π^+-Calculus: An Extension of the π-Calculus to handle Constraints *

Frank D. VALENCIA POSSO † Juan Francisco DIAZ FRIAS ‡ Camilo RUEDA §

12th June 1997

Abstract

The π-calculus is a formal model of concurrent computation based on the notion of naming. It has an important role to play in the search for more abstract theories of concurrent and communicating systems. In this paper we extend the π-calculus with constraints by adding the notion of *constraint agent* to the standard π-calculus concept of agent. We call this extension the π^+-calculus. This paper also includes examples and uses of the extended calculus showing the transparent interaction of constraints and communicating processes.

Keywords: Concurrent Programming, Constraint Programming, π-calculus, π^+-calculus, Formal Calculi, Mobile Processes.

1 Introduction

Research on multiparadigm languages has known increasing interest in the last years. The need to establish a firm base for the integration of what appears to be fundamentally different notions of programming has led to the design of formal calculi for a variety of paradigms. One approach in this direction is to devise a calculus for a particular paradigm and then show how to simulate the others in the calculus. The ρ-calculus [NM95], for example, is a formal base for concurrent constraint programming subsuming the λ-calculus and powerful enough to simulate objects (but not classes) and inheritance.

A more direct approach is to include the notions of the integrated paradigms in the calculus, as in TyCO [Vas94], a calculus of typed concurrent objects. We favor this latter approach in our search for a calculus integrating Object-Oriented and Constraint Programming. Our strategy is to rely on minimal orthogonal extensions to calculi that have already found an established place in the programming language community. We have chosen as our point of departure the π-calculus [MPW92, Mil91], a well known elegant and simple model of concurrent computation which also subsumes the λ-calculus, because we believe that concurrency is fundamental to both Object-Oriented and Constraint Programming. In a sense, TyCO can be seen as following this direction, in that it modifies the π-calculus by including the notion of objects instead of channels and performing communication via messages passing.

To our knowledge there has been no attempt to encode first-order constraints into the π-calculus or to orthogonally extend the π-calculus to include them, much less to integrate in it both objects and constraints. An extension called κ-calculus [Smo94b], only considers equational constraints, whereas [VP96] shows that equational constraints can be encoded into the π-calculus.

*This work is supported in part by grant 1251-14-041-95 from Colciencias-BID.

†Associate Professor, Faculty of Engineering, **Pontificia Universidad Javeriana Cali, Colombia.** Member of AVISPA team. E-mail: fvalenci@atlas.ujavcali.edu.co.

‡Associate Professor, Computer Science Department, Faculty of Engineering, **Universidad del Valle, Colombia.** Member of AVISPA team and GEDI team. E-mail: jdiaz@borabora.univalle.edu.co.

§Associate Professor, Faculty of Engineering, **Pontificia Universidad Javeriana Cali, Colombia.** Chief of AVISPA team and member of GEDI team. E-mail: crueda@atlas.ujavcali.edu.co.

In this paper we present the π^+-calculus, an orthogonal extension of the (polyadic) π-calculus with constraints, where the notion of *constraint agent* is added to the standard π-calculus concept of agent.

In sections 2, 3, and 4 we present the syntax, semantics and uses of the π^+-calculus, respectively. The syntax of the π^+-calculus adds constraint agents to the standard π-calculus agents. Constraint agents performs the well known Ask and Tell operations of CCP languages. The semantics is defined operationally following the transition system for the cc-model used in [Sar93], this in contrast with the denotational semantics of the π-calculus. We illustrate some of the more interesting uses of π^+-calculus: the definition of recursive processes and the notion of cell, which can be used to provide a notion of state compatible with concurrency and constraints. We also illustrate examples to show the transparent interaction of constraints and communicating processes.

Finally, section 5 shows conclusions and future work.

2 Syntax

The syntax of the π^+-Calculus is given in Table 1. There are only two kinds of entities in the π^+-calculus: *Channels* and *Agents* (or *Processes*). The π^+-calculus adds *Constraint* agents and agents declaring variables to the standard π-calculus agents. In the π-calculus, *names* denote channels. The π^+-calculus also allows *variables* and *primitive values* to be channels.

