_2 \xrightarrow{x(y^{\chi})} Q_2}{P_1 \mid P_2 \xrightarrow{\tau} (\nu y^{\chi})(Q_1 \mid Q_2)}$	*Com*: $\dfrac{P_1 \xrightarrow{\bar{x}(y)} Q_1, P_2 \xrightarrow{x(y)} Q_2}{P_1 \mid P_2 \xrightarrow{\tau} Q_1 \mid Q_2}$
Var: $\dfrac{P' \equiv P \xrightarrow{\mu} Q \equiv Q'}{P' \xrightarrow{\mu} Q'}$	*Match*: $\dfrac{P \xrightarrow{\mu} Q}{[x{=}x]P \xrightarrow{\mu} Q}$

Note that α-conversions do not affect markers. Also, we permit exchange restrictions only when the restricted names are different, because otherwise $(\nu x^{\chi})P \equiv (\nu x^{\chi'})P$ and the marker then loses its identity.

Table 1 shows the (annotated) *early* transition system of the π-calculus defined in SOS style.

A different treatment of matching is presented in [7]. There, the structural congruence law $[x = x]P \equiv P$ is assumed and the transitional rule *Match* is removed from Table 1. This latter change requires an accurate handling of free names, otherwise applying the congruence rule from right to left may introduce new free names ad libitum. The need of handling similar kinds of low level details is a recurrent problem in congruence-based semantics, and in [7] we illustrate one of the techniques needed to deal with them.

We conclude this section with a straightforward fact that will be repeatedly used in the proofs later on.

Fact 2.3. If $P \xrightarrow{\mu} Q$ then

(1) If $\mu = \tau$ then $fn(P) \supseteq fn(Q)$.

(2) If $\mu = \bar{x}y$ then $fn(P) \supseteq \{x, y\} \cup fn(Q)$.

(3) If $\mu = \bar{x}(y^{\chi}), xy, x(y^{\chi})$ then $fn(P) \supseteq \{x\} \cup (fn(Q) \backslash \{y\})$.

3. CONTROL FLOW ANALYSIS

The result of analysing a process P is a pair (ρ, κ). The first component, ρ, is an abstract environment which gives information about the set of channels to which names can be bound; the second component, κ, is an abstract channel environment which gives information about the set of channels that can flow over given channels.

One way to view the pair (ρ, κ) is as a record of the actual communications taking place during it execution. Whenever a value a_{val} is output on some channel b_{chan}, as in $\overline{b_{chan}}a_{val}$, it must be duly recorded in the κ component, intuitively by ensuring that $a_{val} \in \kappa(b_{chan})$. Similarly, whenever a variable c_{var} inputs a value on some channel b_{chan}, as in $b_{chan}(c_{var})$, this must also be duly recorded in the ρ component, intuitively by ensuring that $a_{val} \in \rho(c_{var})$ for all $a_{val} \in \kappa(b_{chan})$.

We now make this more precise (also paying attention to an additional marker environment *me* for associating names with markers).

3.1. *Validation*

To *validate* the correctness of a proposed solution (ρ, κ) we state a set 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 maps a name (in particular the free names of a process) to the appropriate channel or binder used when 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 maps a binder to the set of channels that it can be bound to; more precisely, $\rho(\beta)$ must include the set of channels that β could evaluate to. We shall allow one to regard the abstract environment as a function $\rho : (\mathcal{B} \cup \mathcal{C}) \to \wp(\mathcal{C})$ by setting $\forall \chi \in \mathcal{C} : \rho(\chi) = \{\chi\}$.

  We write $\perp$ for the function that maps everything to $\emptyset$. However, we continue to assume that $\forall \chi \in \mathcal{C} : \perp(\chi) = \{\chi\}$.
- $\kappa : \mathcal{C} \to \wp(\mathcal{C})$ is the *abstract channel environment* that maps a channel to the set of channels that can be communicated over it.

  More precisely, $\kappa(\chi)$ must include the set of channels that can be communicated over the channel χ. Also, here we write $\perp$ for the function that maps everything to $\emptyset$.

Note that we use a marker environment, because the identity of names is not preserved under α-conversions (see the rule *Var*). Indeed, it would not suffice to "α-rename the program apart" because this property is not preserved under reduction, in particular when scope extrusion is required. For example, the process $(\nu a^{\chi'})(a(y^{\beta}).a(z^{\beta'}).\overline{y}z \mid !(\nu x^{\chi})\overline{a}x)$ performs a first communication, then α-converts the name x to perform a second communication and becomes $(\nu a^{\chi'})(\nu x^{\chi})(\nu w^{\chi})(\overline{x}w \mid !(\nu x^{\chi})\overline{a}x)$.

A further comment on annotations may clarify their subsequent use. A typical schema for annotating the occurrences of restricted names and of objects of inputs in a process P is to keep all the χ's and the β's distinct; also, the marker environment *me* should map the free names of P to fresh channels. Note that annotating a process in this way is merely mechanical and involves no knowledge about its behaviour.

The detailed definition of our control flow analysis is given by the flow logic in Table 2, where we often write *me*$[x \mapsto \eta]$ to indicate that the *me* is updated with the new association of the name x with the marker η, overwriting a possible previous association. All the rules dealing with a compound process require that the components are validated, apart from the one for matching. Moreover, the second conjunct of the rule for output requires that the set of channels that can be communicated along each element of $\rho(me(x))$ includes the channels to which y can evaluate. Symmetrically, the rule for input demands that the set of channels that can pass along x is included in the set of channels to which y can evaluate. In the clause for restriction, we can simply update the marker environment as *me*$[x \mapsto \chi]$ because $\rho(\chi) = \{\chi\}$ by definition. The condition for matching says that the continuation P needs to be validated if there is at least one channel to which both x and y can evaluate.

EXAMPLE 3.1. 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 $i \in \{0, 1, 2, 3, 4\}$:

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

TABLE 2

Control Flow Analysis for the π-Calulus

$(\rho, \kappa) \models_{me} \mathbf{0}$	iff *true*
$(\rho, \kappa) \models_{me} \tau.P$	iff $(\rho, \kappa) \models_{me} P$
$(\rho, \kappa) \models_{me} \overline{x}y.P$	iff $(\rho, \kappa) \models_{me} P \wedge \forall \chi \in \rho(me(x)) : \rho(me(y)) \subseteq \kappa(\chi)$
$(\rho, \kappa) \models_{me} x(y^{\beta}).P$	iff $(\rho, \kappa) \models_{me[y \mapsto \beta]} P \wedge \forall \chi \in \rho(me(x)) : \kappa(\chi) \subseteq \rho(\beta)$
$(\rho, \kappa) \models_{me} P_1 + P_2$	iff $(\rho, \kappa) \models_{me} P_1 \wedge (\rho, \kappa) \models_{me} P_2$
$(\rho, \kappa) \models_{me} P_1 \mid P_2$	iff $(\rho, \kappa) \models_{me} P_1 \wedge (\rho, \kappa) \models_{me} P_2$
$(\rho, \kappa) \models_{me} (\nu x^{\chi})P$	iff $(\rho, \kappa) \models_{me[x \mapsto \chi]} P$
$(\rho, \kappa) \models_{me} [x = y]P$	iff $(\rho(me(x)) \cap \rho(me(y)) \neq \emptyset \Rightarrow (\rho, \kappa) \models_{me} P$
$(\rho, \kappa) \models_{me} !P$	iff $(\rho, \kappa) \models_{me} P$

A simple check shows that $(\rho, \kappa) \models_{me} P$. The objects x of the inputs on channels a and d are kept distinct for the analysis, because the annotations β_0 and β_3 place them in two different equivalence classes (but this does not influence the dynamic semantics). The reader may have noticed that (ρ, κ) above is not the least solution, e.g., because of the presence of χ_4. This kind of useless channel may occur in solutions, although they do appear neither in annotations nor in the image of a marker environment—nor will they occur in the solutions constructed according to Section 4 (see also Theorem 3.4).

The formulation of our control flow analysis borrows from standard ideas for functional languages. Our current formulation is insensitive to flow and context [31], so terms can be rearranged without affecting the acceptability of a candidate solution; in effect, restrictions can be lifted to the top level, or to the nearest enclosing !, and prefixing of actions can be replaced by their parallel composition. While more complex flow analyses can be devised, these are not necessary for the applications to security studied here.

3.2. *Existence of Solutions*

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 Table 2.

Definition 3.2. The set of proposed solutions can be partially ordered by setting $(\rho, \kappa) \sqsubseteq (\rho', \kappa')$ iff $\forall \beta \in \mathcal{B} : \rho(\beta) \subseteq \rho'(\beta)$ 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; using standard notation 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 $(\bot, \bot)$ for the least element.

Definition 3.3. 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 for $\mathcal{J} = \mathcal{I}$).

This concept plays an important role in the theory of Abstract Interpretation [11, 31]; in other branches of computer science it 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 $(\sqcap\mathcal{I})$. The following theorem then guarantees that there always is a least solution to the specification in Table 2 (just take $(\overline{\rho}, \overline{\kappa}) = (\bot, \bot)$ in the statement below).

Theorem 3.4. *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.

Proof. We proceed by structural induction on P (since Table 2 is defined by structural induction on P). Let

$$\mathcal{J} \subseteq \{(\rho, \kappa) \mid (\rho, \kappa) \models_{m} P \wedge (\rho, \kappa) \sqsupseteq (\overline{\rho}, \overline{\kappa})\}$$

and let J and (ρ_j, κ_j) be given such that $\mathcal{J} = \{(\rho_j, \kappa_j) \mid j \in J\}$. Next define

$$(\rho', \kappa') = \sqcap\mathcal{J} = \sqcap\{(\rho_j, \kappa_j) \mid j \in J\}$$

and recall that the greatest lower bound is defined pointwise and hence that $(\rho', \kappa') \sqsupseteq (\overline{\rho}, \overline{\kappa})$. It remains to check that $(\rho', \kappa') \models_{me} P$. For this we proceed by cases on P making use of the induction hypothesis. Most cases are straightforward and here we only consider two of the more interesting cases.

The Case $x(y^\beta).P$. Since $\forall j \in J : (\rho_j, \kappa_j) \models_{me} x(y^\beta).P$ we have

$$\forall j \in J : (\rho_j, \kappa_j) \models_{me[y \mapsto \beta]} P \quad \text{and} \quad \forall j \in J : \forall \chi \in \rho_j(me)(x)) : \kappa_j(\chi) \subseteq \rho_j(\beta)$$

Using the induction hypothesis and that ρ' is defined in a pointwise manner, we then obtain

$$(\rho', \kappa') \models_{me[y \mapsto \beta]} P \quad \text{and} \quad \forall \chi \in \rho'(me(x)) : \kappa'(\chi) \subseteq \rho'(\beta)$$

thus establishing the desired $(\rho', \kappa') \models_{me} x(y^\beta).P$.

The Case $[x = y]\ P$. Since $\forall j \in J : (\rho_j, \kappa_j) \models_{me} [x = y]P$ we have

$$\forall j \in J : (\rho_j(me(x)) \cap \rho_j(me(y)) \neq \emptyset \Rightarrow (\rho_j, \kappa_j) \models_{me} P$$

Using the induction hypothesis and the pointwise definition of ρ', we then obtain

$$(\rho'(me(x)) \cap \rho'(me(y)) \neq \emptyset \Rightarrow (\rho', \kappa') \models_{me} P$$

thus establishing the desired $(\rho', \kappa') \models_{me} [x = y]P$. ■

3.3. *Correctness*

We state now some auxiliary results that will allow us to establish the semantic correctness of our analysis; they are all independent of the operational semantics and only rely on Table 2.

LEMMA 3.5. *Assume that* $\forall x \in fn(P) : me_1(x) = me_2(x)$; *then* $(\rho, \kappa) \models_{me_1} P$ *if and only if* $(\rho, \kappa) \models_{me_2} P$.

Proof. A straightforward structural induction on P. ■

LEMMA 3.6. *Assume that* $me(y) = me(z)$; *then* $(\rho, \kappa) \models_{me} P$ *if and only if* $(\rho, \kappa) \models_{me} P\{z/y\}$.

