A Denotational Semantics for the π-Calculus

B. Aziz G.W. Hamilton

School of Computer Applications
Dublin City University
Dublin 9, Ireland
{*baziz,hamilton*}*@compapp.dcu.ie*

Abstract

In his categorical framework, Stark defines a domain-theoretic model for the π-calculus based on functor categories. Despite being a sound abstract model, a more concrete semantics is required if it is to be used as a basis for proving properties about mobile systems. In this paper, we concretize Stark's denotational model for the π-calculus and provide a full definition of the semantic domains involved. We also include an example of how the model may be approximated in an abstract interpretation analysis.

1 Introduction

Traditionally, the meaning of a process in the π-calculus [15] has been specified by means of a Structural Operational Semantics (SOS) [17], where processes are transformed into a suitable form for communication and then interactions are controlled by a Labeled Transition System (LTS). However, building program static analyses for large (infinite) applications becomes a hard task to perform due to the fact that the SOS is a small-step semantics, making it hard to arrive at reasonable approximations for such semantics [18].

Recently, several models based on the denotational approach have been suggested as alternatives to the SOS-based models [4, 6, 10, 12, 13, 19]. In this approach, the meaning of a process is usually given as an element in a semantic domain using some semantic function that maintains properties of the translated process. This meaning may then be easily enhanced and approximated leading to some simplified finite abstractions over which automated tools for reasoning about programs and proving their correctness may be built.

In this paper, we substantiate Stark's categorical framework [19] by providing a fully defined denotational semantics for the π-calculus. In his framework, Stark presents a domain-theoretic model of name-passing processes that is based on functor categories. The model is shown to be fully abstract and capable of capturing the operational notions of transitions, strong late bisimilarity, and strong equivalence in the π-calculus. Being of categorical nature, it also provides a syntax-free handling of names, visibility, privacy, and interference.

We give a concrete definition of the morphism of restriction, which interprets the effects of restricted names like deadlocks and bound output actions. The full denotational semantics of the π-calculus is then presented along with a solution to the problem of tracing the origins of names. In [19], the origin of a name is lost as a result of applying α-conversion. Experience has shown that this information is essential in a program analysis that involves determining whether confidential data can be leaked or insecurely communicated [2].

The rest of the paper is structured as follows. In Section 2, we review related work in the area of denotational semantics of the π-calculus. In Section 3, we give an overview of the syntax and the structural operational semantics of the standard language. The categorical and concrete definitions of the semantic domains for the π-calculus are given in Section 4, which are then used in defining the denotational semantics of the π-calculus in Section 5. In Section 6, we give one example of abstractions that may use the semantics of Section 5. Finally, we conclude the work in Section 7 and give directions for future work.

2 Related Work

Despite the fact that most of the work in the area of the π-calculus has focused on the SOS of the language, some research has been carried out to define other semantic frameworks for the language [4, 6, 10, 12, 13]. Most of these have adopted abstract categorical models with little or no concrete definitions.

The model presented in [10] is very close to that of [19]. The major difference is that [10] defines the denotational semantics in terms of a meta language, extracted from the categorical model. Therefore, it is a more abstract framework that relies on a uniform approach capable of giving semantics to a variety of calculi. However, like [19], the work of [10] only handles strong late bisimulation and lacks definitions for other forms of equivalence, like early and open bisimulation as well as weak bisimulation, which partially ignores internal actions. In [13], two set-theoretic denotational models are presented that are fully abstract with respect to *must* and *may* testing. The models are obtained as solutions to domain equations in a functor category.

In [12], a denotational semantics of a higher order version of the π-calculus is presented, where not only ground type messages like channels can be exchanged, but also processes. This approach is interesting since it comes closer to the meaning of code mobility. The model is also based on a solution to a single domain equation in a functor category and is shown to be computationally adequate. A flexible model for the π-calculus based on presheaf categories is presented in [6]. Advantages of this approach include support for function spaces and other traditional models like synchronization, trees, and event structures, as well as a better understanding of bisimulation in the π-calculus. This is because presheaf categories come along with the concept of drawing bisimulation from open maps [14], which yields an easier treatment of bisimulation. Finally, [4] defines a Petri net semantics of the π-calculus based on P/T nets with inhibitor arcs and is proven to be sound with respect to early transition semantics.