Normal Processes: M, N	::=	$\pi.P$	Agent under prefix
	|	$M + N$	Summation
	|	O	Inaction or null process
Constraint agents: R	::=	$!\phi.P$	Tell agent
	|	$?\phi.P$	Ask agent
Agents (or processes) P, Q	::=	$(\upsilon a)P$	New name a in P
	|	$(\upsilon x)P$	New variable x in P
	|	$P \mid Q$	Composition
	|	N	Normal process
	|	$*P$	Replicated agent
	|	R	Constraint agent
Channels C	::=	a	Name
	|	v	Value
	|	x	Variable
Prefixes: π	::=	$C?[x_1 \ldots x_n]$	Reading prefix
	|	$C![C_1 \ldots C_n]$	Writing prefix

Table 1: π^+-calculus syntax

In what follows, we describe the agents informally. In an agent $\pi.P$ the prefix π represents an *atomic action*, the first action performed by $\pi.P$ and P represents the continuation of $\pi.P$. When π is a writing prefix $C![C_1 \ldots C_n]$, $\pi.P$ means "send $C_1, \ldots, C_n$ along channel C and then activate P". When π is a reading prefix $C?[x_1 \ldots x_n]$, $\pi.P$ means "receive the arguments, say $x_1, \ldots, x_n$, along channel C, use them in P and then activate P". In both cases C is called the *subject* of π.

The summation form $M + N$ represents a process able to take part in one -but only one - of two alternatives for communication. The choice of one alternative precludes the other. The null process 0 is the process doing nothing.

Constraint agents are new kind of agents whose behavior depends on a global *store*. A store contains information given by constraints. The Tell agent $!\phi.P$ means "Add ϕ to the store and then activate P." The Ask agent $?\phi.P$ means "Activate P if constraint ϕ is a logical consequence of the information in the store"

The agent $(\upsilon a)P$ restricts the use of the name a to P. Another way to describe this is that $(\upsilon a)P$ declares a new unique name a, distinct from all external names, for use in P. Similarly, $(\upsilon x)P$ (new agent) declares a new variable x, distinct from all external variables in P.

The agent $P \mid Q$ means that P and Q are concurrently active, so they can act independently (and possibly communicate). $*P$ "Bang P" means $P \mid P \ldots$(as many copies as you wish). The operator $*$ is called *replication*. A common instance of replication is $*\pi.P$; a resource which can only be replicated when a requester communicates via π.

Usually, agents of the form $\pi.O$ are written π. We also omit the $.O$ in the constraint agents $!\phi.O$ and $?\phi.O$

In the next section, we describe formally the behavior of agents.

3 Operational Semantics

3.1 Constraint System

The π^+-calculus is parametrized in a Constraint System. A Constraint System consists of [Smo94b, Smo94a]:

- A signature Σ (a set of functions, constants and predicate symbols with equality) including a distinguished infinite set, $\mathcal{N}$, of constants called *names* denoted as $a, b, \ldots, u$. Other constants, called *values*, are written $v_1, v_2, \ldots$.
- A consistent theory Δ (a set of sentences over Σ having a model) satisfying two conditions:
  1. $\Delta \models \neg(a = b)$ for every two distinct names a, b.
  2. $\Delta \models \phi \leftrightarrow \psi$ for every two sentences ϕ, υ over Σ such that ψ can be obtained from ϕ by permutation of names.

Often Δ will be given as the set of all sentences valid in a certain structure (e.g. the structure of finite trees, integers, or rational numbers). Given a constraint system, symbols $\phi, \psi, \ldots$ denote first-order formulae in Σ, henceforth called *constraints*. We say that ϕ *entails* ψ in Δ, written $\phi \models_\Delta \psi$, iff $\phi \rightarrow \psi$ is true in all models of Δ. We say that ϕ is *equivalent* to ψ in Δ, written $\phi \models\!\!\models \psi$, iff $\phi \models_\Delta \psi$ and $\psi \models_\Delta \phi$. We say that ϕ is *satisfiable* in Δ iff $\phi \not\models \bot$. We use $\bot$ for the constraint that is always false and $\top$ for the constraint that is always true.