Proof. The proof is by induction on the size of P. Most cases are straightforward using the fact that $\forall x : me(x) = me(x\{z/y\})$. This leaves us with the cases where the marker environment is modified and here we consider only a typical case.

The Case $P = u(v^\beta).Q$. If $v = y$ the result follows from the above remarks so assume that $v \neq y$. Let w be a fresh name, i.e., let $w \notin fn(Q) \cup \{z, y\}$, in case $z = v$, and let $w = v$ in case $z \neq v$; in both cases $me[w \mapsto \beta](z) = me[w \mapsto \beta](y)$. Then

$$P\{z/y\} = u\{z/y\}(w^\beta).(Q\{w/v\}\{z/y\})$$

and it follows that

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

amounts to

$$(\rho, \kappa) \models_{me[w \mapsto \beta]} Q\{w/v\}\{z/y\} \quad \text{and} \quad \forall \chi \in \rho(me(u\{z/y\})) : \kappa(\chi) \subseteq \rho(\beta)$$

and by the induction hypothesis this amounts to

$$(\rho, \kappa) \models_{me[w \mapsto \beta]} Q\{w/v\} \quad \text{and} \quad \forall \chi \in \rho(me(u)) : \kappa(\chi) \subseteq \rho(\beta),$$

which by Lemma 3.5 amounts to

$$(\rho, \kappa) \models_{me[w \mapsto \beta, v \mapsto \beta]} Q\{w/v\} \quad \text{and} \quad \forall \chi \in \rho(me(u)) : \kappa(\chi) \subseteq \rho(\beta)$$

so that by the induction hypothesis this amounts to

$$(\rho, \kappa) \models_{me[w \mapsto \beta, v \mapsto \beta]} Q \quad \text{and} \quad \forall \chi \in \rho(me(u)) : \kappa(\chi) \subseteq \rho(\beta),$$

which by Lemma 3.5 amounts to

$$(\rho, \kappa) \models_{me[w \mapsto \beta]} Q \quad \text{and} \quad \forall \chi \in \rho(me(u)) : \kappa(\chi) \subseteq \rho(\beta),$$

which again amounts to

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

as was to be shown. ■

COROLLARY 3.7. *Assume that $z \notin fn(P)$ and $\eta \in \mathcal{B} \cup \mathcal{C}$; then $(\rho, \kappa) \models_{me[y \mapsto \eta]} P$ if and only if $(\rho, \kappa) \models_{me[z \mapsto \eta]} P\{z/y\}$.*

Proof. By Lemma 3.6

$$(\rho, \kappa) \models_{me[y \mapsto \eta, z \mapsto \eta]} P \text{ iff } (\rho, \kappa) \models_{me[y \mapsto \eta, z \mapsto \eta]} P\{z/y\}$$

and by Lemma 3.5 and $z \notin fn(P)$

$$(\rho, \kappa) \models_{me[y \mapsto \eta]} P \text{ iff } (\rho, \kappa) \models_{me[z \mapsto \eta]} P\{z/y\}$$

as was to be shown. ■

LEMMA 3.8. *Assume that $P \equiv Q$; then $(\rho, \kappa) \models_{me} P$ if $(\rho, \kappa) \models_{me} Q$.*

Proof. The proof is by induction on the construction of $P \equiv Q$ and here we only consider the two harder cases.

The Case of α-Equivalence. Consider the subcase $(\nu x^{\chi})P \equiv (\nu y^{\chi})(P\{y/x\})$ where $y \notin fn((\nu x^{\chi})P)$. We calculate that

$$(\rho, \kappa) \models_{me} (\nu x^{\chi})P$$

is equivalent to

$$(\rho, \kappa) \models_{me[x \mapsto \chi]} P,$$

which by Corollary 3.7 is equivalent to

$$(\rho, \kappa) \models_{me[y \mapsto \chi]} P\{y/x\}$$

(since either $y \notin fn(P)$ or $y = x$), which is equivalent to

$$(\rho, \kappa) \models_{me} (\nu y^{\chi})(P\{y/x\})$$

as was to be shown. The other subcase is similar.

The Cases. $(\nu x^{\chi})(P_1 \mid P_2) \equiv (\nu x^{\chi})P_1 \mid P_2$ (if $x \notin fn(P_2)$) and $(\nu x^{\chi})P \equiv$ (if $x \notin fn(P)$) are easy consequences of Lemma 3.5 and the case $(\nu x^{\chi})(\nu y^{\chi'})P \equiv (\nu y^{\chi'})(\nu x^{\chi})P$ (if $x \neq y$) is straightforward. ■

LEMMA 3.9. *Assume that $(\rho, \kappa) \models_{me} P$ and $me(w) \in \rho(me(z))$; then $(\rho, \kappa) \models_{me} P\{w/z\}$.*

Proof. The proof is by structural induction on P. Most cases are straightforward using the fact

$$\forall x : \rho(me(x\{w/z\})) \subseteq \rho(me(x)).$$

Here we only consider the two harder cases.

The Case $P = u(v^\beta).Q$. By Lemma 3.8 (since α-equivalence is part of the structural congruence) we may without loss of generality assume that v is neither w nor z. Then we may calculate that

$$(\rho, \kappa) \models_{me} u(v^\beta).Q$$

amounts to

$$(\rho, \kappa) \models_{me[v \mapsto \beta]} Q \quad \text{and} \quad \forall \chi \in \rho(me(u)) : \kappa(\chi) \subseteq \rho(\beta),$$

which, by the induction hypothesis and the fact stated above, imply that

$$(\rho, \kappa) \models_{me[v \mapsto \beta]} Q\{w/z\} \quad \text{and} \quad \forall \chi \in \rho(me(u\{w/z\})) : \kappa(\chi) \subseteq \rho(\beta),$$

which is equivalent to the required

$$(\rho, \kappa) \models_{me} (u(v^\beta).Q)\{w/z\}.$$

The Case $P = [x = y]Q$. Our assumption $(\rho, \kappa) \models_{me} P$ amounts to

$$(\rho(me(x)) \cap \rho(me(y)) \neq \emptyset \Rightarrow (\rho, \kappa) \models_{me} Q$$

and our goal is to show

$$\rho(me(x\{w/z\})) \cap \rho(me(y\{w/z\})) \neq \emptyset \Rightarrow (\rho, \kappa) \models_{me} Q\{w/z\}$$

as this amounts to $(\rho, \kappa) \models_{me} P\{w/z\}$. By the induction hypothesis it suffices to show that

$$\rho(me(x\{w/z\})) \cap \rho(me(y\{w/z\})) \neq \emptyset \Rightarrow \rho(me(x)) \cap (\rho(me(y)) \neq \emptyset$$

that is immediate using the fact stated at the beginning of the proof. ■

Subject Reduction

To establish the semantic correctness of our analysis we rely on the definition of the early semantics in Table 1 as well as on the analysis in Table 2. The subject reduction result below applies to *all* the solutions of the analysis and hence in particular to the least. The operational semantics only rewrites processes at “top level” where it is natural to demand that all free names are bound to channels (rather than to binders); this is formalised by the condition $me[fn(-)] \subseteq \mathcal{C}$. In the statement below, we write $(\rho, \kappa) \models^{\mathcal{C}}_{me} P$ as a shorthand for $(\rho, \kappa) \models_{me} P \wedge me[fn(P)] \subseteq \mathcal{C}$.

THEOREM 3.10. *If $(\rho, \kappa) \models^{\mathcal{C}}_{me} P$ and $P \xrightarrow{\mu} Q$ we have:*

$$\textit{if } \mu = \tau \textit{ then } (\rho, \kappa) \models^{\mathcal{C}}_{me} Q; \tag{1}$$

$$\textit{if } \mu = \bar{x}y \textit{ then } (\rho, \kappa) \models^{\mathcal{C}}_{me} Q, \textit{and } me(y) \in \kappa(me(x)); \tag{2a}$$

$$\textit{if } \mu = \bar{x}(y^\chi) \textit{ then } (\rho, \kappa) \models^{\mathcal{C}}_{me[y \mapsto \chi]} Q, \textit{ and } \chi \in \kappa(me(x)); \tag{2b}$$

$$\textit{if } \mu = xy \textit{ and } me(y) \in \kappa(me(x)) \textit{ then } (\rho, \kappa) \models^{\mathcal{C}}_{me} Q; \tag{3a}$$

$$\textit{if } \mu = x(y^\chi) \textit{ and } \chi \in \kappa(me(x)) \textit{ then } (\rho, \kappa) \models^{\mathcal{C}}_{me[y \mapsto \chi]} Q. \tag{3b}$$

Proof. The proof is by induction on the construction of $P \xrightarrow{\mu} Q$ and with subcases depending on whether case (1), (2a), (2b), (3a), or (3b) applies. Throughout assume that $me[fn(P)] \subseteq \mathcal{C}$, $(\rho, \kappa) \models_{me} P$ (i.e. $(\rho, \kappa) \models^{\mathcal{C}}_{me} P$) and $P \xrightarrow{\mu} Q$.

The Case (1). That $me[fn(Q)] \subseteq \mathcal{C}$ is immediate from Fact 2.3. It remains to show that $(\rho, \kappa) \models_m Q$. In the case of *Tau* this is immediate; clearly the axioms *Out*, *FreeIn*, and *BoundIn* do not apply. Thanks to Lemma 3.8 and the induction hypothesis (1), the property is preserved by the rules *Var, Par, Sum, Res,* and *Match*; clearly the rule *Open* does not apply.

For suitable Q_1 and Q_2 such that $Q = Q_1 \mid Q_2$, in the case of rule *Close* the induction hypothesis (2b) ensures that

$$(\rho, \kappa) \models_{me[y\mapsto\chi]} Q_1 \wedge \chi \in \kappa(me(x))$$

and the induction hypothesis (3b) ensures that

$$(\rho, \kappa) \models_{me[y\mapsto\chi]} Q_2$$

thereby establishing the desired

$$(\rho, \kappa) \models_{me} (\nu y^\chi)(Q_1 \mid Q_2).$$

The case of rule *Com* is similar but uses the induction hypotheses (2a) and (3a).

The Case (2a). That $me[fn(Q)] \subseteq \mathcal{C}$ is immediate from Fact 2.3; furthermore $\rho(me(x)) = \{me(x)\} \subseteq \mathcal{C}$ and $\rho(me(y)) = \{me(y)\} \subseteq \mathcal{C}$. It remains to show that $(\rho, \kappa) \models_{me} Q$ and that $me(y) \in \kappa(me(x))$. In the case of *Out* it follows from Table 2 that $(\rho, \kappa) \models_{me} Q$ and $\forall\chi \in \rho(me(x)) : \rho(me(y)) \subseteq \kappa(\chi)$, which amounts to $me(y) \in \kappa(me(x))$; the axioms *Tau*, *FreeIn*, and *BoundIn* do not apply. Thanks to Lemma 3.8 and the induction hypothesis (2a) the property is preserved by the rules *Var*, *Par*, *Sum*, *Res*, and *Match*; clearly the rules *Open*, *Close*, and *Com* do not apply.

The Case (2b). That $(me[y \mapsto \chi])[fn(Q)] \subseteq \mathcal{C}$ is immediate from Fact 2.3; furthermore $\rho(me(x)) = \{me(x)\} \subseteq \mathcal{C}$. It remains to show that $(\rho, \kappa) \models_{me[y\mapsto\chi]} Q$ and that $\chi \in \kappa(me(x))$. None of the axioms *Tau, Out, FreeIn,* and *BoundIn* nor any of the rules *Close* or *Com* can apply. Thanks to Lemma 3.8 and the induction hypothesis (2b) the property is preserved by the rules *Var, Par, Sum, Res,* and *Match.* In the case of rule *Open* the induction hypothesis (2a) and $y \neq x$ ensure that $(\rho, \kappa) \models_{me[y\mapsto\chi]} Q$ and $(me[y \mapsto \chi])(y) \in \kappa((me[y \mapsto \chi])(x))$, which establish the desired result.

The Case (3a). Here we also assume that $me(y) \in \kappa(me(x))$ so that $me(y) \in \mathcal{C}$. Then $me[fn(Q)] \subseteq \mathcal{C}$ is immediate from Fact 2.3 that also implies $\rho(me(x)) = \{me(x)\} \subseteq \mathcal{C}$; it remains to show that $(\rho, \kappa) \models_{me} Q$.

