A Privacy Analysis for the π-calculus: The Denotational Approach

B. Aziz and G.W. Hamilton
School of Computer Applications,
Dublin City University,
Dublin 9, Ireland
{baziz,hamilton}@computing.dcu.ie

Abstract

We present a non-uniform static analysis for the π-calculus that is built on a denotational semantics of the language and is useful in detecting instances of information leakage and insecure communications in systems with multi-level security policies. To ensure the termination of the analysis, we propose an abstraction, which maintains a finite number of names to be generated by any process. We prove the safety of the analysis and review a prototype of the analysis called the Picasso tool.

Keywords: π-calculus, static analysis, security, denotational semantics

1 Introduction

Privacy of information has always been one of the most important aspects of security in computer systems. In any environment where multiple users share computing resources, the need for protecting against unauthorized access to confidential data becomes a necessity. Such protection becomes harder to guarantee in the presence of the emerging mobile technologies, since networks and communication topologies continuously change and the possibility of interacting with unknown contexts is ever more likely.

In this paper, we present a static analysis for the privacy of information in mobile systems specified using the language of the π-calculus [25]. The analysis is based on the abstract interpretation framework [11, 12] and uses a denotational semantics for the π-calculus based on Stark's abstract model [27, 2] (other denotational models for the π-calculus include [10, 16, 19, 20]). The analysis is aimed at detecting two types of privacy breaches in systems with multilevel security: *information leakage* and *insecure communications*. The former is concerned with capturing the movement of data between processes, where a high-level datum is leaked whenever a low-level process obtains it. The latter is concerned with capturing the movement of data over channels, where a high-level datum is insecurely communicated whenever it is passed over a low-level channel. One advantage of this security model over other models like the Bell and La Padula (BLP) model [3] is that privacy is dependent on the levels of the communicated data rather than the levels of the communicating processes. This offers extra flexibility whenever processes with different levels wish to communicate.

The main novelty about this work is the use of the denotational semantics as a basis for an abstract interpretation for mobile systems. This semantics is defined to include a renaming mechanism that traces copies of names back to their origin, which is normally lost as a result of α-conversion. A non-standard interpretation of processes in the π-calculus is then given that associates with each input parameter the set of names that can substitute it during communications. This semantics is abstracted by limiting the number of names, which can be generated within a process, hence ensuring the termination of the analysis.

The rest of this paper is structured as follows. After comparing with related work in Section 2, we review the syntax and denotational semantics of the π-calculus in Section 3. The non-standard semantics is introduced in Section 4 to capture the substitution of names. This semantics is proven to be correct with respect to the standard semantics. In Section 5, we introduce the abstract semantics and prove its safety. In Section 6, we demonstrate, with an example, how the abstract semantics may be used in analyzing closed systems, whereas in Section 7, we extend the analysis to deal with open systems. Finally, in Section 8, we conclude the work and briefly discuss future directions.

2 Related Work

In recent years, the use of the static analysis approach in analyzing mobile systems and verifying their security properties has grown into a major area of research where a variety of techniques have been employed to varying degrees of success. Therefore, we shall only give here a few examples of static analyses that have been applied to the π-calculus. In [21, 22, 24], type checking systems are used to control access to resources and the flow of information in various versions of the π-calculus extended with multilevel security. In [22], a combination of location types and capabilities is used to determine the set of local resources a particular process has access to. The notion of *non-interference* [17] is formalized in [21] in terms of typed traces to ensure that no implicit flow of confidential information occurs, while [24] employs an enriched syntax and an advanced typing system. Recently, types have been explored in [23] in a trust analysis to guarantee that a trusted context does not use untrusted data, and in [9], dynamically created sorts called *groups* have been used to achieve non-uniformity of properties. One disadvantage of the type-based approach is that type systems do not always admit principal types.

Control flow techniques have also been utilized in [5, 6, 7] to determine the set of names that a name may be bound to and the set of channels that may be sent over a channel. This information is used in [5] to infer that a process confines secret names to itself, where proposed solutions are verified and a least solution is shown to always exist. This analysis is extended in [6] to include the BLP property and a constructive procedure is further presented in [7]. One example of the limitations shared by these analyses is the inability to detect deadlocks, as a result of the inability to interpret restrictions.

The abstract interpretation framework [11, 12] is often used in situations where safe (but imprecise) abstractions of the runtime properties are acceptable. In [14], a non-uniform analysis is presented for detecting instances of leakage of confidential data in the presence of unknown contexts. The analysis builds on the work of [29], which presents a sound and non-uniform description of how topologies evolve in closed π-calculus specifications without nested replications. The same approach is utilized again in [15] in an occurrence counting analysis for detecting the exhaustion of resources, mutual exclusion, and deadlocks. Although these analyses reveal some interesting results, they lack simplicity as they all build on a small-step structural operational semantics, which requires sophisticated abstractions to trace the movement of data.

3 The Language

We adopt the standard version of the π-calculus as shown in Figure 1. Processes are $P, Q, \ldots \in \mathcal{P}$ and names are $x, y, \ldots \in \mathcal{N}$ possibly subscripted with numbers. A process is constructed from standard π-calculus terms, where conditional statems replace the matching/mismatching constructs and infinite behaviour is expressed by replication. The notions of substitution and α-conversion as well

P	:=	$\mathbf{0}$	Null
		$x(y).P$	Input action
		$\overline{x}\langle y\rangle.P$	Output action
		$\tau.P$	Silent action
		$\mathit{if}\ [x=y]\ \mathit{then}\ P\ \mathit{else}\ Q$	Conditional
		$P+Q$	Summation
		$P \mid Q$	Parallel Composition
		$(\nu x)P$	Restriction
		$!P$	Replication

Figure 1: Syntax of the π-calculus.