3 The π-Calculus

The π-calculus [15] is a compact notation used in specifying systems of mobile and distributed processes. The notion of a *name* is central to the calculus, where names may stand for communication channels as well as passive data. Mobility of processes is modeled as the movement of links in the virtual world of linked processes. This movement is achieved by allowing processes to send and receive names (including channel names) over communication channels thereby allowing communication topologies to change throughout the lifetime of a process.

3.1 Syntax

We write the countably infinite set of names as $\mathcal{N} = \{x, y, z, \ldots\}$ possibly subscripted with integers and the set of processes as $\mathcal{P} = \{P, Q, R, \ldots\}$. The full syntax of the standard monadic version of the π-calculus is defined in Figure 1.

P	:=	$\mathbf{0}$	Null Process
		$\pi.P$	Guarded Process
		$P + Q$	Summation
		$P \mid Q$	Parallel Composition
		$(\nu x)P$	Restriction
		$!P$	Replication
		$[x = y]P$	Match
		$[x \neq y]P$	Mismatch
π	:=	$x(y)$	Input Prefix
		$\overline{x}\langle y \rangle$	Output Prefix
		τ	Silent Prefix

Figure 1: Syntax of the π-calculus

A null process $\mathbf{0}$ is an inactive process that is unable to perform any actions. A guarded process $\pi.P$ performs an atomic action π and continues as P. An atomic action can be an input, output, or silent action. An input action $x(y).P$ substitutes a message z received over x for each free occurrence of y in P and continues as $P[z/y]$. An output action $\overline{x}\langle y \rangle.P$ sends y over x and continues as P. A silent action $\tau.P$ is not observable; the process reacts internally and continues as P. A summation $P + Q$ chooses one between P and Q and refuses the other. Parallel composition $P \mid Q$ interleaves P and Q with the possibility of communications between them. Restriction $(\nu x)P$ creates a fresh name x within the scope of P. Infinite behaviour may be expressed by the replication operator $!P$ which creates as many parallel copies of P as desired. Finally, matching $[x = y]P$ compares x and y and continues as P if they are the same else it blocks. Mismatching $[x \neq y]P$ is the dual of matching.

The name y is *bound* to P in $x(y).P$ and $(\nu y)P$, otherwise it is *free*. The set $bn(P)$ is the set of bound names and $fn(P)$ the set of free names of P. Moreover, the set of all names of a process P is written as $n(P) = bn(P) \cup fn(P)$.

3.2 Structural Operational Semantics

The original semantics of the π-calculus introduced in [15] was a SOS inspired by Berry and Boudol's chemical abstract machine [3]. This semantics relies on two main relations: the *structural congruence* and the *reduction* relations. The structural congruence relation $\equiv$ is defined as the least congruence satisfying the following rules:

- Processes are identified up to the renaming of bound names (α-conversion).
- $(\mathcal{P}/\equiv, +, \mathbf{0})$ and $(\mathcal{P}/\equiv, |, \mathbf{0})$ are commutative monoids.
- $(\nu x)\mathbf{0} \equiv \mathbf{0}$
- $(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$
- $!P \equiv (!P \mid P)$
- $(\nu x)(P \mid Q) \equiv P \mid (\nu x)Q, \quad \text{if } x \notin fn(P)$

The reaction relation $\longrightarrow$ is defined by the transition system of Figure 2.