In the case of *FreeIn, P* is on the form $x(w^\beta).R$ and Q is $R\{y/w\}$. By Lemma 3.8 (since α-equivalence is part of the structural congruence) we may use rule *Var* to guarantee that $w \neq y$. From $(\rho, \kappa) \models_{me} x(w^\beta).R$ we have that $(\rho, \kappa) \models_{me[\omega\mapsto\beta]} R$ and $\kappa(me(x)) \subseteq \rho(\beta)$. It follows that $me(y) \in \rho(\beta)$, which may be rewritten as $(me[w \mapsto \beta])(y) \in \rho((me[w \mapsto \beta])(w))$. From Lemma 3.9 we then get $(\rho, \kappa) \models_{me[w\mapsto\beta]} R\{y/w\}$ and the desired $(\rho, \kappa) \models_{me} R\{y/w\}$ follows, by Lemma 3.5.

Neither the axioms *Tau, Out,* and *BoundIn,* nor the rules *Open, Close,* or *Com* are applicable. Thanks to Lemma 3.8 and induction hypothesis (3a) the property is preserved by the rules *Var, Par, Sum, Res,* and *Match.*

The Case (3b). Here we also assume that $\chi \in \kappa(me(x)) \subseteq \mathcal{C}$. From Fact 2.3 we have $(me[y \mapsto \chi])[fn(Q)] \subseteq \mathcal{C}$ and $\rho(me(x)) = \{me(x)\} \subseteq \mathcal{C}$ so that it remains to show that $(\rho, \kappa) \models_{me[y\mapsto\chi]} Q$.

The axiom *BoundIn* applies in this case so P has the form $x(y^\beta).Q$. Let now $w \notin \{y\} \cup fn(Q)$ and note that $x(w^\beta).R \equiv x(y^\beta).Q$ for $R = Q\{w/y\}$. Since $Q \equiv R\{y/w\}$ it follows from Lemma 3.8 that it suffices to show that $(\rho, \kappa) \models_{me[y\mapsto\chi]} R\{y/w\}$. From $(\rho, \kappa) \models_{me} P$ and Lemma 3.8 it follows that $(\rho, \kappa) \models_{me} x(w^\beta).R$ and hence that $(\rho, \kappa) \models_{me[w\mapsto\beta]} R$ and $\kappa(me(x)) \subseteq \rho(\beta)$. Using Lemma 3.5 it now follows that $(\rho, \kappa) \models_{me[w\mapsto\beta][y\mapsto\chi]} R$. As in the previous case we have $(me[w \mapsto \beta][y \mapsto \chi])(y) \in \rho((me[w \mapsto \beta][y \mapsto \chi](w))$ and Lemma 3.9 gives $(\rho, \kappa) \models_{me[w\mapsto\beta][y\mapsto\chi]} R\{y/w\}$. The desired $(\rho, \kappa) \models_{me[y\mapsto\chi]} R\{y/\omega\}$ follows by Lemma 3.5.

Neither the axioms *Tau, Out,* and *FreeIn,* nor the rules *Open, Close,* or *Com* are applicable. Thanks to Lemma 3.8 and induction hypothesis (3b) the property is preserved by the rules *Var, Par, Sum, Res,* and *Match.*

The inclusions occurring in the above items mildly constrain the environment where the process under validation operates. If one only considers closed systems, which can only perform τ moves, the inclusions become useless, as all names are bound; in essence we retain only item (1).

4. CONSTRUCTION OF SOLUTIONS

There is also a constructive procedure for obtaining the least solution which operates in $O(N^5)$ time in the size of processes (see [18, 31]). To describe it we shall concentrate on a process, $P_\star$, of interest as well as a marker environment, $me_\star$, giving the binders and channels of free names in the process. To obtain a finite algorithm we prefer to restrict our attention to a finite universe, $\mathcal{U}_\star$, containing all the relevant binders and channels and the set $me[fn(P_\star)] =_{\text{def}} \{me(x) \mid x \in fn(P_\star)\}$. We shall say that the size, N, of $P_\star$ is the maximum of the number of symbols in $P_\star$ and the number of elements in $\mathcal{U}_\star$. In case we take an annotation which has all markers different from each other and disjoint from the image of the marker environment, N linearly depends on the size of $P_\star$, only.

Validating a solution $(\rho, \kappa) \models_{me} P$ amounts to checking a number of individual constraints. We now define a function $\mathcal{G}_C\llbracket P\rrbracket_{me}$ for explicitly extracting the set of constraints to be checked. In doing so we shall make it clear that we are extracting the constraints in the form of syntax, and hence replace ρ of Table 2 by r and κ by k.

Definition 4.1. We now define two classes of constraints.

- An *individual constraint* is on one of the forms $n_1 \subseteq n_2$, $\{\chi\} \subseteq n_0$, or $n_1 \cap n_2 \neq \emptyset$, where each n_i will be on one of the three forms $r(\beta)$, $r(\chi)$ or $k(\chi)$.
- A *composite constraint* is on the form $\{ic_1, \ldots, ic_m\} \Rightarrow$ ic where $m \geq 0$, all ic_i (on the left-hand side) are individual constraints on either form $\{\chi\} \subseteq n_0$ or $n_1 \cap n_2 \neq \emptyset$, and ic (on the right-hand side) is an individual constraint on the form $n_1 \subseteq r(\beta)$ or $n_1 \subseteq k(\chi)$.

The details of $\mathcal{G}_C\llbracket P\rrbracket_{me}$ are given in Table 3. In the clause for $[x = y]P$ we cannot decide whether or not $r(me(x)) \cap r(me(y)) \neq \emptyset$ and hence we ensure that the constraints subsequently generated will in fact explicitly test this. Technically, this is achieved by means of the subscript C to the function $\mathcal{G}_C\llbracket P\rrbracket_{me}$: each constraint generated will be conditional on all of the constraints in the (initially empty) set C. Similarly, in the clauses for $\bar{x}y.P$ and $x(y^\beta).P$ we use the assumptions about the universe $\mathcal{U}_\star$, in particular that $me[fn(P_\star)] \subseteq \mathcal{U}_\star$, to generate a sufficiently large set of constraints that then explicitly test for $\{\chi\} \subseteq r(me(x))$.

The call $\mathcal{G}_\emptyset\llbracket P_\star\rrbracket_{me_\star}$ will give rise to at most $O(N)$ recursive calls, each call directly responsible for generating at most $O(N)$ constraints (see the clauses for input and output). The set of constraints C occurring as index to the recursive calls $\mathcal{G}_C\llbracket P\rrbracket_{me}$ can have size at most $O(N)$ (see the clause for matching). Hence at most $O(N^2)$ constraints of size $O(N)$ will be produced.

TABLE 3

Constraint Generation for the π-Calculus

$\mathcal{G}_C\llbracket 0\rrbracket_{me}$	$= \emptyset$
$\mathcal{G}_C\llbracket r\cdot P\rrbracket_{me}$	$= \mathcal{G}_C\llbracket P\rrbracket_{me}$
$\mathcal{G}_C\llbracket \bar{x}y\cdot P\rrbracket_{me}$	$= \mathcal{G}_C\llbracket P\rrbracket_{me} \cup \{(C \cup \{\{\chi\} \subseteq r(me(x))\}) \Rightarrow r(me(y)) \subseteq k(\chi) \mid \chi \in \mathcal{U}_\star \cap \mathcal{C}\}$
$\mathcal{G}_C\llbracket x(y^\beta)\cdot P\rrbracket_{me}$	$= \mathcal{G}_C\llbracket P\rrbracket_{me[y\mapsto\beta]} \cup \{(C \cup \{\{\chi\} \subseteq r(me(x))\}) \Rightarrow k(\chi) \subseteq r(\beta) \mid \chi \in \mathcal{U}_\star \cap \mathcal{C}\}$
$\mathcal{G}_C\llbracket P_1 + P_2\rrbracket_{me}$	$= \mathcal{G}_C\llbracket P_1\rrbracket_{me} \cup \mathcal{G}_C\llbracket P_2\rrbracket_{me}$
$\mathcal{G}_C\llbracket P_1 \mid P_2\rrbracket_{me}$	$= \mathcal{G}_C\llbracket P_1\rrbracket_{me} \cup \mathcal{G}_C\llbracket P_2\rrbracket_{me}$
$\mathcal{G}_C\llbracket (\nu x^\chi)P\rrbracket_{me}$	$= \mathcal{G}_C\llbracket P\rrbracket_{me[x\mapsto\chi]}$
$\mathcal{G}_C\llbracket [x = y]P\rrbracket_{me}$	$= \mathcal{G}_{C\cup\{r(me(x))\cap r(me(y))\neq\emptyset\}}\llbracket P\rrbracket_{me}$
$\mathcal{G}_C\llbracket !P\rrbracket_{me}$	$= \mathcal{G}_C\llbracket P\rrbracket_{me}$

The Semantics of the Constraints

To relate Tables 2 and 3 we need to interpret a set of constraints with respect to a proposed solution (ρ, κ). First we define an evaluation function for left-hand sides:

DEFINITION 4.2. We define $\mathcal{V}_{(\rho,\kappa)}[\![n]\!]$ as follows:

$$\mathcal{V}_{(\rho,\kappa)}[\![n]\!] = \begin{cases} \rho(\beta) & \text{if } n = r(\beta) \\ \{\chi\} & \text{if } n = r(\chi) \\ \kappa(\chi) & \text{if } n = k(\chi). \end{cases}$$

Next we define a satisfaction relation.

DEFINITION 4.3. We define

$$\begin{array}{ll} (\rho,\kappa) \text{ sat } n_1 \subseteq n_2 & \text{iff } \mathcal{V}_{(\rho,\kappa)}[\![n_1]\!] \subseteq \mathcal{V}_{(\rho,\kappa)}[\![n_2]\!] \\ (\rho,\kappa) \text{ sat } \{\chi\} \subseteq n & \text{iff } \chi\{h\} \subseteq \mathcal{V}_{(\rho,\kappa)}[\![n]\!] \\ (\rho,\kappa) \text{ sat } n_1 \cap n_2 \neq \emptyset & \text{iff } \mathcal{V}_{(\rho,\kappa)}[\![n_1]\!] \cap \mathcal{V}_{(\rho,\kappa)}[\![n_2]\!] \neq \emptyset \end{array}$$

and also

$$(\rho,\kappa) \text{ sat } \{ic_1, \ldots, ic_m\} \Rightarrow ic \quad \text{iff } \left(\bigwedge_{i=1}^{m} (\rho,\kappa) \text{sat } ic_i\right) \Rightarrow ((\rho,\kappa) \text{ sat } ic)$$

and finally

$$(\rho,\kappa) \text{ SAT } C \quad \text{iff } \forall cc \in C : (\rho,\kappa) \text{ sat } cc$$

The relationship between Tables 2 and 3 is now given by the following result.

LEMMA 4.4. $((\rho,\kappa)\,\text{SAT}\ \mathcal{G}_C[\![P]\!]_{me})$ *iff* $(((\rho,\kappa)\,\text{SAT}\ C) \Rightarrow ((\rho,\kappa) \models_{me} P))$.

Proof. We proceed by structural induction on P. Most cases are straightforward and here we only consider two of the more interesting cases.

The Case $x(y^\beta)\cdot P$. We have that (ρ,κ) SAT $\mathcal{G}_C[\![x(y^\beta).P]\!]_{me}$ is equivalent to

$$(\rho,\kappa) \text{ SAT } \mathcal{G}_C[P]_{me[y\mapsto\beta]}$$
$$(\rho,\kappa) \text{ SAT } \{C \cup \{\{\chi\} \subseteq r(me(x))\}) \Rightarrow k(\chi) \subseteq r(\beta) \mid \chi \in \mathcal{U}_\star \cap \mathcal{C}\}.$$

Using the induction hypothesis this is equivalent to

$$((\rho,\kappa) \text{ SAT } C) \Rightarrow (\rho,\kappa) \models_{me[y\mapsto\beta]} P$$
$$((\rho,\kappa) \text{ SAT } C) \Rightarrow (\forall \chi \in \mathcal{U}_\star \cap \mathcal{C} \ :\ \chi \in \rho(me(x)) \Rightarrow \kappa(\chi) \subseteq \rho(\beta))$$

and given the assumptions about the universe $\mathcal{U}_\star$ this is equivalent to the desired $((\rho,\kappa)$ SAT $C) \Rightarrow (\rho,\kappa) \models_{me} x(y^\beta).P$.