as free names *fn(P)*, bound names *bn(P)* and the set of all names of a process $n(P) = \mathit{fn(P)} \cup \mathit{bn(P)}$ also apply. We assume that all the names in *n(P)* are initially distinct, particularly $\mathit{fn(P)} \cap \mathit{bn(P)} = \emptyset$. Later in the analysis, we shall employ the notion of $act(P)$ to signify the set of all the actions that are present in the specification of P, as defined by the following:

$act(\mathbf{0}) = \emptyset$
$act(\pi.P) = \{\pi\} \cup act(P)$, for $\pi \in \{x(y), \overline{x}\langle y\rangle, \tau\}$
$act(\mathit{if}\ [x=y]\ \mathit{then}\ P\ \mathit{else}\ Q) = act(P) \cup act(Q)$
$act(P+Q) = act(P) \cup act(Q)$
$act(P \mid Q) = act(P) \cup act(Q)$
$act((\nu x)P) = act(P)$
$act(!P) = act(P)$

Our standard semantics of the π-calculus is inspired by the domain-theoretic model of name-passing processes suggested by Stark [27] and modified in [2]. The model is based on functor categories and is motivated by the solution to the following predomain equations, which describe a process in terms of the basic actions it can perform (input/output, deadlock, and termination):

$$Pi \cong 1 + P(Pi_\perp + In + Out) \tag{1}$$
$$In \cong N \times (N \to Pi_\perp) \tag{2}$$
$$Out \cong N \times (N \times Pi_\perp + N \multimap Pi_\perp) \tag{3}$$

Where $Pi_\perp$ is the object of processes, N is the object of names, and *In* and *Out* represent input and output actions respectively, allowing for bound outputs as in $(\nu y)\overline{x}\langle y\rangle$ in order to be able to express scope extrusion. $P(-)$ is a power operation taken as the adaptation of Plotkin's (convex) powerdomain $P^\natural$ to bifinite predomains [26] whereas $\multimap$ is a non-standard exponential taken as the lifted function space that supplies a fresh name to a process that uses it.

The solution to equations (1–3) is constructed within a functor category C that is symmetric monoidal closed. In order to build C, an initial category B^I is suggested, where B is the category of bifinite predomains and continuous maps

described in [26] and I is the category of finite sets of names, $s \subseteq \wp(\mathcal{N})$, and injections between those sets, $f : s \to s'$. The definition of $Pi_\perp$ is then a functor $I \to B$, where $Pi_\perp s$ signifies the domain of all processes with free names in s, and $Pi_\perp f : Pi_\perp s \to Pi_\perp s'$ denotes a re-labeling operator. Finally, the object of names N is taken as the inclusion from a set of names s to a discrete predomain s (this expresses the lack of structure in names).

Within C, a number of morphisms leading into $Pi_\perp$ are defined as follows:

$$\emptyset : 1 \to Pi_\perp \tag{4}$$

$$\{\!|-|\!\} : (Pi_\perp + In + Out)_\perp \to Pi_\perp \tag{5}$$

$$\uplus : Pi_\perp \times Pi_\perp \to Pi_\perp \tag{6}$$

$$new : (N \multimap Pi_\perp) \to Pi_\perp \tag{7}$$

These morphisms describe the manner in which processes are constructed in the π-calculus. The $\emptyset$ morphism denotes terminated and deadlocked processes. The singleton map $\{\!|-|\!\}$ is used to interpret input, free/bound output, and silent actions. The least element $\{\!|\perp|\!\}$ represents the undefined process where $\{\!|\perp|\!\} \sqsubseteq \emptyset$ and $\emptyset$ is incomparable otherwise. Finally, the $\uplus$ morphism is a standard powerdomain union representing non-deterministic choice between two processes while the *new* morphism is used to interpret restrictions.

In order to be able to build an abstract interpretation, a concrete definition of the semantic domain is necessary. This is achieved by explicitly specifying finite elements $p, q \ldots \in Pi_\perp s$ using morphisms (4–6). Name binding is expressed as a λ-abstraction of the form $\lambda y.p$, where a name y is bound to a process p. An input action thus is represented as $in(x, \lambda y.p)$ and a bound output action as $out(x, \lambda y.p)$. The remaining free output and silent actions are represented as $out(x, y, p)$ and $tau(p)$, respectively.

A concrete definition of the *new* morphism over s interprets restriction in terms of morphisms (4–6). This definition is illustrated in Figure 2 and a comprehensive explanation of the rules can be found in [28, 2]. Conceptually, these

$$
\begin{array}{lcl}
new_s(\lambda x.\emptyset) & = & \emptyset \\
new_s(\lambda x.\{\!|\perp|\!\})\ \phi & = & \{\!|\perp|\!\} \\
new_s(\lambda x.\{\!|in(y, \lambda z.p)|\!\}) & = & \begin{cases} \emptyset, & \text{if } x = y \\ \{\!|in(y, \lambda z.new_{s\cup\{z\}}(\lambda x.p))|\!\}, & \text{otherwise} \end{cases} \\
new_s(\lambda x.\{\!|out(y, z, p)|\!\}) & = & \begin{cases} \emptyset, & \text{if } x = y \\ \{\!|out(y, \lambda z.p)|\!\}, & \text{if } x = z \neq y \\ \{\!|out(y, z, new_s(\lambda x.p))|\!\}, & \text{otherwise} \end{cases} \\
new_s(\lambda x.\{\!|out(y, \lambda z.p)|\!\}) & = & \begin{cases} \emptyset, & \text{if } x = y \\ \{\!|out(y, \lambda z.new_{s+\{z\}}(\lambda x.p))|\!\}, & \text{otherwise} \end{cases} \\
new_s(\lambda x.\{\!|tau(p)|\!\}) & = & \{\!|tau(new_s(\lambda x.p))|\!\} \\
new_s(\lambda x.(p_1 \uplus p_2)) & = & new_s(\lambda x.p_1)\ \uplus\ new_s(\lambda x.p_2)
\end{array}
$$

Figure 2: The concrete definition of *new* over s.

rules capture deadlocked situations arising from an attempt to communicate over a fresh channel, and scope extrusion situations where free outputs are turned into bound outputs. In all the other situations, restriction has no effects. It is important to note here that the freshness requirement imposed by $\multimap$ in the definition of *new* is implemented later in the semantics by a special labeling mechanism.