$$\text{(Comm): } (\overline{x}\langle z\rangle.P) \mid (x(y).Q) \longrightarrow (P \mid Q[z/y]) \qquad \text{(Res): } \frac{P \longrightarrow P'}{(\nu x)P \longrightarrow (\nu x)P'}$$

$$\text{(Par): } \frac{P \longrightarrow P'}{(P \mid Q) \longrightarrow (P' \mid Q)} \qquad \text{(StruCong): } \frac{P \equiv P' \longrightarrow Q' \equiv Q}{P \longrightarrow Q}$$

Figure 2: The rules for the transition relation $\longrightarrow$

The axiom (Comm) defines communications in the π-calculus. A process willing to send a message over some channel can communicate with another process running in parallel and willing to receive a message over the same channel. Rules (Par) and (Res) allow a process to evolve under the parallel composition and restriction contexts respectively. Finally, the structural congruence rule (StruCong) is included to allow processes to evolve according to the structural congruence rules.

4 A Categorical Framework

In this section, we review Stark's categorical model [19] and provide concrete definitions of the morphisms representing process behaviour. As it turns out, this is only necessary for the restriction morphism.

4.1 Abstract Model

The abstract model is mainly motivated by the solution to the following predomain equations, which define a π-calculus process in terms of the different 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 and $\multimap$ is a non-standard exponential.

The solution to equations (1–3) above is constructed within a functor category C that is symmetric monoidal closed [19]. In order to construct C, an initial category B^I is proposed, where B is the category of bifinite predomains and continuous maps described in [16] 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. 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). Finally, the power operation $P(-)$ is taken as the adaptation of Plotkin's powerdomain $P^\natural$ to bifinite predomains [16] and the non-standard operation $\multimap$ as a lifted function space that supplies a fresh name to a process.

Within C, a number of morphisms leading into $Pi_\perp$ are defined as in Figure 3. These morphisms describe the manner in which processes are constructed

$\emptyset : 1 \to Pi_\perp$	(Null)
$\{\!\|-\|\!\} : (Pi_\perp + In + Out)_\perp \to Pi_\perp$	(Singleton)
$\uplus : Pi_\perp \times Pi_\perp \to Pi_\perp$	(Choice)
$new : (N \multimap Pi_\perp) \to Pi_\perp$	(Restriction)

Figure 3: Morphisms in C leading into the $Pi_\perp$ object

in the π-calculus. The (Null) morphism denotes a terminated or deadlocked process. The (Singleton) map is used to interpret input, free/bound output, and silent actions where $\{\!|\perp|\!\}$ is the least element representing the undefined process and $\{\!|\perp|\!\} \sqsubseteq \emptyset$. Otherwise, $\emptyset$ is incomparable. The (Choice) morphism is the standard powerdomain union representing non-deterministic choice between two processes. Finally, the (Restriction) morphism is used to interpret the effects of restriction like deadlocks and bound outputs.

4.2 Concrete Definitions

The manner by which elements $p, q \ldots$ of the concrete semantic domain $Pi_{\perp}s$ of processes are constructed follows two steps: first, the primitive elements resulting from the $\emptyset$, $\{\!|-|\!\}$, and $\uplus$ morphisms are directly joined to $Pi_{\perp}s$ without requiring further interpretation. Name binding is expressed as a λ-abstraction of the form $\lambda x.p$, where x is bound to p. Hence, an input action $x(y).P$ is represented as $in(x, \lambda y.p)$. Free output and silent actions are represented as $out(x, y, p)$ and $tau(p)$, respectively. In [19], an additional $\lambda\underline{x}.p$ operator was used to express bound output actions like $out(x, \lambda\underline{y}.p)$, where y is considered to be fresh (i.e. $y \notin s$) and it is extruded beyond the scope of p. However, this operator is superfluous in our semantics, as we use a more disciplined mechanism for generating new names, which ensures that any newly created names are always fresh (distinct).