The Case $[x = y]P$. We have that (ρ,κ) SAT $\mathcal{G}_C[\![[x = y]P]\!]_{me}$ is equivalent to

$$(\rho,\kappa) \text{ SAT } \mathcal{G}_{C\cup\{r(me(x))\cup r(me(y))\neq\emptyset\}}[\![P]\!]_{me}.$$

Using the induction hypothesis and the definition of SAT this is equivalent to

$$((\rho,\kappa) \text{ SAT } C \wedge (\rho,\kappa) \text{ sat } (r(me(x)) \cap r(me(y)) \neq \emptyset)) \Rightarrow ((\rho,\kappa) \models_{me} P),$$

which using the definition of sat may be rewritten as

$$(\rho,\kappa)\ \mathsf{SAT}\ C \wedge (\rho(me(x)) \cap \rho(me(y)) \neq \emptyset)) \Rightarrow ((\rho,\kappa) \models_{me} P)$$

that is equivalent to the desired $((\rho,\kappa)\ \mathsf{SAT}\ C) \Rightarrow (\rho,\kappa)\ \models_{me} [x = y]P$). ■

Solving the Constraints

We now demonstrate how to solve a set of constraints as might have been generated above. For our purposes, merely obtaining an algorithm running in $O(N^5)$ time, it suffices to perform a simple iterative procedure upon a function f_C associated with the set of constraints C.

DEFINITION 4.5. Given a set C of constraints, the function f_C maps a proposed solution into another one as

$$f_C(\rho,\kappa) = (\rho',\kappa'),$$

where

$$\rho'(\beta) = \bigcup_{(\{ic_1,\ldots,ic_m\}\Rightarrow n \subseteq r(\beta)) \in C} \begin{cases} \mathcal{V}_{(\rho,\kappa)}[\![n]\!] & \text{if } \bigwedge_{i=1}^m (\rho,\kappa)\ \mathsf{sat}\ ic_i \\ \emptyset & \text{otherwise} \end{cases}$$

and

$$\kappa'(\chi) = \bigcup_{(\{ic_1,\ldots,ic_m\}\Rightarrow n \subseteq k(\chi)) \in C} \begin{cases} \mathcal{V}_{(\rho,\kappa)}[\![n]\!] & \text{if } \bigwedge_{i=1}^m (\rho,\kappa)\ \mathsf{sat}\ ic_i \\ \emptyset & \text{otherwise.} \end{cases}$$

A related function $\hat{f}_C$ is given by

$$\hat{f}_C(\rho,\kappa) = (\rho,\kappa) \sqcup f_C(\rho,\kappa).$$

The relationship between the two functions, f_C and $\hat{f}_C$, and satisfaction, $\mathsf{SAT}\ C$, of the set of constraints is given by the following result.

LEMMA 4.6. *$(\rho,\kappa)\ \mathsf{SAT}\ C$ iff $f_C(\rho,\kappa) \sqsubseteq (\rho,\kappa)$ iff $\hat{f}_C(\rho,\kappa) = (\rho,\kappa)$ whenever C is a set of composite constraints.*

Proof. The second "iff" is immediate so consider the first. A composite constraint in C has the form $\{ic_1,\ldots,ic_m\} \Rightarrow n_1 \subseteq n_2$; there are two possibilities for n_2 and here we just consider the case where n_2 is $r(\beta)$. Then

$$(\rho,\kappa)\ \mathsf{sat}\ (\{ic_1,\ldots,ic_m\} \Rightarrow n_1 \subseteq r(\beta))$$

is equivalent to

$$\left(\bigwedge_{i=1}^m (\rho,\kappa)\ \mathsf{sat}\ ic_i\right) \Rightarrow \mathcal{V}_{(\rho,\kappa)}[\![n_1]\!] \subseteq \rho(\beta),$$

which can be rewritten as

$$\left(\begin{cases} \mathcal{V}_{(\rho,\kappa)}[\![n_1]\!] & \text{if } \bigwedge_{i=1}^m (\rho,\kappa)\ \mathsf{sat}\ ic_i \\ \emptyset & \text{otherwise} \end{cases}\right) \subseteq \rho(\beta).$$

It follows that

$$(\rho,\kappa)\ \mathsf{SAT}\ \{(\{ic_1,\ldots,ic_m\} \Rightarrow n_1 \subseteq r(\beta')) \in C \mid \beta' = \beta\}$$

is equivalent to

$$\bigcup_{(\{ic_1,\ldots,ic_m\}\Rightarrow n_1 \subseteq r(\beta'))\in C \wedge \beta'=\beta} \left(\begin{cases} \mathcal{V}_{(\rho,\kappa)}[\![n_1]\!] & \text{if } \bigwedge_{i=1}^{m}(\rho,\kappa) \;\mathsf{sat}\; ic_i \\ \emptyset & \text{otherwise} \end{cases} \right) \subseteq \rho(\beta)$$

and the desired result easily follows. ■

To prepare for the iteration we need the following standard result.

LEMMA 4.7. *f_C is monotone and $\hat{f}_C$ is monotone and extensive whenever C is a set of composite constraints.*

Proof. It is immediate that $\hat{f}_C$ is extensive (meaning that $\forall(\rho,\kappa):(\rho,\kappa) \sqsubseteq \hat{f}_C(\rho,\kappa)$) and that it is monotone (meaning that $\forall(\rho,\kappa) \sqsubseteq (\rho',\kappa'): \hat{f}_C(\rho,\kappa) \sqsubseteq \hat{f}_C(\rho',\kappa')$) if f_C is. To see that f_C is monotone let $(\rho,\kappa) \sqsubseteq (\rho',\kappa')$ and consider a composite constraint $\{ic_1,\ldots,ic_m\} \Rightarrow n_1 \subseteq n_2$ in C; there are two possibilities for n_2 and here we just consider the case where n_2 is $r(\beta)$. It is immediate that $\mathcal{V}_{(\rho,\kappa)}[\![n_1]\!] \subseteq \mathcal{V}_{(\rho',\kappa')}[\![n_1]\!]$ and we also have (ρ,κ) sat $ic_i \Rightarrow (\rho',\kappa')$ sat ic_i because ic_i is either of the form $n'_1 \cap n'_2 \neq \emptyset$ or $\{\chi\} \subseteq n'$. This establishes the result. ■

Setting $C = \mathcal{G}_\emptyset[P_\star]_{me_\star}$ it now follows that the least (ρ,κ) (above a given $(\overline{\rho},\overline{\kappa})$, typically $(\bot,\bot)$) that satisfies $(\rho,\kappa) \models_{me_\star} P_\star$ equals the least fixed point of f_C (above $(\overline{\rho},\overline{\kappa})$). Because of our assumptions about the universe, $\mathcal{U}_\star$, the set C is finite and $\hat{f}_C$ operates over a complete lattice of finite size. Standard fixed point theory then ensures that the desired (ρ,κ) is obtained as $\hat{f}_C^{\,j}(\overline{\rho},\overline{\kappa})$ upon stabilisation, i.e., for the least j such that $\hat{f}_C^{\,j}(\overline{\rho},\overline{\kappa}) = \hat{f}_C^{\,j+1}(\overline{\rho},\overline{\kappa})$, where it is certain that such j exists. We may summarise the development as follows.

THEOREM 4.8. *Writing $C = \mathcal{G}_\emptyset[\![P_\star]\!]_{me_\star}$ there is a least natural number j such that $\hat{f}_C^{\,j}(\overline{\rho},\overline{\kappa}) = \hat{f}_C^{\,j+1}(\overline{\rho},\overline{\kappa})$; it satisfies that*

$$\hat{f}_C^{\,j}(\overline{\rho},\overline{\kappa}) \models_{me_\star} P_\star$$

and that $\hat{f}_C^{\,j}(\overline{\rho},\overline{\kappa}) = \sqcap\{(\rho,\kappa) \mid (\rho,\kappa) \models_{me_\star} P_\star \wedge (\rho,\kappa) \sqsupseteq (\overline{\rho},\overline{\kappa})\}$.

The tuple (ρ,κ) contains *O(N)* elements taking values in a set of size *O(N)*. Hence at most $O(N^2)$ iterations of the function will be needed. Each iteration can be performed by considering each of the $O(N^2)$ constaints of form $\{ic_1,\ldots,ic_m\} \Rightarrow n_1 \subseteq n_2$, calculating $ic_1,\ldots,ic_m$ and n_1 in time *O(N)* and then updating the entry for n_2 accordingly. Hence the desired solution can be obtained in time $O(N^5)$ and space $O(N^3)$.

EXAMPLE 4.9. As an example suppose that (ρ_1,κ_1) is the least solution for P_1 and that (ρ_2,κ_2) is the least solution for P_2 (w.r.t. the same *me*). It follows that there also is a least solution (ρ,κ) for $P_1 \mid P_2$ and since this is a solution for P_1 and for P_2 we have

$$(\rho_1,\kappa_1) \sqcup (\rho_2,\kappa_2) \sqsubseteq (\rho,\kappa).$$

We therefore know that (ρ,κ) equals

$$\hat{f}_C^{\,j}((\rho_1,\kappa_1) \sqcup (\rho_2,\kappa_2))$$

upon stabilisation.

To see that j cannot always be taken to be 0 consider the two processes $P_1 = x(y^{\beta_0}).(\nu x^{\chi_1})\overline{x}x$ and $P_2 = (\nu x^{\chi_1})(\nu z^{\chi_2})\overline{x}z$, the marker environment $me(x) = \chi_1$, and the following definitions of ρ_i and κ_i:

	$\kappa_i(\chi_1)$	$\rho_i(\beta_0)$	$\cdots$
$i = 1$	$\{\chi_1\}$	$\{\chi_1\}$	$\cdots$
$i = 2$	$\{\chi_2\}$	$\emptyset$	$\cdots$

Setting $(\rho', \kappa') = (\rho_1, \kappa_1) \sqcup (\rho_2, \kappa_2)$ we have $\kappa'(\chi_1) = \{\chi_1, \chi_2\}$ and $\rho'(\beta_0) = \{\chi_1\}$ and hence the constraint $\kappa'(\chi_1) \subseteq \rho'(\beta_0)$ fails.

5. PREVENTING LEAKAGE

A common approach to system security is based on putting objects and subjects into security classes. We demonstrate that control flow analysis helps in statically detecting useful information on security. We consider in this section a static property ensuring that a channel, devised to be secret to a process *P*, is never communicated to an external user. In the next section, we statically check another property that concerns how information flows between processes at different levels.

The π-calculus does not distinguish between data, that may be secret, and channels that can be seen as capabilities of accessing data. More refined calculi, notably the *spi*-calculus [3], make such a distinction and permit a finer description of security properties. We choose here a pure calculus of computations, without encryption and decryption primitives, in order to concentrate on the applicability of the control flow analysis to security issues. In fact, the scoping rules of the π-calculus are sufficient for a careful use of channels, because processes can generate and pass new names, making the channels they denote available for communication. In a sense, learning the name of a channel amounts to possessing the capability to communicate on it, and restriction governs the visibility of names.

A process *P* could have the security requirement of keeping secret (some of) its channels, i.e., not to communicate them to the external environment. A process matches this requirement if it never performs an extrusion of a secret channel, as made precise below. In the following, we assume that $\mathcal{S}$ is a set of *secret* channels given by a designated authority, e.g., the designer or the user of a process *P*, who implicitly introduces also the set $\mathcal{P}$ of *public* channels as the complement of $\mathcal{S}$. A priori, binders are neither public nor secret; the actual solution of the analysis establishes which kind of channels can be bound to each binder.