Using the concrete definition of $Pi_\perp s$, the interpretation of a process P with free names in s can be given as an element $\mathcal{S}(\![P]\!)_s\ \rho \in Pi_\perp s$. The rules of this interpretation are shown in Figure 3, where the parameter ρ represents a multiset containing all the processes composed in parallel with P. We use the singleton $\{\!|-|\!\}_\rho$ and the multiset union $\uplus_\rho$ operators over ρ, which should not be confused with the morphisms $\{\!|-|\!\}$ and $\uplus$ over $Pi_\perp$.

$$
\begin{array}{lll}
(\mathcal{S}1) & \mathcal{S}(\![\mathbf{0}]\!)_s\ \rho & = \emptyset \\
(\mathcal{S}2) & \mathcal{S}(\![x(y).P]\!)_s\ \rho & = \{\!|in(x, \lambda y.(\mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)_{s\cup\{y\}}))|\!\} \\
(\mathcal{S}3) & \mathcal{S}(\![\overline{x}\langle y\rangle.P]\!)_s\ \rho & = \biguplus_{x(z).P'\in\rho} (\mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho[P'[y/z]/x(z).P']]\!)_s) \uplus \\
 & & \quad \{\!|out(x, y, \mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)_s)|\!\} \\
(\mathcal{S}4) & \mathcal{S}(\![\tau.P]\!)_s\ \rho & = \{\!|tau(\mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)_s)|\!\} \\
(\mathcal{S}5) & \mathcal{S}(\![\mathit{if}\ [x=y]\ \mathit{then} & \\
 & \quad P\ \mathit{else}\ Q]\!)_s\ \rho & = \begin{cases} \mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)_s, & \text{if } x = y \\ \mathcal{R}(\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)_s, & \text{otherwise} \end{cases} \\
(\mathcal{S}6) & \mathcal{S}(\![P+Q]\!)_s\ \rho & = (\mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)_s) \ \uplus\ (\mathcal{R}(\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)_s) \\
(\mathcal{S}7) & \mathcal{S}(\![P \mid Q]\!)_s\ \rho & = \mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)_s \\
(\mathcal{S}8) & \mathcal{S}(\![(\nu x)P]\!)_s\ \rho & = new_s(\lambda x_i.\mathcal{R}(\![\{\!|P[x_i/x]|\!\}_\rho \uplus_\rho \rho]\!)_{s+\{x_i\}}) \\
 & & \quad \text{where, } i = min\{i \mid i \in \mathbb{N} \wedge x_i \notin s\} \\
(\mathcal{S}10) & \mathcal{S}(\![!P]\!)_s\ \rho & = \mu p.p \ \uplus\ \mathcal{S}(\![P]\!)_s\ (\{\!|!P|\!\}_\rho \uplus_\rho \rho) \\
(\mathcal{R}0) & \mathcal{R}(\![\rho]\!)_s & = \biguplus_{P\in\rho} \mathcal{S}(\![P]\!)_s\ (\rho \backslash \{\!|P|\!\}_\rho)
\end{array}
$$

Figure 3: Standard semantics of the π-calculus.

The meaning of the composed processes in ρ is given by rule $(\mathcal{R}0)$ as an interleaving of all possible the actions within these processes. Rule $(\mathcal{S}1)$ interprets the meaning of a null process directly as the $\emptyset$ morphism. Rules $(\mathcal{S}2) - (\mathcal{S}4)$ deal with the cases of processes guarded by input, output, and silent actions after which the residues are composed with the elements of ρ. In $(\mathcal{S}2)$, the input parameter y is joined to s as it is free (though not necessarily fresh) within the residue. In $(\mathcal{S}3)$, when interpreting an output action, communication may or may not take place according to the availability of processes in ρ that are guarded by the appropriate input action. If it does take place, the residues of those processes are modified accordingly. Rules $(\mathcal{S}5)$ and $(\mathcal{S}6)$ deal with match and mismatch respectively. If the condition of the rule holds, the residue P is composed with the processes in ρ, else it blocks. Rule $(\mathcal{S}7)$ interprets each of the processes in a summation as composed with the processes in ρ. The alternatives are combined by the $\uplus$ morphism. Rule $(\mathcal{S}8)$ interprets the meaning of

two parallel processes as being composed with the rest of processes in ρ. Rule ($\mathcal{S}9$) introduces a labeling mechanism into restriction which ensures that the created name is always distinct from previous copies. This mechanism successively generates names $x_1, x_2 \ldots$ each time (νx) is applied. This removes the need for α-conversion and allows the new copies to be traced back to their origin. Finally, in ($\mathcal{S}10$), replication is interpreted recursively over ρ using a fixed-point of the choices of processes p.