The second step is to interpret the effects of restricting elements p, q using the *new* morphism. This requires a concrete definition of the *new* morphism over a finite set of names s as illustrated by the rules of Figure 4 (similar to the definition given in [20]). The intuition behind these rules is that deadlocked situations may arise when an attempt is made to communicate over a restricted channel. Also, fresh names sent immediately over a channel will turn a free output into a bound output extruding the fresh message beyond the scope of the residue. Again, note the usage of the normal binding operator $\lambda x.p$, as names are guaranteed to be fresh. In all the other contexts, restriction has no effect or it is distributed over the process.

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

Figure 4: The definition of the *new* morphism over s

5 Denotational Semantics

Using the definitions of the previous section, we can now give a fully defined denotational semantics for the standard monadic version of the π-calculus using the semantic relation $\mathcal{C}(\![P]\!)_s \in Pi_\perp s$ defined inductively over P with free names in s as shown in Figure 5. Within these rules, the parameter ρ is a multiset environment of all the processes that are composed in parallel with P. We shall use the singleton map $\{\!|-|\!\}_\rho$ and the multiset union $\uplus_\rho$ operators over ρ, which should not be confused with the morphisms $\{\!|-|\!\}$ and $\uplus$ used to combine elements of $Pi_\perp s$. We shall also employ a separate rule $(\mathcal{R})$, which interprets the meaning of the composition of all the processes in ρ using the $\mathcal{C}(\![P]\!)_s$ relation.

$$
\begin{array}{lll}
(\mathcal{C}1) & \mathcal{C}(\![\mathbf{0}]\!)_s\ \rho & = \emptyset \\
(\mathcal{C}2) & \mathcal{C}(\![x(y).P]\!)_s\ \rho & = \{\!|in(x, \lambda y.(\mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)_{s\cup\{y\}}))|\!\} \\
(\mathcal{C}3) & \mathcal{C}(\![\overline{x}\langle y\rangle.P]\!)_s\ \rho & = \biguplus_{x(z).P'\in\rho} \mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho[P'[y/z]/x(z).P']]\!)_s) \uplus \\
 & & \quad \{\!|out(x, y, \mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)_s)|\!\} \\
(\mathcal{C}4) & \mathcal{C}(\![\tau.P]\!)_s\ \rho & = \{\!|tau(\mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)_s)|\!\} \\
(\mathcal{C}5) & \mathcal{C}(\![[x = y]P]\!)_s\ \rho & = \begin{cases} \mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)_s, & \text{if } x = y \\ \emptyset, & \text{otherwise} \end{cases} \\
(\mathcal{C}6) & \mathcal{C}(\![[x \neq y]P]\!)_s\ \rho & = \begin{cases} \mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)_s, & \text{if } x \neq y \\ \emptyset, & \text{otherwise} \end{cases} \\
(\mathcal{C}7) & \mathcal{C}(\![P + Q]\!)_s\ \rho & = (\mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)_s)\ \uplus\ (\mathcal{R}(\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)_s) \\
(\mathcal{C}8) & \mathcal{C}(\![P \mid Q]\!)_s\ \rho & = \mathcal{R}(\![\{\!|P|\!\}_\rho \uplus_\rho \{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)_s \\
(\mathcal{C}9) & \mathcal{C}(\![(\nu x)P]\!)_s\ \rho & = new_s(\lambda x_i.\mathcal{R}(\![\{\!|P[x_i/x]|\!\}_\rho \uplus_\rho \rho]\!)_{s\cup\{x_i\}}) \\
 & & \quad \text{where, } i = min\{i \mid i \in \mathbb{N} \wedge x_i \notin s\} \\
(\mathcal{C}10) & \mathcal{C}(\![!P]\!)_s\ \rho & = \mathcal{C}(\![P]\!)_s\ (\{\!|!P|\!\}_\rho \uplus_\rho \rho) \\
 & & \\
(\mathcal{R}) & \mathcal{R}(\![\rho]\!)_s & = \biguplus_{P\in\rho} \mathcal{C}(\![P]\!)_s\ (\rho \backslash \{\!|P|\!\}_\rho)
\end{array}
$$

Figure 5: Denotational Semantics of the π-Calculus

Rule $(\mathcal{C}1)$ interprets the meaning of a null process directly as the $\emptyset$ morphism.