Definition 5.1. The pair P, *me* is *admissible* for the set $\mathcal{S} \subseteq \mathcal{C}$ of secret channels, if and only if $me[fn(P)] \subseteq \mathcal{C}$ and $me[fn(P)] \cap \mathcal{S} = \emptyset$. Then, the set of public channels is $\mathcal{P} = \mathcal{C} \backslash \mathcal{S}$.

Note that the condition of admissibility is equivalent to $me[fn(P)] \subseteq \mathcal{P}$; i.e., all free names are public channels.

A Dynamic Notion

Now, we 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 neither it nor any of its derivatives can perform an extrusion of a name bound to a secret channel. For this to make sense it is important to assume that the environment (or an external user) cannot guess any secret channels. We formalise this by a constrained notion of computation called *censored*. Essentially, a computation is censored if no name *y*, with $me(y) \in \mathcal{S}$, can be read from the environment through an input.

Definition 5.2. Given *P*, *me*, $\mathcal{S}$ a *censored* step $(P, me) \overset{\mu}{\mapsto} (Q, me')$ is defined whenever the following conditions hold:

1. $P \xrightarrow{\mu} Q$
2. (a) if $\mu = xy$, then me$(y) \in \mathcal{P}$

   (b) if $\mu = x(y^\chi)$, then $\chi \in \mathcal{P}$

$$\text{where } me' = \begin{cases} me & \text{if } \mu = \tau, \bar{x}y, xy \\ me[y \mapsto \chi] & \text{if } \mu = \bar{x}(y^\chi), x(y^\chi). \end{cases}$$

A *censored* computation $(P, me) \mapsto^* (Q, me') \overset{\mu}{\mapsto} (R, me'')$ is made of censored steps, whose labels are all immaterial, apart from the last one.

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 role played by the second condition of a censored step.

LEMMA 5.3. *Let P, me be admissible for $\mathcal{S}$ and $(P,\ me) \overset{\mu}{\mapsto} (Q,\ me')$ be such that $\mu = \bar{x}(y^{\chi})$ implies $\chi \in \mathcal{P}$; then $Q,\ me'$ are admissible for $\mathcal{S}$.*

Proof. The proof is by cases on μ. If $\mu = \tau$ or $\mu = \bar{x}y$ then $me' = me$; from the hypotheses and Fact 2.3 we have that $me'\ [fn(Q)] \subseteq \mathcal{P}$. If $\mu = \bar{x}(y^{\chi})$ we have that $me' = me[y \mapsto \chi]$; from the hypotheses and Fact 2.3 we get that $me'[fn(Q)] \subseteq \mathcal{P}$. If $\mu = xy$, $me(y) \in \mathcal{P}$ (by censored step) and $me' = me$, therefore the hypotheses and Fact 2.3 ensure that $me'[fn(Q)] \subseteq \mathcal{P}$. If $\mu = x(y^{\chi})$ we have that $\chi \in \mathcal{P}$ (by censored step) and that $me' = me[y \mapsto \chi]$ so that the hypotheses together with Fact 2.3 ensure that $me'[fn(Q)] \subseteq \mathcal{P}$. ■

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

DEFINITION 5.4. The process P has *no leaks* with respect to $\mathcal{S}$, me if and only if $(P, me) \mapsto^* (Q, me')$ implies that there is no pair (R, me'') such that $(Q, me') \overset{\bar{x}(y^{\chi})}{\longmapsto} (R, me'')$ with $\chi \in \mathcal{S}$.

Of course when P is stuck, it has no leaks. Note that if P has no leaks, with respect to $\mathcal{S}$, me, and P, me are admissible for $\mathcal{S}$, then for all Q such that $(P, me) \mapsto^* (Q, me')$, Q, me' are admissible for $\mathcal{S}$, due to Lemma 5.3.

A Static Notion

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*[1] and we prove that it is a sufficient condition for a process to have no leaks.

DEFINITION 5.5. Let P, me be admissible for a given $\mathcal{S}$. A process P is *confined* with respect to $\mathcal{S}$, me if and only if there exists (ρ, κ) such that

$$\text{(a) } (\rho, \kappa) \models_{me} P \qquad \text{and} \qquad \text{(b) } \kappa(\chi) = \mathcal{P} \quad \text{if } \chi \in \mathcal{P}.$$

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

Note that if a process P is confined, by admissibility we also have that $me[fn(P)] \subseteq \mathcal{P}$. 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 secret channels, as no restriction is put on them. We now show that confinement is preserved by censored computations.

LEMMA 5.6 (Subject reduction for confinement). *Let P be confined* (*w.r.t.* $\mathcal{S}$, me) *and* $(P, me) \overset{\mu}{\mapsto} (Q, me')$; *then Q is confined* (*w.r.t.* $\mathcal{S}$, me').

Proof. The proof is by cases on μ and considers the confining solution (ρ, κ) for P. If $\mu = \tau$ or $\mu = \bar{x}y$ we have that $me' = me$; it follows from Theorem 3.10 that $(\rho, \kappa) \models_{me'} Q$ and from Lemma 5.3 that Q, me' is admissible for $\mathcal{S}$. If $\mu = \bar{x}(y^{\chi})$ we have $me' = me[y \mapsto \chi]$ and $x \in fn(P)$ (by Fact 2.3); it follows from Theorem 3.10 and confinement that $(\rho, \kappa) \models_{me'} Q$ and that $\chi \in \kappa(me(x)) = \mathcal{P}$; then, by Lemma 5.3 it follows that Q, me is admissible for $\mathcal{S}$. If $\mu = xy$ or $\mu = x(y^{\chi})$ we have $x \in fn(P)$ (because of Fact 2.3). In the first subcase $me' = me$ and $me(y) \in \mathcal{P} = \kappa(me(x))$ (by censored step and confinement); Theorem 3.10 guarantees that $(\rho, \kappa) \models_{me'} Q$ and Lemma 5.3 that Q, me' is admissible for $\mathcal{S}$. In the second subcase $me' = me[y \mapsto \chi]$, $\chi \in \mathcal{P} = \kappa(me(x))$ (by censored step and confinement); Theorem 3.10 guarantees that $(\rho, \kappa) \models_{me'} Q$ and Lemma 5.3 that Q, me' is admissible for $\mathcal{S}$. ■

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

THEOREM 5.7. *If P is confined* (*with respect to* $\mathcal{S}$, me) *then P has no leaks* (*with respect to* $\mathcal{S}$, me).

[1] In the literature on security, confinement is also used with different meanings.

Proof. By Lemma 5.6 it is enough to prove that $(P, me) \overset{\overline{x}(y^{\chi})}{\longmapsto} (Q, me')$ implies $\chi \in \mathcal{P}$. Since $x \in fn(P)$ by Fact 2.3, we have that $me(x) \in \mathcal{P}$ and $\kappa(me(x)) = \mathcal{P}$ (by confinement). By Theorem 3.10, $\chi \in \kappa(me(x))$ and hence $\chi \in \mathcal{P}$ as was to be shown. ■

EXAMPLE 5.8. We consider here an abstract version of the Wide Mouthed Frog protocol, adapted from [3]. Intuitively, two principals A and B wish to communicate using classical shared-key cryptography, and the first forwards a secret key, say c_{AB}, to the second one, with the help of a server. The server shares the secret keys c_{AS} and c_{SB} with A and B, respectively. So, the principal A passes its key c_{AB}, encrypted with c_{AS}, to the server. The server decrypts the message, encrypts it with c_{SB} and passes it to B. Afterwards A sends the message M, encrypted with c_{AB}, to B.

Here principals will be represented by processes and keys by secret channels, because the π-calculus has no cryptographic primitives. The specification follows:

- $A = (\nu M^{\chi_M})(\nu c_{AB}^{\chi_{AB}})\overline{c_{AS}}c_{AB} \,.\, \overline{c_{AB}}M$
- $S = c_{AS}(x^{\beta_x}) \,.\, \overline{c_{SB}}x$
- $B = c_{SB}(y^{\beta_y}) \,.\, y(z^{\beta_z})$
- $P = (\nu c_{AS}^{\chi_{AS}})(\nu c_{SB}^{\chi_{SB}})(A|S|B)$

The following solution is the least one:

$$\rho(\beta) = \begin{cases} \{\chi_{AB}\} & \text{if } \beta = \beta_x, \beta_y \\ \{\chi_M\} & \text{if } \beta = \beta_z \end{cases} \qquad \kappa(\chi) = \begin{cases} \{\chi_M\} & \text{if } \chi = \chi_{AB} \\ \{\chi_{AB}\} & \text{if } \chi = \chi_{AS}, \chi_{SB} \\ \emptyset & \text{if } \chi = \chi_M \end{cases}$$

If we take $\mathcal{S} = \{\chi_{AB}, \chi_{AS}, \chi_{SB}, \chi_M\}$, for all choices of $\mathcal{P}$ there is a solution confining P (with respect to $\mathcal{S}$). Thus, secrecy of M is guaranteed.

Suppose to have a new name c_{AC}, with $me(c_{AC}) = \chi_c \in \mathcal{P}$. Let A be extended as follows

$$A = (\nu M^{\chi_M})\big(\nu c_{AB}^{\chi_{AB}}\big)\overline{c_{AS}}c_{AB}.\overline{c_{AS}}M.\overline{c_{AC}}c_{AB}.$$

Intuitively, A sends c_{AB} along c_{AC} after the completion of the protocol. For all solutions, it turns out that $\kappa(\chi_{c_{AC}}) \supseteq \{\chi_{AB}\} \not\subseteq \mathcal{P}$. Therefore, there is no solution confining P, with respect to $\mathcal{S}$. Indeed, the analysis reveals the extrusion of c_{AB}. With our specification in the π-calculus, this action does not affect the secrecy of M, yet it signals a potential problem, as an extrusion represents the publication of the corresponding secret key. Indeed, if we take the *spi*-calculus, where encrypted messages are transmitted as cleartext, an enemy may intercept M encrypted, store it, and then decrypt it after c_{AB} has been extruded (this violates the so-called forward-secrecy property, see e.g., [2]).

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 $\mathcal{S} = \{\chi\}$ and $me(y) \neq \chi$. Indeed not all deadlocks can be detected statically. So the extrusion of the name x along channel y is considered a possible violation of secrecy.

Checking the Static Condition

One approach to checking confinement of a process P with respect to $\mathcal{S}$, me is to use the polynomial time algorithm devised in the previous section. First, we check that P, me (or $P_\star$, $me_\star$) is indeed admissible for the $\mathcal{S}$ given. Next, we construct sets of constraints C_a and C_b corresponding closely to the two conditions in Definition 5.5. For C_a we may simply use $\mathcal{G}_\emptyset[\![P]\!]_{me}$. For C_b we generate constraints for half of the equality, namely for the inclusion "$\supseteq$." To be specific, write $\mathcal{P}_\star = \mathcal{P} \cap \mathcal{U}_\star$ and let C_b consist of the constraints $\mathcal{P}_\star \subseteq k(\chi)$ for all $\chi \in \mathcal{P}_\star$. (These constraints can be expanded into a form allowed by Definition 4.1 unlike what would be the case if we let C_b take care of the entire equality.) We then solve the set of constraints in polynomial time so as to get the least solution. Then P is confined if and only if the least solution has a κ component equal to the one displayed in Definition 5.5. This approach to determining whether or not P is confined (w.r.t. to $\mathcal{S}$, me) clearly operates in polynomial time.

6. MULTILEVEL SECURITY

Another way of enforcing security is by defining a hierarchy of levels for processes. The security requirement is that a process classified at 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 of a high level. Sometimes this noninterference property is referred to as *no read-up/no write-down* [16]. These requirements amount to the simple security property that is part of the multilevel access control policy of [5, 14].

To study the no read-up/no write-down property we need an extension to the syntax of the π-calculus to assign security levels to processes. Accordingly, the operational semantics requires a little extension. Moreover, we will adapt the static analysis as well to take care of these levels.

In the following we will only introduce the necessary extensions to the setting of the previous sections.

Syntax

In order to simplify our presentation, we make here two assumptions that are common in the literature (see e.g., [43]). First, we consider only two levels of security clearance: L for low and H for high. So, we introduce the set $\mathcal{L} = \{L, H\}$. The case with a hierarchy of levels is studied in [8]. Second, we assume to have systems made of processes in parallel, and only these top-level components are labelled by clearance levels.