4 Non-standard Semantics

The standard meaning of a process in the π-calculus as an element of the $Pi_{\perp}s$ domain will be modified in this section to a non-standard meaning, which determines the set of names to which a name in a process can be bound. To facilitate this, we first introduce a name environment $\phi_{\mathcal{E}} : \mathcal{N} \rightarrow \wp(\mathcal{N})$ initially mapping each name to the empty set. Hence, $\forall x \in \mathcal{N} : \phi_{\mathcal{E}0}(x) = \emptyset$. Informally, $\phi_{\mathcal{E}}$ allows for any name substitutions resulting from communications to be captured. For example, if a message z is received by the input action $x(y).P$ (which is substituted for y), then z will be added to the set $\phi_{\mathcal{E}}(y)$. We shall refer to the total function space of $\phi_{\mathcal{E}}$ environments as $D = \mathcal{N} \rightarrow \wp(\mathcal{N})$ ordered by set inclusion as follows:

$$\phi_{\mathcal{E}1} \sqsubseteq_D \phi_{\mathcal{E}2} \textit{ iff } \forall x \in \mathcal{N}, \phi_{\mathcal{E}1}(x) \subseteq \phi_{\mathcal{E}2}(x)$$

The bottom element is $\phi_{\mathcal{E}0}$, where $\forall \phi_{\mathcal{E}} \in D, \phi_{\mathcal{E}0} \sqsubseteq_D \phi_{\mathcal{E}}$.

The non-standard interpretation of a π-calculus process P is now defined as an element $\mathcal{E}(\llbracket P \rrbracket)_s\ \rho\ \phi_{\mathcal{E}} \in D$ on the structure of P by the set of rules of Fig. 4. Again the interpretation uses a ρ parameter to hold terms composed in parallel. However, rule ($\mathcal{R}0$) uses the union of environments $\cup_{\phi}$ to join all the resulting $\phi_{\mathcal{E}}$ environments. This union is defined over some name x as follows:

$$(\phi_{\mathcal{E}1} \cup_{\phi} \phi_{\mathcal{E}2})(x) = \phi_{\mathcal{E}1}(x) \cup \phi_{\mathcal{E}2}(x)$$

Rule ($\mathcal{E}1$) states that a null process will terminate without altering the $\phi_{\mathcal{E}}$ environment. Rules ($\mathcal{E}2 - \mathcal{E}4$) deal with guarded processes, where $\phi_{\mathcal{E}}$ is modified according to any communications occurring in ($\mathcal{E}3$). Rules ($\mathcal{E}5$) and ($\mathcal{E}6$) again check for any matching and mismatching of names, respectively. Rule ($\mathcal{E}7$) interprets summation as the union of $\phi_{\mathcal{E}}$ environments resulting from the composition of the two processes with ρ. Rule ($\mathcal{E}8$) composes two processes in parallel with ρ. Rule ($\mathcal{E}9$) creates a new name x_i distinct from any previous copies of x. Finally, in ($\mathcal{E}10$), replication is interpreted recursively over ρ using a fixed-point calculation of the union of $\phi_{\mathcal{E}}$ environments.

The correctness of the non-standard semantics with respect to the standard semantics of Fig. 3 is expressed by the following theorem:

Theorem 1 (Correctness of the non-standard semantics)
$\forall P, \rho, \phi_{\mathcal{E}} : \mathcal{S}(\llbracket P \rrbracket)_s\ \rho = \ldots \mathcal{S}(\llbracket P' \rrbracket)_s\ \rho' \ldots \wedge \mathcal{E}(\llbracket P \rrbracket)_s\ \rho\ \phi_{\mathcal{E}} = \ldots \mathcal{E}(\llbracket P' \rrbracket)_s\ \rho'\ \phi'_{\mathcal{E}} \ldots \wedge Q \in \rho \wedge Q[x/y] \in \rho' \Rightarrow x \in \phi'_{\mathcal{E}}(y)$

$$
\begin{array}{lllll}
(\mathcal{E}1) & \mathcal{E}(\![\mathbf{0}]\!)_s\ \rho\ \phi_{\mathcal{E}} & = & \phi_{\mathcal{E}} & \\
(\mathcal{E}2) & \mathcal{E}(\![x(y).P]\!)_s\ \rho\ \phi_{\mathcal{E}} & = & \phi_{\mathcal{E}} & \\
(\mathcal{E}3) & \mathcal{E}(\![\overline{x}\langle y\rangle.P]\!)_s\ \rho\ \phi_{\mathcal{E}} & = & \bigcup_{\substack{\phi \\ x(z).P' \in \rho}} (\mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho[P'[y/z]/x(z).P']]\!)_s\ \phi'_{\mathcal{E}}) & \\
 & & & \text{where } \phi'_{\mathcal{E}} = \phi_{\mathcal{E}}[(\phi_{\mathcal{E}}(z) \cup \{y\})/z] & \\
(\mathcal{E}4) & \mathcal{E}(\![\tau.P]\!)_s\ \rho\ \phi_{\mathcal{E}} & = & \mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)_s\ \phi_{\mathcal{E}} & \\
(\mathcal{E}5) & \mathcal{E}(\![[x=y]P]\!)_s\ \rho\ \phi_{\mathcal{E}} & = & \begin{cases} \mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)_s\ \phi_{\mathcal{E}}, & \text{if } x = y \\ \phi_{\mathcal{E}}, & \text{otherwise} \end{cases} & \\
(\mathcal{E}6) & \mathcal{E}(\![[x\neq y]P]\!)_s\ \rho\ \phi_{\mathcal{E}} & = & \begin{cases} \mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)_s\ \phi_{\mathcal{E}}, & \text{if } x \neq y \\ \phi_{\mathcal{E}}, & \text{otherwise} \end{cases} & \\
(\mathcal{E}7) & \mathcal{E}(\![P+Q]\!)_s\ \rho\ \phi_{\mathcal{E}} & = & (\mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)_s\ \phi_{\mathcal{E}})\ \cup_\phi\ (\mathcal{R}(\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)_s\ \phi_{\mathcal{E}}) & \\
(\mathcal{E}8) & \mathcal{E}(\![P \mid Q]\!)_s\ \rho\ \phi_{\mathcal{E}} & = & \mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)_s\ \phi_{\mathcal{E}} & \\
(\mathcal{E}9) & \mathcal{E}(\![(\nu x)P]\!)_s\ \rho\ \phi_{\mathcal{E}} & = & \mathcal{R}(\![\{\!|P[x_i/x]|\!\}_\rho \uplus_\rho \rho]\!)_{s+\{x_i\}}\ \phi_{\mathcal{E}} & \\
 & & & \text{where, } i = min\{i \mid i \in \mathbb{N} \wedge x_i \notin s\} & \\
(\mathcal{E}10) & \mathcal{E}(\![!P]\!)_s\ \rho\ \phi_{\mathcal{E}} & = & \phi_{\mathcal{E}}\ \cup_\phi\ \mu\phi.\mathcal{E}(\![P]\!)_s\ (\{\!|!P|\!\}_\rho \uplus_\rho \rho)\ \phi & \\
(\mathcal{R}0) & \mathcal{R}(\![\rho]\!)_s\ \phi_{\mathcal{E}} & = & \bigcup_{\substack{\phi \\ P \in \rho}} \mathcal{E}(\![P]\!)_s\ (\rho \backslash \{\!|P|\!\}_\rho)\ \phi_{\mathcal{E}} &
\end{array}
$$