Rules $(\mathcal{C}2) - (\mathcal{C}4)$ deal with the cases of processes guarded by input, output, and silent actions. In $(\mathcal{C}2)$, the input parameter y is joined to s as it is free (though not necessarily fresh) within the residue. In $(\mathcal{C}3)$, 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. In $(\mathcal{C}4)$, a silent action is interpreted and the residue is composed with the processes in ρ. The residues of those processes are modified accordingly. Rules $(\mathcal{C}5)$ and $(\mathcal{C}6)$ match and mismatch two names respectively. If the condition of the rule holds, the residue P is composed with the processes in ρ, else it blocks. Rule $(\mathcal{C}7)$ interprets each of the processes in a summation as composed with the processes in ρ. The alternatives are combined by the $\uplus$ morphism. Rule $(\mathcal{C}8)$ interprets the meaning of two parallel processes as merely composed with the rest of processes in ρ. Rule $(\mathcal{C}9)$ introduces a labeling mechanism into restriction which ensures that the created name is always distinct from previous copies. This mechanism successively generates the names $x_1, x_2 \ldots$ each time the restriction operator (νx) is applied. This removes the need for α-conversion and allows the new copies to be traced back to their origin. Finally, in $(\mathcal{C}10)$, replication is interpreted recursively using ρ.

Similar to [19], this model can be shown to capture the notions of strong late bisimilarity and strong equivalence by showing that the semantics both reflects and preserves transitions in the structural operational semantics.

6 An Abstraction Example

One area of research where the semantics of Figure 5 can be found useful is the area of abstract interpretation analysis [7, 8], where safe properties of the analyzed programs are extracted based on some abstraction of the concrete semantic domain of the language. This abstraction is naturally directed to assist the aim of the analysis by keeping the (abstract) semantic domain finite. It also may be built on top of a correctly extended semantics enhanced with information relevant to the particular properties of the analysis. An example of such information could be an environment containing the set of names a particular name may be associated with during the execution of a process. This set can then be used to detect privacy breaches in systems with multilevel security policies [9]. Such breaches include instances of high-level data being leaked to low-level processes (information leakage) and situations where high-level data being sent over low-level channels (insecure communications) [2].

For example, consider the set of non-standard semantics of the π-calculus shown in Figure 6, where $\mathcal{E}(\llbracket P \rrbracket)_s \ \rho \ \phi \in (\mathcal{N} \to \wp(\mathcal{N}))$. Here, $\phi : \mathcal{N} \to \wp(\mathcal{N})$ has replaced elements $p, q \in Pi_{\perp}s$ as the non-standard meaning of a process in the π-calculus. The ϕ environment maps a name to the set of names that may replace that name during runtime. However, allowing the ϕ environment as it is may well render it incomputable due to the presence of infinite behaviour like $!((\nu z)\overline{x}\langle z\rangle) \mid !(x(y))$. Therefore, a reasonable abstraction is necessary to keep the value of $\phi(x)$ finite. One such abstraction could be represented by the

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

Figure 6: Non-Standard Semantics of the π-Calculus

following abstract version of rule ($\mathcal{E}9$):

$$(\mathcal{E}9A)\ \mathcal{E}(\llbracket (\nu x)P \rrbracket)_s\ \rho\ \phi = \mathcal{R}(\llbracket \{\!| P[x^\sharp/x] |\!\}_\rho \uplus_\rho \rho \rrbracket)_{s \cup \{x^\sharp\}}\ \phi$$

Where, $x^\sharp = \alpha_k(x_i)$ and $i = min\{i \mid i \in \mathbb{N} \wedge x_i \notin s\}$ for some integer limit k. $\alpha_k(x_i)$ is an abstraction function that allows non-uniform abstraction of names ($\alpha_0(x_i)$ denotes uniform abstraction):

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

With ($\mathcal{E}9A$), the mapping ϕ of names to sets of names is going to operate over the finite domain of names $n(P)$. Hence, a least fixed-point of the semantics is guaranteed to be computable. Clearly, with this abstraction one should still be able to relate the names $x_1, x_2 \ldots$ resulting from ($\mathcal{E}9$) and the single copy x resulting from ($\mathcal{E}9A$) thanks to the labeling mechanism we use in the interpretation of (νx). It is important to mention that such abstractions are mainly justified by the fact the many copies of the same name share some common property, for example the same security level.