The new syntax, defined below, imports the syntax in Definition 2.2 for processes.

Definition 6.1. Systems, denoted by $S, S', S_1, S_2, \ldots \in$ Sys are built from processes according to the syntax

$$S ::= \langle P \rangle^l \mid (\nu x^\chi)S \mid S|S \mid !S,$$

where $\langle P \rangle^l$ expresses that P has level $l \in \mathcal{L}$.

The definition of free names is trivially extended by letting

$$\begin{array}{ll} fn(\langle P \rangle^l) = fn(P) & fn((\nu x^\chi)S) = fn(S)\backslash\{x\} \\ fn(S_1 \mid S_2) = fn(S_1) \cup fn(S_2) & fn(!S) = fn(S). \end{array}$$

Semantics

The structural congruence now takes care of systems, i.e.:

- $S \equiv S'$ if S and S' are α-equivalent;
- $\langle \mathbf{0} \rangle^H \equiv \langle \mathbf{0} \rangle^L$, and $(Sys/_\equiv, |, \langle \mathbf{0} \rangle^H)$ is a commutative monoid;
- $(\nu x^\chi)(\nu y^{\chi'})S \equiv (\nu y^{\chi'})(\nu x^\chi)S$, if $x \neq y$, $(\nu x^\chi)(S_1 \mid S_2) \equiv (\nu x^\chi)S_1 \mid S_2$ if $x \notin fn(S_2)$, and $(\nu x^\chi)S \equiv S$ if $x \notin fn(S)$;
- $!S \equiv S \mid !S$.

The operational semantics is defined in Table 4, and uses the transition relation defined in Table 1. The transitions have the form $S \underset{l}{\xrightarrow{\mu}} S'$, with $l \in \mathcal{L} \cup \{\epsilon\}$. When different from ϵ, the additional level label l records the clearance of the system performing the transition. When a communication is derived the levels of the sender S_o and of the receiver S_i are discarded, leading to a transition of the form $S_o \mid S_i \underset{\epsilon}{\xrightarrow{\tau}} S'$ (see the rules *ComS* and *CloseS* in Table 4).

Analysis

The solution of our new analysis is a triple (ρ, κ, σ) and the flow logic judgements are on either form

$$(\rho, \kappa, \sigma) \models_{me} S \quad \text{or} \quad (\rho, \kappa, \sigma) \models_{me}^{l} P.$$

TABLE 4

Early Transition System for the π-Calculus with Security Levels

$$Lev: \frac{P \xrightarrow{\mu} Q}{\langle P\rangle^l \xrightarrow[l]{\mu} \langle Q\rangle^l} \qquad VarS: \frac{S_1' \equiv S_1,\ S_1 \xrightarrow[l]{\mu} S_2,\ S_2 \equiv S_2'}{S_1' \xrightarrow[l]{\mu} S_2'}$$

$$ParS: \frac{S_1 \xrightarrow[l]{\mu} S_1'}{S_1|S_2 \xrightarrow[l]{\mu} S_1'|S_2},\ bn(\mu) \cap fn(S_2) = \emptyset \qquad ResS: \frac{S \xrightarrow[l]{\mu} S'}{(\nu x^\chi)S \xrightarrow[l]{\mu} (\nu x^\chi)S'},\ x \notin n(\mu)$$

$$CloseS: \frac{S_1 \xrightarrow[l_1]{\bar{x}(y^\chi)} S_1', S_2 \xrightarrow[l_2]{x(y^\chi)} S_2'}{S_1|S_2 \xrightarrow[\epsilon]{\tau} (\nu y^\chi)(S_1'|S_2')} \qquad OpenS: \frac{S \xrightarrow[l]{\bar{x}y} S'}{(\nu y^\chi)S \xrightarrow[l]{\bar{x}(y^\chi)} S'},\ y \neq x$$

$$ComS: \frac{S_1 \xrightarrow[l_1]{\bar{x}y} S_1',\ S_2 \xrightarrow[l_2]{xy} S_2}{S_1|S_2 \xrightarrow[\epsilon]{\tau} S_1'|S_2'}$$

The role of ρ, κ, me is exactly as in the previous sections. The purpose of the new entries l and $\sigma = \langle\sigma_{in}, \sigma_{out}\rangle$, the *abstract communication structure*, is the following:

- $l \in \mathcal{L}$ keeps track of the current security level of the process under validation and is not needed for systems.
- $\sigma_{in}, \sigma_{out} : \mathcal{L} \to (\mathcal{C} \to \wp(\mathcal{C}))$ give the set of the channels that can be bound to the possible objects of an input and an output action respectively, performed by the subprocesses labelled by $l \in \mathcal{L}$, on a given channel $\chi \in \mathcal{C}$.

The analysis in Table 5 extends the analysis in Table 2, to deal with the levels of security clearance by imposing two further conditions while checking an input and an output prefix. The channels that can be bound to the object of an input (respectively, an output) action along channel χ must be included in $\sigma_{in}(l)(\chi)$ (respectively, $\sigma_{out}(l)(\chi)$), where l is the current security level. This is determined by, and is the only task of, the clause for the analysis of the system $\langle P\rangle^l$. The rules for the other systems are just as the rules for the corresponding processes. The technical results of Section 3 concerning the set of solutions forming a Moore family, as well as the computation of solutions in $O(N^5)$ time and $O(N^3)$ space via the generation of constraints, are easily adapted to the setting at hand. We therefore dispense with the details.

TABLE 5

Control Flow Analysis for the π-Calculus with Security Levels

$(\rho,\kappa,\sigma) \models^l_{me} 0$	iff *true*
$(\rho,\kappa,\sigma) \models^l_{me} \tau\,.\,P$	iff $(\rho,\kappa,\sigma) \models^l_{me} P$
$(\rho,\kappa,\sigma) \models^l_{me} \bar{x}y.P$	iff $(\rho,\kappa,\sigma) \models^l_{me} P \wedge \forall\chi \in \rho(me(x)): \begin{pmatrix} \rho(me(y)) \subseteq \kappa(\chi) \wedge \\ \rho(me(y)) \subseteq \sigma_{out}(l)(\chi) \end{pmatrix}$
$(\rho,\kappa,\sigma) \models^l_{me} x(y^\beta).P$	iff $(\rho,\kappa,\sigma) \models^l_{me[y\mapsto\beta]} P \wedge \forall\chi \in \rho(me(x)): \begin{pmatrix} \kappa(\chi) \subseteq \rho(\beta) \wedge \\ \kappa(\chi) \subseteq \sigma_{in}(l)(\chi) \end{pmatrix}$
$(\rho,\kappa,\sigma) \models^l_{me} P_1 + P_2$	iff $(\rho,\kappa,\sigma) \models^l_{me} P_1 \wedge (\rho,\kappa,\sigma) \models^l_{me} P_2$
$(\rho,\kappa,\sigma) \models^l_{me} P_1 \mid P_2$	iff $(\rho,\kappa,\sigma) \models^l_{me} P_1 \wedge (\rho,\kappa,\sigma) \models^l_{me} P_2$
$(\rho,\kappa,\sigma) \models^l_{me} (\nu x^\chi)P$	iff $(\rho,\kappa,\sigma) \models^l_{me[x\mapsto\chi]} P$
$(\rho,\kappa,\sigma) \models^l_{me} [x = y]P$	iff $(\rho(me(x)) \cap \rho(me(y)) \neq \emptyset \Rightarrow (\rho,\kappa,\sigma) \models^l_{me} P$
$(\rho,\kappa,\sigma) \models^l_{me} !P$	iff $(\rho,\kappa,\sigma) \models^l_{me} P$
$(\rho,\kappa,\sigma) \models_{me} \langle P\rangle^l$	iff $(\rho,\kappa,\sigma) \models^l_{me} P$
$(\rho,\kappa,\sigma) \models_{me} (\nu x^\chi)S$	iff $(\rho,\kappa,\sigma) \models_{me[x\mapsto\chi]} S$
$(\rho,\kappa,\sigma) \models_{me} S_1 \mid S_2$	iff $(\rho,\kappa,\sigma) \models_{me} S_1 \wedge (\rho,\kappa,\sigma) \models_{me} S_2$
$(\rho,\kappa,\sigma) \models_{me} !S$	iff $(\rho,\kappa,\sigma) \models_{me} S$

EXAMPLE 6.2. Consider the following system S. It consists of the processes R and Q of low level and of P of high level.

- $R = \overline{a}b.\overline{a}b.\overline{b}c$
- $Q = a(x^{\beta_x}).\overline{x}x$
- $P = a(y^{\beta_y}).y(z^{\beta_z}) \cdot ([y = z]\overline{y}a + y(w^{\beta_w}))$
- $S = !(\langle R\rangle^L \mid \langle Q\rangle^L \mid \langle P\rangle^H)$,

where the marker environment me is such that $me\,(fv) = \chi_{fv}$ for all the free names $fv \in \{a, b, c\}$. The triple (ρ, κ, σ) is defined as follows, where the bound names are $bv \in \{x, y, z, w\}$:

$$\rho(\beta_{bv}) = \begin{cases} \{\chi_b\} & \text{if } bv = x, y \\ \{\chi_a, \chi_b, \chi_c\} & \text{if } bv = z, w \end{cases} \qquad \kappa(\chi_{fv}) = \begin{cases} \{\chi_b\} & \text{if } fv = a \\ \{\chi_a, \chi_b, \chi_c\} & \text{if } fv = b \\ \emptyset & \text{if } fv = c \end{cases}$$

$$\begin{array}{ll} \sigma_{in}(L)(\chi_a) = \{\chi_b\} & \sigma_{in}(H)(\chi_a) = \{\chi_b\} \\ \sigma_{in}(L)(\chi_b) = \emptyset & \sigma_{in}(H)(\chi_b) = \{\chi_a, \chi_b, \chi_c\} \\ \sigma_{in}(L)(\chi_c) = \emptyset & \sigma_{in}(H)(\chi_c) = \emptyset \end{array}$$

$$\begin{array}{ll} \sigma_{out}(L)(\chi_a) = \{\chi_b\} & \sigma_{out}(H)(\chi_a) = \emptyset \\ \sigma_{out}(L)(\chi_b) = \{\chi_b, \chi_c\} & \sigma_{out}(H)(\chi_b) = \{\chi_a\} \\ \sigma_{out}(L)(\chi_c) = \emptyset & \sigma_{out}(H)(\chi_c) = \emptyset. \end{array}$$

Recall that $\rho(\chi) = \{\chi\}$. A simple check shows that $(\rho, \kappa, \sigma) \models_{me} S$.

The semantic soundness of our new analysis is stated by the new subject reduction theorem, which is almost indentical to Theorem 3.10. The only differences are the new judgements $(\rho, \kappa, \sigma) \models_{me} S$ and the obvious conditions about the new components σ and l. Its proof is a straightforward adaptation of the proof of Theorem 3.10, so we omit it. We also omit the easy extensions of the relevant lemmata in Section 3. As before, $(\rho, \kappa, \sigma) \models^{\mathcal{C}}_{me} S$ stands for $(\rho, \kappa, \sigma) \models^{\mathcal{C}}_{me} S \wedge me[fn(S)] \subseteq \mathcal{C}$.

THEOREM 6.3. *If* $(\rho, \kappa, \sigma) \models^{\mathcal{C}}_{me} S$ *and* $S \underset{l}{\overset{\mu}{\rightarrow}} S'$ *we have*:

$$\text{if } \mu = \tau \text{ then } (\rho, \kappa, \sigma) \models^{\mathcal{C}}_{me} S'; \tag{1}$$

$$\text{if } \mu = \overline{x}y \text{ then } (\rho, \kappa, \sigma) \models^{\mathcal{C}}_{me} S', me(y) \in \kappa(me(x)), \text{ and } me(y) \in \sigma_{out}(l)(me(x)); \tag{2a}$$

$$\text{if } \mu = \overline{x}(y^{\chi}) \text{ then } (\rho, \kappa, \sigma) \models^{\mathcal{C}}_{me[y\mapsto x]} S', \chi \in \kappa(me(x)) \text{ and } \chi \in \sigma_{out}(l)(me(x)); \tag{2b}$$

$$\text{if } \mu = xy \text{ and } me(y) \in \kappa(me(x)) \text{ then } (\rho, \kappa, \sigma) \models^{\mathcal{C}}_{me} S', \text{ and } me\,(y) \in \sigma_{in}(l)(me(x)); \tag{3a}$$

$$\text{if } \mu = x(y^{\chi}) \text{ and } \chi \in \kappa(me\,(x)) \text{ then } (\rho, \kappa, \sigma) \models^{\mathcal{C}}_{me[y\mapsto\chi]} S', \text{ and } \chi \in \sigma_{in}(l)\,(me(x)), \tag{3b}$$

with $l \in \{L, H\}$ *in all cases, except for the case* (1) *where* $l \in \{L, H\} \cup \{\varepsilon\}$.