Figure 4: Non-standard semantics of the π-calculus.

Proof: The proof proceeds by induction on the structure of P. The only interesting case is that of rules ($\mathcal{S}3$) and ($\mathcal{E}3$), since these express name substitutions resulting from communications. The restriction rules ($\mathcal{S}9$) and ($\mathcal{E}9$) do not affect $\phi_{\mathcal{E}}$ as we are interested in substitutions resulting from communications alone.□

The correctness requirement states that for any instance of name substitution $Q[x/y]$ resulting from communications in the standard semantics, the same substitution will be captured by the $\phi_{\mathcal{E}}$ environment in the non-standard semantics by adding x to the set $\phi_{\mathcal{E}}(y)$.

5 Abstract Semantics

Despite the fact that the non-standard semantics of the previous section allows for the capturing of name substitution, it operates over an infinite domain D, which may result in the non-termination of the semantics. This is due to the presence of infinite behaviour such as $!((\nu z)\overline{x}\langle z\rangle) \mid\ !x(y)$, which when executed, will cause an infinite number of fresh messages z to be created resulting in an

unlimited growth in s. This will render any fixed-point calculations over the domain D incomputable.

To overcome this problem, we use an approximation that restricts the number of names that can be generated within the semantics. If we assume an integer constraint k, then the number of separate instances of the name z above will be restricted to k instances (i.e. $z_0 \ldots z_{k-1}$). If the value of k is 1, then the performed analysis will be uniform, otherwise, it will be non-uniform. Non-uniform analyses are often useful whenever the property of interest changes with the different runs of the analyzed process. In Section 4, it turns out that a uniform analysis is quite sufficient for the property we are interested in, i.e. security levels of names. Subsequent copies of a name can never change their levels and will always obtain the level of the origin.

In order to enforce the integer constraint k, we need a special function.

Definition 2 *Define the abstraction function,* $\alpha_k : \mathbb{N} \times \mathcal{N} \rightarrow \mathcal{N}$*, as follows:*

$$\alpha_k(x_i) = \begin{cases} x_i, & \text{if } i < k \\ x_{k-1}, & \text{otherwise} \end{cases}$$

Based on α_k, the abstract semantic domain $D^\sharp : \mathcal{N} \rightarrow \wp(\mathcal{N})$ is defined as the total function space of the abstract $\phi_\mathcal{A}$ environments ordered by set inclusion as before. The abstract interpretation of a process P in the π-calculus is now given as an element $\mathcal{A}[\![P]\!]_s\ \rho\ \phi_\mathcal{A} \in D^\sharp$ expressed by the set of rules of Fig. 5.

The rules are very similar to the non-standard semantics of Fig. 4 except for one difference in the interpretation of restriction in rule ($\mathcal{A}9$). Instead of creating an unbounded number of fresh copies of the name x each time ($\mathcal{A}9$) is applied, we restrict the number of these copies to k by applying $\alpha_k(x_i)$. This abstraction ensures that the set $\phi_\mathcal{A}(x)$ will have a finite number of elements throughout the semantics. It also implies that the fixed-point calculation of ($\mathcal{A}10$) is computable since $D^\sharp$ is kept finite and the relation $\mathcal{A}[\![P]\!]_s\ \rho\ \phi_\mathcal{A}$ is a continuous function. This will guarantee the termination of the analysis. Furthermore, the safety of our abstraction with reference to the non-standard semantics of the previous section is given by the following theorem.

Theorem 3 (Safety of the analysis)
$\forall P \in \mathcal{P}, \phi_\mathcal{E} \in D, \phi_\mathcal{A} \in D^\sharp : \mathcal{E}[\![P]\!]_s\ \rho_0\ \phi_{\mathcal{E}0} = \phi_\mathcal{E} \wedge \mathcal{A}[\![P]\!]_s\ \rho_0\ \phi_{\mathcal{A}0} = \phi_\mathcal{A} \Rightarrow \forall x \in s, y \in \phi_\mathcal{E}(x) : \alpha_k(y) \in \phi_\mathcal{A}(\alpha_k(x))$