7 Conclusion and Future Directions

In conclusion, we have presented concrete definitions of the abstract denotational framework presented by Stark in his categorical framework [19]. These definitions, we believe, are necessary for any program analysis that relies on the concrete semantics of the π-calculus, like an abstract interpretation-based analysis. The use of a labeling mechanism for names allows for the tracing of their origins. We hope in future work to extend Stark's model to include richer formalisms, like the spi calculus [1] and the mobile ambients [5]. The former is an extension of the π-calculus with cryptographic primitives, whereas the latter embodies a notion of location. Also, it is quite interesting to find out how the model deals with other notions of bisimilarity, like weak and open bisimilarity, which could be quite useful in modeling non-interference properties [11].

References

[1] M. Abadi and A.D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proceedings of the* 4^{th} *ACM Conference on Computer and Communications Security*, Zurich, Switzerland, pages 36–47, 1997.

[2] B. Aziz and G.W. Hamilton. A Static Analysis for Privacy in the π-calculus (Working Paper).

[3] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, vol. 96, pages 217–248, 1992.

[4] N. Busi and R. Gorrieri. A Petri Net Semantics for π-Calculus. In *Proceedings of the* 66^{th} *International Conference on Concurrency Theory (CONCUR'95)*, Philadelphia, PA, U.S.A., pages 145–159, 1995.

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

[6] G.L. Cattani, I. Stark, and G. Winskel. Presheaf Models for the π-Calculus. In *Category Theory and Computer Science: Proceedings of the 7th International Conference CTCS '97*, Santa Margherita Ligure, Italy. LNCS 1290, pages 106–126, Springer-Verlag, 1997.

[7] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the* 4^{th} *ACM Symposium on Principles of Programming Languages*, Los Angeles, California, U.S.A., pages 238–252, 1977.

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

[9] D. Denning. Certification of programs for secure information flow. Communications of the ACM, vol. (20), pages 504–513, 1977.

[10] M.P. Fiore, E. Moggi, and D. Sangiorgi. A Fully-Abstract Model for the π-calculus. In *Proceedings of the* 11^{th} *Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1996.

[11] J.A. Goguen and J. Meseguer. Security Policy and Security Models. IEEE Symposium on Security and Privacy, Oakland, CA, USA. IEEE Computer Society Press, pages 11-20, 1982.

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

[13] M. Hennessy. A Fully Abstract Denotational Semantics for the π-Calculus. To appear in *Theoretical Computer Science*.

[14] A. Joyel, M. Nielsen, and G. Winskel. Bisimulation from open maps. *Information and Computation*, vol. 127(2), pages 164–185, 1996.

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

[16] A.M. Pitts. A co-induction principle for recursively defined domains. *Theoretical Computer Science*, vol. 124, pages 195–219, 1994.

[17] G.D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, Denmark, 1981.

[18] D.A. Schmidt. Abstract Interpretation of Small-Step Semantics. In *Proceedings of the* 5^{th} *LOMAPS Workshop on Analysis and Verification of Multiple-Agent*, Languages, Stockholm, 1996. LNCS 1192, Springer-Verlag, pages 76–99, 1997.

[19] I. Stark. A Fully Abstract Domain Model for the π-Calculus. In *Proceedings of the* 11^{th} *Annual IEEE Symposium on Logic in Computer Science*, pages 36–42. IEEE Computer Society Press, 1996.

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