A Dynamic Notion

We now introduce the dynamic version of the no read-up/no write-down property. As we did with censored steps, we impose a restriction on the channels that can be read by a process P with a given clearance l. We assume that the environment is always willing to listen to P, but it can select which information is to be transmitted to P. To formalize the intentions of the environment, we use a function

$$\varsigma : \mathcal{L} \rightarrow (\mathcal{C} \rightarrow \wp(\mathcal{C}))$$

that maps a label $l \in \mathcal{L}$ and a channel $\chi \in \mathcal{C}$ to the set of channels that the environment considers secure to communicate to $\langle P \rangle^l$.

Definition 6.4. Given S, *me*, ς, a *granted* step is $(S, me) \overset{\mu}{\underset{l}{\mapsto}} (S', me')$, and is defined whenever

1. $S \overset{\mu}{\underset{l}{\mapsto}} S'$, and
2. (a) if $\mu = xy$, then $me(y) \in \varsigma(l)(me(x))$
   (b) if $\mu = x(y^\chi)$, then $\chi \in \varsigma(l)(me(x))$,

$$\text{where } me' = \begin{cases} me & \text{if } \mu = \tau, \bar{x}y, xy \\ me[y \mapsto \chi] & \text{if } \mu = \bar{x}(y^\chi), x(y^\chi) \end{cases}$$

A granted computation $(S, me) \mapsto^* (S', me') \overset{\mu}{\underset{l}{\mapsto}} (S'', me'')$ is made of granted steps.

The definition of our version of the no read-up/no write-down property follows.

Definition 6.5. A system S is *no read-up/no write-down* (*nru/nwd* for short) with respect to ς, *me* if and only if: whenever $(S, me) \mapsto^* (S', me') \overset{\tau}{\underset{\epsilon}{\mapsto}} (S'', me'')$, where the last granted step is a communication (between S_o and S_i) that has been deduced with either

(a) the rule *ComS*, using the premises $S_o \overset{\bar{x}y}{\underset{l_o}{\rightarrow}} S_o'$ and $S_i \overset{xy}{\underset{l_i}{\rightarrow}} S_i'$, or

(b) the rule *CloseS*, using the premises $S_o \xrightarrow[l_o]{\bar{x}(y^\chi)} S_o'$ and $S_i \xrightarrow[l_i]{x(y^\chi)} S_i'$,

then $l_o = H$ implies $l_i = H$.

A Static Notion

We define now a static property that guarantees that a process is nru/nwd. Besides finding a solution (ρ, κ, σ) for a process P, we require that the channels that can pass along a given channel χ include those that can be read and sent along χ, recorded by the abstract communication structure σ (condition 2a below). More interesting is item (b) of the same condition, where the channels read along χ should include those that the environment is willing to supply, expressed by ς. The last condition is the key condition. It requires that a channel χ cannot be used for sending an object from a process with high level H to a process with low level L.

Definition 6.6. Let S, *me* be such that *me* $[fn(S)] \subseteq \mathcal{C}$. Then S is *discreet* (w.r.t. ς, *me*) if and only if there exists (ρ, κ, σ) such that

1. $(\rho, \kappa, \sigma) \models_{me} S$
2. $\forall l \in \{L, H\}, \chi \in \mathcal{C}$: (a) $\kappa(\chi) \supseteq \sigma_{in}(l)(\chi) \cup \sigma_{out}(l)(\chi)$
   (b) $\sigma_{in}(l)(\chi) \supseteq \varsigma(l)(\chi)$
3. $\forall \chi \in \mathcal{C} : \sigma_{out}(H)(\chi) \cap \sigma_{in}(L)(\chi) = \emptyset$.

Below, we show that the property of being discreet is preserved under granted steps.

Lemma 6.7 (Subject Reduction for Discreetness). *If S is discreet with respect to ς, me, and $(S, me) \overset{\mu}{\underset{l}{\mapsto}} (S', me')$, then S' is discreet with respect to ς, me'.*

Proof. Theorem 6.3 suffices to prove both $me[fn(S')] \subseteq \mathcal{C}$ and $(\rho, \kappa, \sigma) \models_{me} S'$. The proof of the second and third items of discreetness is immediate, because the solution does not change. The only delicate point for the application of Theorem 6.3 is when the granted step is an input, while in the other cases the proof is trivial. For input, consider first the case $\mu = xy$. It suffices to show that $me(y) \in \kappa(me(x))$. Conditions (2a) and (2b) of Definition 6.6 guarantee that $\kappa(me(x)) \supseteq \sigma_{in}(l)(me(x)) \supseteq \varsigma(l)(me(x))$. Furthermore, $me(y) \in \varsigma(l)(me(x))$ because the step is granted. The case when $\mu = x(y^\chi)$ is similar. ■

Theorem 6.8. *If S is discreet* (w.r.t. ς, *me*), *then S is nru/nwd* (w.r.t. ς, *me*).

Proof. Since our computations are granted, by Lemma 6.7 it is enough to check that, if $(S, me) \overset{\tau}{\underset{\epsilon}{\mapsto}} (S', me')$ then S' is nru/nwd, with τ being a communication between S_o and S_i, defined as in Definition 6.5. Without loss of generality we may assume that all restrictions in S occur outermost or inside some $\langle \cdots \rangle$. Assume, per absurdum, that $l_o = H$ and $l_i = L$. Then there exists $\langle P_o \rangle^H$ and $\langle P_i \rangle^L$ such that either

(a) $\langle P_o \rangle^H \overset{\bar{x}y}{\underset{H}{\rightarrow}} \langle P_o' \rangle^H$ and $\langle P_i \rangle^L \overset{xy}{\underset{L}{\rightarrow}} \langle P_i' \rangle^L$, or

(b) $\langle P_o \rangle^H \xrightarrow[H]{\bar{x}(y^\chi)} \langle P_o' \rangle^H$ and $\langle P_i \rangle^L \xrightarrow[L]{x(y^\chi)} \langle P_i' \rangle^L$,

given the assumptions about S_o and S_i. Given that $(\rho, \kappa, \sigma) \models_{me} S$ we obtain that $(\rho, \kappa, \sigma) \models_{\overline{me}} \langle P_o \rangle^H$ and $(\rho, \kappa, \sigma) \models_{me} \langle P_i \rangle^L$, for some suitable $\overline{me}$ taking care of the restrictions mentioned above. The analysis and Theorem 6.3 tell us that

(a) $\overline{me}(y) \in \sigma_{out}(H)(\overline{me}(x))$ and $\overline{me}(y) \in \sigma_{in}(L)(\overline{me}(x))$, or

(b) $\chi \in \sigma_{out}(H)(\overline{me}(x))$ and $\chi \in \sigma_{in}(L)(\overline{me}(x))$.

This contradicts item 3 of Definition 6.6, that demands the intersection of the two sets to be empty. ■

EXAMPLE 6.9. It is easy to prove that the process S validated in Example 6.2 is discreet. In particular, the following two conditions hold.

- $\sigma_{out}(H)(\chi_a) \cap \sigma_{in}(L)(\chi_a) = \emptyset \cap \{\chi_b\} = \emptyset$.
- $\sigma_{out}(H)(\chi_b) \cap \sigma_{in}(L)(\chi_b) = \{\chi_a\} \cap \emptyset = \emptyset$.

As was the case for confinement, the polynomial time algorithm of Section 4 can be used to perform a polynomial time check of whether or not a process P is discreet with respect to given ς, me.

7. RELATED WORK

There are two strains of related work: the first concerns static analysis techniques, the second (mainly) their applications to security.

Static analysis of programs aims at developing efficient techniques that may be used to obtain approximate answers about how a program executes without actually executing it. Traditionally, this has been a main component in the construction of efficient implementations of programming languages. There is a vast literature on static program analysis and we here survey some of the more important approaches; we refer to [31] for further details.

Control Flow Analysis (e.g., [40]) deals with the problem of dynamic dispatch where it is *not* apparent which function (or method) is actually being invoked and where an analysis is needed in order to determine this information. This is a problem that appears all the time in higher order functional languages and many developments of control flow analysis have indeed been performed for functional languages, but also object-oriented languages [35] and languages with concurrency [18] can be addressed.

Recently, Venet [41, 42] has used Abstract Interpretation [11] to analyse processes in a fragment of the π-calculus, with particular attention to the usage of channels. Type Systems are intimately connected to ensuring the well-formedness of programs. In recent years they have been extended with annotations and effects and are becoming a very popular tool for the analysis of calculi of computations (e.g., [1, 9, 34, 43, 44]); in our view, type systems are particularly useful when they admit principal types, and this is not the case for all of the developments cited.

These approaches are not so dissimilar as might appear at first sight. In fact, Control Flow Analyses can be expressed in the Constraint Based Formulation used here, in the terminology of Type Systems and using abstract interpretation. We refer to [10, 31] and their references for further information about the relationships among these approaches.

We now turn to the second strain of related work. The first studies in system security reach back to the 1970s and were mainly carried out in the area of operating systems; see the detailed survey by

Landwehr [24]. Denning's book [14] reports her pioneering work, inspired also by the work by Bell and LaPadula [5], Fenton [15], and Lampson [23]. They developed methods for detecting violations of secure flow while statically analysing the code.

Recently, security classes have been formalized as types and the control of flow is based on type checking. Heintze and Riecke [22] refine Denning's analysis and study a noninterference property on the SLam Calculus (Secure λ-calculus). Well-typed processes are such that values of low-level expressions are independent of the values of high-level parameters. Volpano *et al.* develop a type system to ensure secure information flow in a sequential imperative language [44]; this work was later extended by the first two authors in a concurrent, shared-memory based setting [43].

The work closest to our proposal in Section 5 is that by Abadi [1], who studies the secrecy of channels and of encrypted messages using the *spi*-calculus, an extension of the π-calculus devised for writing secure protocols. His is a more ambitious goal than ours, because the *spi*-calculus also has cryptographic primitives. Semantic correctness of the type system, formalised using testing equivalence, guarantees that there is no leaking of secret information. 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 (in the manner of Exercise 5.4 of [31]) and our notion of validation corresponds to validate the type annotation. However, whereas we established that a best solution always exists (via Moore families) and gave a procedure for constructing solutions, Abadi [1] does not establish the existence of the analogous notion of principal types, and hence it is unclear whether or not a sound and complete typing algorithm exists. The semantic correctness is checked against two different dynamic notions (testing equivalence versus no leaks).

Other interesting papers in this area are [4, 12, 13, 17, 36–39]. Particularly relevant are Hennessy and Riely's papers [37, 38] that give a type system for Dπ, a variant of the π-calculus with explicit sites that harbour mobile processes. A well-typed process correctly uses its capabilities and never compromises the integrity of well-behaved sites. Recently also Cardelli and Gordon [9] proposed a type system for the ambient calculus ensuring that a well-typed mobile computation cannot cause certain kinds of run-time faults, even when capabilities can be exchanged between processes.

The idea of static analysis for security has been followed also in the Java world, for example in the Java Bytecode Verifier [25]. Also, Abadi faces in [2] the problem of implementing secure systems and proposes to use full abstraction to check that the compiled code enjoys the same security properties of the source program.

The dynamic point of view has been adopted by a certain number of information flow models [16, 19, 26, 27] (to cite only a few). Here, only the external observable behaviour is the object of the analysis. A classical example is the noninterference model of [19, 20], where the actions of the group of high level users have no effect on what the group of low level users can observe. McCullough [26, 27] extends this definition to cope with non determinism. Focardi and Gorrieri [16] use a process algebras setting, by using SPA (Security Process Algebra), an extension of CCS [28]. Our property of no read-up/no write-down could be considered a variant of the noninterference property, studied by most of the authors mentioned above.