Proof: The proof is straightforward by induction on the structure of P.□

6 Analysis of Closed Systems

So far, we have dealt with standard π-calculus processes without any security considerations. In this section, we demonstrate how the abstract semantics of the previous section can be applied in detecting two types of privacy breaches: *information leakage* and *insecure communications*. We shall consider here the

$$
\begin{array}{lllll}
(\mathcal{A}1) & \mathcal{A}([\![\mathbf{0}]\!])_s\ \rho\ \phi_{\mathcal{A}} & = & \phi_{\mathcal{A}} \\
(\mathcal{A}2) & \mathcal{A}([\![x(y).P]\!])_s\ \rho\ \phi_{\mathcal{A}} & = & \phi_{\mathcal{A}} \\
(\mathcal{A}3) & \mathcal{A}([\![\overline{x}\langle y\rangle.P]\!])_s\ \rho\ \phi_{\mathcal{A}} & = & \bigcup_{\substack{\phi \\ x(z).P' \in \rho}} (\mathcal{R}([\![\{\!|P|\!\}_\rho \uplus_\rho \rho[P'[y/z]/x(z).P']]\!])_s\ \phi'_{\mathcal{A}}) \\
 & & & \text{where } \phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[(\phi_{\mathcal{A}}(z) \cup \{y\})/z] \\
(\mathcal{A}4) & \mathcal{A}([\![\tau.P]\!])_s\ \rho\ \phi_{\mathcal{A}} & = & \mathcal{R}([\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!])_s\ \phi_{\mathcal{A}} \\
(\mathcal{A}5) & \mathcal{A}([\![[x=y]P]\!])_s\ \rho\ \phi_{\mathcal{A}} & = & \begin{cases} \mathcal{R}([\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!])_s\ \phi_{\mathcal{A}}, & \text{if } x = y \\ \phi_{\mathcal{A}}, & \text{otherwise} \end{cases} \\
(\mathcal{A}6) & \mathcal{A}([\![[x\neq y]P]\!])_s\ \rho\ \phi_{\mathcal{A}} & = & \begin{cases} \mathcal{R}([\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!])_s\ \phi_{\mathcal{A}}, & \text{if } x \neq y \\ \phi_{\mathcal{A}}, & \text{otherwise} \end{cases} \\
(\mathcal{A}7) & \mathcal{A}([\![P+Q]\!])_s\ \rho\ \phi_{\mathcal{A}} & = & (\mathcal{R}([\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!])_s\ \phi_{\mathcal{A}})\ \cup_\phi\ (\mathcal{R}([\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!])_s\ \phi_{\mathcal{A}}) \\
(\mathcal{A}8) & \mathcal{A}([\![P \mid Q]\!])_s\ \rho\ \phi_{\mathcal{A}} & = & \mathcal{R}([\![\{\!|P|\!\}_\rho \uplus_\rho \{\!|Q|\!\}_\rho \uplus_\rho \rho]\!])_s\ \phi_{\mathcal{A}} \\
(\mathcal{A}9) & \mathcal{A}([\![(\nu x)P]\!])_s\ \rho\ \phi_{\mathcal{A}} & = & \mathcal{R}([\![\{\!|P[\alpha_k(x_i)/x]|\!\}_\rho \uplus_\rho \rho]\!])_{s+\{\alpha_k(x_i)\}}\ \phi_{\mathcal{A}} \\
 & & & \text{where, } i = min\{i \mid i \in \mathbb{N} \wedge x_i \notin s\} \\
(\mathcal{A}10) & \mathcal{A}([\![!P]\!])_s\ \rho\ \phi_{\mathcal{A}} & = & \phi_{\mathcal{A}}\ \cup_\phi\ \mu\phi.\mathcal{A}([\![P]\!])_s\ (\{\!|!P|\!\}_\rho \uplus_\rho \rho)\ \phi \\
(\mathcal{R}0) & \mathcal{R}([\![\rho]\!])_s\ \phi_{\mathcal{A}} & = & \bigcup_{\substack{\phi \\ P \in \rho}} \mathcal{A}([\![P]\!])_s\ (\rho \backslash \{\!|P|\!\}_\rho)\ \phi_{\mathcal{A}}
\end{array}
$$

Figure 5: Abstract semantics of the π-calculus.

case of closed systems, where no intruder is assumed to be running in parallel with the system. This fact may be expressed by setting $\rho_0 = \emptyset$.

First, we assume a finite lattice $L = (S_L, \sqsubseteq, \sqcap, \sqcup, \top, \bot)$ of security levels ranged over by $l, l', \ldots$ with $\top_L = l_{top}$ and $\bot_L$ being the undefined level. Using L, we can define a security policy that localizes names depending on the level of the process to which those names are bound and provides a predetermined level for free names.

Definition 4 *Assuming a process P has a level l, define a security policy, $\zeta : \mathcal{N} \to L$, as follows:*

$$
\forall x \in \mathcal{N} : \zeta(x) = \begin{cases} l_x, & \textit{if } x \in fn(P) \\ l, & \textit{if } x \in bn(P) \\ \bot_L, & \textit{otherwise} \end{cases}
$$

Where l_x is the level of free names and is dependant on the administrators' policy. The uniformity of our analysis follows from the fact that names cannot change their values in ζ and the different copies of the same name will always

obtain the same value. This is mainly due to the manner that ζ assigns levels to names: based on the levels of the processes in which those names are created (hence creating a sense of location).

Using ζ and the results of the abstract semantics, we can now analyze for the following two properties, assuming Q has level l and is a subprocess of the system *Sys* (note the strict order on levels):

Property 5 (Information Leakage) $\exists x \in n(Sys), x' \in \phi_{\mathcal{A}}(x), y \in bn(Q)$ *then the leakage of* x *to* Q *is expressed by the result that:* $\zeta(y) \sqsubset \zeta(x') \wedge x \in \phi_{\mathcal{A}}(y)$*, where* $\mathcal{A}[\![Sys]\!]_s\ \rho_0\ \phi_{\mathcal{A}0} = \phi_{\mathcal{A}}$.

Property 6 (Insecure communications) $\exists x \in n(Sys), x' \in \phi_{\mathcal{A}}(x), z(y) \in act(Sys), z' \in \phi_{\mathcal{A}}(z)$ *then the insecure communication of* x *over* z *is expressed by the result that:* $\zeta(z') \sqsubset \zeta(x') \wedge x \in \phi_{\mathcal{A}}(y)$*, where* $\mathcal{A}[\![Sys]\!]_s\ \rho_0\ \phi_{\mathcal{A}0} = \phi_{\mathcal{A}}$.

Consider the following example, which consists of a *Proxy* and a *System*, both of which are running at a security level l_{low}. The *System* consists of a *Server*, a database *DB*, and a back-processing program *Prog*. The *Server* is also running at level l_{low} and is designed to receive local data k (e.g. the client's credit card details) from the *Proxy*. The *Server* then writes k to the database *DB*, which is assumed to be running at level l_{high}. *DB* may be used by *Prog*, which is running at level l_{mid}, to perform some calculations on the database. The overall specification of the system would be as follows:

$$!Proxy \mid\ !System$$

where,
$Proxy \stackrel{\texttt{def}}{=} (\nu k)\overline{port_1}\langle k\rangle$
$System \stackrel{\texttt{def}}{=} (\nu\ db)(\nu\ port_2)(Server \mid DB \mid Prog)$
$Server \stackrel{\texttt{def}}{=} port_1(y).\overline{port_2}\langle y\rangle$
$DB \stackrel{\texttt{def}}{=} port_2(w).\overline{db}\langle w\rangle$
$Prog \stackrel{\texttt{def}}{=} db(u).Prog'$
$\zeta(port_1) = l_{bot}$
$\zeta(k) = \zeta(port_2) = \zeta(db) = l_{low}$.

The assumption about the order of levels is $l_{bot} \sqsubset l_{low} \sqsubset l_{mid} \sqsubset l_{high}$, where l_{bot} is the safest level that can be assigned to the external context. The result of the analysis will be the following:

$\phi_{\mathcal{A}}(k) = \emptyset$
$\phi_{\mathcal{A}}(port_1) = \emptyset$
$\phi_{\mathcal{A}}(port_2) = \emptyset$
$\phi_{\mathcal{A}}(db) = \emptyset$
$\phi_{\mathcal{A}}(y) = \{k\}$
$\phi_{\mathcal{A}}(w) = \{k\}$
$\phi_{\mathcal{A}}(u) = \{k\}$

This example represents an interesting security scenario. Although the communication between *DB* and *Prog* may seem as a leakage since we have $k \in \phi_{\mathcal{A}}(u)$ and $l_{mid} \sqsubset l_{high}$, our analysis does not signal any information leakage. This is because $\zeta(k) \sqsubset \zeta(u)$, which does not satisfy Property 5. On the other hand, the analysis indicates that k has been insecurely communicated over $port_1$ as $k \in \phi_{\mathcal{A}}(y)$ and $\zeta(port_1) \sqsubset \zeta(k)$, which satisfy Property 6. Again, the uniformity of the analysis is demonstrated by the fact that the different runs of the proxy will always produce names $k_1, k_2, \ldots$ all of which share the same level l_{low}.

Now, if we swap the security levels of *Proxy* and *Prog* so that *Proxy* has level l_{mid} and *Prog* has level l_{low}, then clearly Property 5 is satisfied as $\zeta(u) \sqsubset \zeta(k)$.

7 Analysis of Open Systems

Open systems refer to systems that are expected to run within a wider context whose specification may be unknown and hence, could pose security threats. Such a context is also referred to sometimes as the *intruder*. A famous example is the Internet.

Often, to cater for the absence of the intruder's code, the abstract semantics has to be modified to include scenarios where communications between the system and the intruder are possible [14]. However, in our analysis, we shall assume the existence of the intruder's specification, I, and hence no special encoding in the semantics is required. For example, consider the following intruder I of a process P:

$$\begin{array}{rcl} I & \stackrel{\texttt{def}}{=} & (\nu i)[!i(u).((\nu z)\overline{u}\langle z\rangle.\bar{i}\langle z\rangle \mid \overline{u}\langle k\rangle \mid u(w).\bar{i}\langle w\rangle) \quad | \quad !\bar{i}\langle n\rangle] \quad | \\ & & (\nu i)[!i(u).((\nu z)\overline{u}\langle z\rangle.\bar{i}\langle z\rangle \mid \overline{u}\langle k\rangle \mid u(w).\bar{i}\langle w\rangle) \quad | \quad !\bar{i}\langle m\rangle] \quad | \\ & & \vdots \end{array}$$

Where the public names $n, m, k, \ldots \notin \mathcal{N} \cap bn(P)$. This specification is capable of performing any input/output actions initially over public names and then over any new names created by the intruder and extruded beyond its scope and names that will be learnt from P. Such a specification will be contained directly in the ρ parameter as the initial value, i.e. $\rho_0 = \{I\}$.

If we consider the example of the previous section, where *Proxy*, *Server*, *DB*, and *Prog* are all assumed to be running at the same level l_{mid}, and we further assume that I has level l_{bot}. Clearly, we would obtain a result where k is both insecurely communicated and leaked to I, since I has the initial knowledge of all the public names, as for example, $n = port_1$ and $k \in \phi_{\mathcal{A}}(w)$.

8 Conclusion and Future Work

We presented an abstract interpretation-based static analysis for detecting instances of high-level information leakage and insecure communications in the π-calculus. The analysis demonstrates how multilevel process classification in a

denotational setting could be useful in distinguishing between the two properties for closed and open systems. A ML-based prototype of the analysis is already under implementation.

The use of the denotational semantics could be quite interesting in exploring other, more complex, concepts of privacy. For example, a formalization of the notion of weak bisimulation in Stark's semantics would be helpful in building a static analysis for non-interference. Also, a similar model could be adopted for other formalisms, specially the spi calculus [1] and the Mobile Ambients of [8]. The spi calculus is an extension of the π-calculus with cryptographic primitives and its model would benefit the analysis of cryptographic-based properties, like the leakage of keys, and the authenticity and integrity of the communicated data. The Mobile Ambients, on the other hand, explicitly incorporates the notion of location in its modeling of the mobility of code. This would be useful in the analysis of location-based systems, like firewalls.