8. CONCLUSIONS

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 role of the placeholder y; this change was motivated by the inclusion of α-conversion in the semantics. Annotations can be done mechanically and do not affect the behaviour of processes; typically, all markers will be different. The results of our analysis approximate the actual solutions, because they may give a super-set of the corresponding actual values.

We defined judgements on solutions and processes and a set of clauses that operate on judgements so as to validate the correctness of the solution. We proved that a best solution always exists and we also presented a constructive procedure for generating solutions, which is in $O(N^5)$.[2] Note that the only

[2] A recent result [33] shows that time complexity can be reduced to $O(n^3)$.

point where we addressed reachability is in the two clauses for matching in Tables 2 and 5. There, we avoid analysing the "continuation" if the match cannot possibly be passed. Similar considerations could be applied to input and output actions, so as to avoid analysing the continuation, e.g., in case the empty set is associated with the subject of the action (see [8]).

We described how our analysis can be used to establish two simple properties of security. The first property we called confinement, the second property we called discreetness. The two properties are orthogonal, as the first puts constraints on the communications to the environment, and the second property enforces a discipline on the communications internal to the process under analysis. So, one can separately check them on a process.

An immediate extension to our (two-level) notion of confinement consists of defining a hierarchy of classification levels associated with channels. Equally immediate is extending our analysis in Section 5 to statically check that a high level of information is never transmitted on a channel with a lower level of security classification.

We did not consider, but we plan to, the more general notion of the no read-up/no write-down property that assigns levels of confidentiality also to the exchanged data (i.e., the objects of input and output actions). Processes with low level clearance are then not allowed to access (i.e., they can neither send nor receive) highly classified data. Note in passing that (an extension of) our first security property is not enough for getting the full version of the nru/nwd property; indeed, processes with different clearances are allowed to send objects with different classification along the same channel, while this is forbidden in the case considered in Section 5. A further extension to the nru/nwd property might consider to have a partial order of clearance levels (see Chapter 5 of [14]).

Future work will consider calculi more oriented to security. The last two authors have already considered in [21, 32] the ambient calculus [9] that extends the π-calculus with an explicit notion of mobility of computation. The main application concerns validating the protectiveness of a firewall, meaning that it does not allow agents to enter unless they know the required passwords. We are also interested in the *spi*-calculus [3] that enriches the π-calculus with primitives for encryption and decryption and distinguishes between data and channels. The core analysis for the *spi*-calculus should remain the same as presented here, while some sort of data flow analysis seems necessary to track data manipulations. It would also be interesting to study authentication properties. Preliminary work on a control flow analysis for the spi-calculus can be found in Bodei's thesis [6], where a notion of secrecy and a restricted form of authentication are considered.

ACKNOWLEDGMENTS

We thank Martín Abadi, Andrew Gordon, Roberto Gorrieri, and Dennis Volpano for interesting discussions. Special thanks are due to the referees for their precise and invaluable suggestions.

A preliminary version of Sections 1, 2, 3.1, and 4 appeared in [7]; some ideas of Section 5 are presented in [8]. We thank René Rydhof Hansen for constructing two web-based systems for checking confinement and discreetness based on the algorithms reported here. They are available from at http://www.daimi.au.dk/∼fn/FlowLogic.html under the entries for [7] and [8].

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.

REFERENCES

1. Abadi, M. (1999), Secrecy by typing in security protocols, *J. ACM* **46**(5), 749–786, September 1999. Extended Abstract, *in* "Proceedings of Theoretical Aspects of Computer Software 1997," pp. 611–638, Springer-Verlag, Berlin/New York. [Extended version to appear in *J. ACM*]
2. Abadi, M. (1998), Protection in programming-language translations, *in* "Proceedings of ICALP'98," Lecture Notes in Computer Science, Vol. 1443, pp. 868–883, Springer-Verlag, Berlin/New York.
3. Abadi, M., and Gordon, A. D. (1999), A calculus for cryptographic protocols—The Spi calculus, *Inform. and Comput.* **148**(1), 1–70.
4. Amadio, R. M. (1997), An asynchronous model of locality failure, and process mobility, *in* "Proceedings of COORDINATION'97," Lecture Notes in Computer Science, Vol. 1282, pp. 374–391, Springer-Verlag, Berlin/New York.
5. Bell, D. E., and LaPadula, L. J. (1976), "Secure Computer Systems: Unified Exposition and Multics Interpretation," Technical Report ESD-TR-75-306, Mitre C.

6. Bodei, C. (2001), "Security Issues in Process Calculi," Ph.D. thesis, TD-2/00 Department of Computer Science, University of Pisa.
7. Bodei, C., Degano, P., Nielson, F., and RiisNielson, H. (1998), Control flow analysis for the π-calculus, *in* "Proceedings of CONCUR'98," Lecture Notes in Computer Science, Vol. 1466, pp. 84–98, Springer-Verlag, Berlin/New York.
8. Bodei, C., Degano, P., Nielson, F., and Riis Nielson, H. (1999), Static analysis of processes for no read-up and no write-down, *in* "Proceedings of FoSSaCS'99," Lecture Notes in Computer Science, Vol. 1578, pp. 120–134, Springer-Verlag, Berlin/ New York.
9. Cardelli, L., and Gordon, A. D. (1998), Mobile ambients, *in* "Proceedings of FoSSaCS'98," Lecture Notes in Computer Science, Vol. 1378, pp. 140–155, Springer-Verlag, Berlin/New York.
10. Cousot, P. (1997), Types as abstract interpretations, *in* "Proceedings of POPL'97," pp. 316–331, Assoc. Comput. Mach. Press, New York.
11. Cousot, P., and Cousot, R. (1979), Systematic design of program analysis frameworks, *in* "Proceedings of POPL'79," pp. 269–282, Assoc. Comput. Mach. Press, New York.
12. Dam, M. (1998), Proving trust in systems of second-order processes (extended abstract), *in* "Proceedings of HICSS 31," Vol. VII, pp. 255–264, IEEE Comput Soc., Los Alamitoes, CA.
13. De Nicola, R., Ferrari, G., and Pugliese, R., Venneri, E. (2000), Types for access control, *Theoret. Comput. Sci.* **240**(1), 215–254.
14. Denning, D. E. (1982), "Cryptography and Data Security," Addison-Wesley, Reading MA.
15. Fenton, J. (1979), "Information Protection Systems," Ph.D. thesis, University of Edinburgh.
16. Focardi, R., and Gorrieri, R. (1997), The compositional security checker: A tool for the verification of information flow security properties, *IEEE Trans. Software Eng.* **23**(9).
17. Fournet, C., Laneve, C., Maranget, L., and Remy, D. (1997), Implicit typing à la ML for the join calculus, *in* "Proceedings of CONCUR'97," Lecture Notes in Computer Science, Vol. 1243, pp. 196–212, Springer-Verlag, Berlin/New York.
18. Gasser, K. L. S., Nielson, F., and Nielson, H. R. (1997), Systematic realisation of control flow analysis CML, *in* "Proceedings of ICFP' 97," pp. 38–51, Assoc. Comput. Mach. Press, New York.
19. Goguen, J. A., and Meseguer, J. (1982), Security policy and security models, *in* "Proceedings of the 1982 IEEE Symposium on Research on Security and Privacy," pp. 11–20, IEEE Press, New York.
20. Goguen, J. A., and Meseguer, J. (1984), Unwinding and inference control, *in* "Proceedings of the Symposiom on Security and Privacy," pp. 75–87, IEEE Comput. Soc., Los Alamitos, CA.
21. Hansen, R. R., Jensen, J. G., Nielson, F., and Nielson, H. R. (1999), Abstract interpretation of mobile ambients, *in* "Proceedings of SAS' 99," Lecture Notes in Computer Science, Vol. 1694, pp. 135–148. Springer-Verlag, Berlin/New York.
22. Heintze, N., and Riecke, J. G. (1998), The SLam calculus: Programming with secrecy and integrity, *in* "Proceedings of POPL '98," pp. 365–377, Assoc. Comput. Mach. Press, New York.
23. Lampson, B. W. (1973), A note on the confinement problem, *Comm. Assoc. Comput. Mach.* **16**(10), 613–615.
24. Landwehr, C. E. (1981), Formal models for computer security, *Assoc. Comput. Mach. Comput. Serveys* **13**(3), 247–278.
25. Lindholm, T., and Yellin, F. (1996), "The Java Virtual Machine Specification," Addison-Wesley, Reading, MA.
26. McCullough, D. (1987), Specifications for multi-level security and hook-up property, *in* "Proceedings of the 1987 IEEE Symposium on Research on Security and Privacy," IEEE Press, New York.
27. McCullough, D. (1988), Noninterference and the composability of security properties, *in* "Proceedings of the 1988 IEEE Symposium on Research on Security and Privacy," pp. 177–186, IEEE Press, New York.
28. Milner, R. (1989), "Communication and Concurrency," Prentice-Hall, London.
29. Milner, R., Parrow, J., and Walker, D. (1992), A calculus of mobile processes (I and II), *Inform. and Comput.* **100**(1), 1–77.
30. Milner, R., Parrow, J., and Walker, D. (1993), Modal logics for mobile processes, *Theoret. Comput. Sci.* **114**(1), 49–171.
31. Nielson, F., Nielson, H. R., and Hankin C. (1999), "Principles of Program Analysis, Springer-Verlag, Berlin/New York.
32. Nielson, F., Nielson, H. R., Hansen, R. R., and Jensen, J. G. (1999), Validating firewalls in mobile ambients, *in* "Proc. Concur'99," Lecture Notes in Computer Science, Vol. 1664, pp. 463–477.
33. Nielson, F., and Seidl, H. (2001), Control-Flow Analysis in Cubic Time, in "Proceedings of ESOP 2001," LNCS, Vol. 2028, pp. 252–268, Springer-Verlag, Berlin/New York.
34. Ørbæk, P., and Palsberg, J. (1997), Trust in the λ-calculus, *J. Funct. Programming* **7**(6), 557–591.
35. Palsberg, J., and Schwartzbach, M. J. (1994), "Object-Oriented Type Systems," Wiley, New York.
36. Pierce, B. C., and Sangiorgi, D. (1996), Typing and sub-typing for mobile processes, *Math. Structures Comput. Sci.* **6**(5), 409–454.
37. Riely, J., and Hennessy, M. (1998), A typed language for distributed mobile processes, *in* "Proceedings of POPL'98," pp. 378–390, Assoc. Comput. Mach. Press, New York.
38. Riely, J., and Hennessy, M. (1999), Trust and partial typing in open systems of mobile agents, *in* "Proceedings of the 26th Symposium on Principles of Programming Languages (POPL' 99)," pp. 93–104, Assoc. Comput. Mach. Press, New York.
39. Sewell, P. (1998), Global/local subtyping and capability inference for a distributed π-calculus, *in* "Proceedings of ICALP'98," Lecture Notes in Computer Science, Vol. 1443, pp. 695–706. Springer-Verlag, Berlin/New York.
40. Shivers, O. (1988), Control flow analysis in Scheme, *in* "Proceedings of PLDI'88," Vol. 7(1), ACM SIGPLAN Notices.
41. Venet, A. (1997), Abstract interpretation of the π-calculus, *in* "Analysis and Verification of Multiple-Agent Languages," Lecture Notes in Computer Science, Vol. 1192, pp. 51–75, Springer-Verlag, Berlin/New York.
42. Venet, A. (1998), Automatic determination of communications topologies in mobile systems, *in* "Proceedings of SAS'98," Lecture Notes in Computer Science Vol. 1503, pp. 152–167. Springer-Verlag, Berlin/New York.
43. Volpano, D., and Smith, G. (1998), Secure information flow in a multi-threaded imperative language, *in* "Proceedings of POPL'98," pp. 355–364, Assoc. Comput. Mach. Press, New York.
44. Volpano, D., Smith, G., and Irvine, C. (1996), A sound type system for secure flow analysis, *J. Comput. Security* **4**, 4–21.