References

[1] Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: The spi calculus. Fourth ACM Conference on Computer and Communications Security, Zurich, Switzerland (1997) 36–47

[2] Aziz, B., Hamilton, G.W.: A Denotational Semantics for the π-Calculus. Fifth Irish Workshop in Formal Methods (IWFM'01), Trinity College, Dublin, Ireland (2001) (to appear)

[3] Bell, D.E., La Padula, L.: Secure Computer Systems: Mathematical Foundations and Model. MITRE Corp. **M74-244** (1973)

[4] Bergstra, J., Klop, J.: Process Algebra for Synchronous Communication. Information and Control **60(1\3)** (1982) 109–137

[5] Bodei, C., Dagano, P., Nielson, F., Nielson, H.R.: Control Flow Analysis for the π-calculus. Ninth International Conference on Concurrency Theory (ConCur'98), Nice, France. Springer-Verlag, LNCS **1466** (1998) 84–98

[6] Bodei, C., Dagano, P., Nielson, F., Nielson, H.R.: Static Analysis of Processes for No Read-Up and No Write-Down. Foundations of Software Science and Computation Structures (FoSSaCS'99), Amsterdam, The Netherlands. Springer-Verlag, LNCS **1578** (1999) 120–134

[7] Bodei, C., Dagano, P., Nielson, F., Nielson, H.R.: Static Analysis for the π-calculus with Applications to Security. Information and Computation. (to appear)

[8] Cardelli, L., Gordon, A.D.: Mobile Ambients. Foundations of Software Science and Computational Structures. Springer-Verlag, LNCS **1378** (1998) 140–155

[9] Cardelli, L., Ghelli, G., Gordon, A.D.: Secrecy and group creation. Eleventh International Conference on Concurrency Theory (CONCUR00), University Park, PA, USA. Springer-Verlag, LNCS **1877** (2000) 365–379

[10] Cattani, G.L., Stark, I., Winskel, G.: Presheaf Models for the π-Calculus. Category Theory and Computer Science: Proceedings Seventh International Conference (CTCS'97), Santa Margherita Ligure, Italy. Springer-Verlag, LNCS **1290** (1997) 106–126

[11] Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. Proceedings Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA (1977) 238–252

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

[13] Denning, D.: Certification of programs for secure information flow. Communications of the ACM **20** (1977) 504–513

[14] Feret, J.: Confidentiality analysis of mobile systems. Seventh International Static Analysis Symposium (SAS'00), University of California, Santa Barbara, USA. Springer-Verlag, LNCS **1824** (2000) 135–154

[15] Feret, J.: Occurrence counting analysis for the pi-calculus. Proceedings Workshop on Geometry and Topology in Concurrency Theory, Pennsylvania State, USA. Elsevier, ENTCS **39(2)** (2001)

[16] Fiore, M.P., Moggi, E., Sangiorgi, D.: A Fully-Abstract Model for the π-calculus (extended abstract). Eleventh Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA. IEEE Computer Society Press (1996) 43–54

[17] Goguen, J.A., Meseguer, J.: Security Policy and Security Models. IEEE Symposium on Security and Privacy, Oakland, CA, USA. IEEE Computer Society Press (1982) 11–20

[18] Hamilton, G.W., Aziz, B., Gray, D., Power, J., Sinclair, D.: Excommunication: Transforming Pi-Calculus Specifications to Remove Internal Communication. Working paper **CA-0801**, School of Computer Applications, Dublin City University.
`http://www.compapp.dcu.ie/CA_Working_Papers/wp01.html`#0801

[19] Hartonas, C.: Denotational Semantics for a Higher-Order Extension of the Monadic π-Calculus. Report Math&CS1999-1, Technological Education Institute (TEI) of Larissa, Greece (1999)

[20] Hennessy, M.: A Fully Abstract Denotational Semantics for the π-Calculus. Theoretical Computer Science (To appear)

[21] Hennessy, M., Riely, J.: Information Flow vs. Resource Access in the Asynchronous Pi-Calculus. International Colloquium on Automata, Languages and Programming, Geneva, Switzerland. Springer-Verlag, LNCS **1853** (2000) 415–427

[22] Hennessy, M., Riely, J.: Resource Access Control in Systems of Mobile Agents. Third International Workshop on High-Level Concurrent Languages (HLCL'98), Nice, France. Elsevier, ENTCS **16(3)** (1998) 1–15

[23] Hepburn, M., Wright, D.: Trust in the Pi-Calculus. Third Conference on Principles and Practice of Declarative Programming (PPDP01), Florence, Italy (2001) (to appear)

[24] Honda, K., Vasconcelos, V., Yoshida, N.: Secure information flow as typed process behavior. European Symposium on Programming (ESOP 2000), Berlin, Germany. Springer-Verlag, LNCS **1782** (2000) 180–199

[25] Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, (Parts I and II). Information and Computation **100** (1992) 1–77

[26] Pitts, A.M.: A co-induction principle for recursively defined domains. Theoretical Computer Science **124** (1994) 195–219.

[27] Stark, I.: A Fully Abstract Domain Model for the π-Calculus. Eleventh Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA. IEEE Computer Society Press (1996) 36-42

[28] Stark, I.: Outline of a Denotational Semantics for the Pi-Calculus. Electronic Note.
`http://www.dcs.ed.ac.uk/home/stark/publications/fuladm.html`

[29] Venet, A.: Abstract Interpretation of the π-calculus. Fifth LOMAPS Workshop on Analysis and Verification of Multiple-Agent Languages, Stockholm, Sweden. Springer-Verlag, LNCS **1192** (1997) 51–75