As usual, we will use infinitely many $x, y, \ldots \in \mathcal{V}$ to denote logical variables, designating some fixed but unknown element in the domain under consideration. The sets $fv(\phi) \subset \mathcal{V}$ and $bv(\phi) \subset \mathcal{V}$ denote the sets of free an bound variables in ϕ, respectively. Finally, $fn(\phi) \subset \mathcal{N}$ is the set of names appearing in ϕ.

As we said before, constraint agents act relative to a store. A store is defined in terms of the underlined constraint system:

Definition 3.1 (Store) *A store $S = \phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_r$ (with $r \geq 0$) is a constraint in Σ. When $r = 0$, S is said to be the empty store (i.e., $S = \top$). When $S \models_\Delta \bot$, S is said to be the unsatisfiable store.*

The operational semantics of the π^+-calculus will be defined in terms of a equivalence relation, $\equiv_{\pi+}$, on configurations describing computation, states and a one-step reduction relation, $\xrightarrow{\pi^+}$ describing transitions on these configurations. A configuration is a tuple $\langle P; S\rangle$ consisting of an agent P and a store S.

3.2 Structural Congruence and equivalence on configurations

We identify first the binding operators in the π^+-calculus: The binding operator for names is $(\upsilon a)P$ which declares a new name a in P. There are two binding operators for variables: $(\upsilon x)P$ which binds x in P and $C?[x_1 \ldots x_n].P$ which declares formal parameters $x_1, \ldots, x_n$ in P. So we can define *free names* $fn(P)$, *bound names* $bn(P)$, *free variables* $fv(P)$, *bound variables* $bn(P)$ of a process P in the usual way. In a similar way as [Mil91] we define structural congruence for the π^+-calculus.

Definition 3.2 (Structural Congruence) *Let structural congruence, $\equiv$, be the smallest congruence relation over agents which satisfies the following axioms:*

- *Agents are identical if they only differ by a change of bound variables or bound names.*
- *$(\mathcal{NP}/\equiv, +, O)$ and $(\mathcal{A}/\equiv, |, O)$ are symmetric monoids, where $\mathcal{NP}$ and $\mathcal{A}$ are the set of normal processes and agents respectively.*
- $*P \equiv P \mid *P$.
- $(\upsilon a)O \equiv O$, $(\upsilon x)O \equiv O$, $(\upsilon a)(\upsilon b)P \equiv (\upsilon b)(\upsilon a)P$, $(\upsilon x)(\upsilon y)P \equiv (\upsilon y)(\upsilon x)P$, $(\upsilon a)(\upsilon x)P \equiv (\upsilon x)(\upsilon a)P$.
- *If $a \notin fn(P)$ then $(\upsilon a)(P \mid Q) \equiv P \mid (\upsilon a)Q$.*
- *If $x \notin fn(P)$ then $(\upsilon x)(P \mid Q) \equiv P \mid (\upsilon x)Q$.*
- *If $\phi \models\!\!\!\models \psi$ and $P \equiv Q$ then $!\phi.P \equiv !\psi.Q$ and $?\phi.P \equiv ?\psi.Q$*

Definition 3.3 (π^+-equivalence relation on configurations) *We will say that $< P_1; S_1 >$ is π^+-equivalent to $< P_2; S_2 >$, written $< P_1; S_1 > \equiv_{\pi+} < P_2; S_2 >$, if $P_1 \equiv P_2$, $S_1 \models\!\!\!\models S_2$, $fn(S_1) = fn(S_2)$ and $fv(S_1) = fv(S_2)$. $\equiv_{\pi+}$ is said to be the π^+-equivalence relation on configurations.*

The behavior of an agent P is defined by transitions from an initial configuration $\langle P; \top\rangle$. A transition, $\langle P; S\rangle \xrightarrow{\pi^+} \langle P'; S'\rangle$, means that $\langle P; S\rangle$ can be transformed into $\langle P'; S'\rangle$ by a single computational step. For simplicity, we assume that all variables and names are declared in the initial configuration i.e., $fv(P) = fn(P) = \emptyset$. We define transitions on configurations next.

3.3 Reduction relation

The reduction relation ,$\xrightarrow{\pi^+}$, over configurations is the least relation satisfying the following rules:

$$\text{COMM: } \frac{S \models_{\Delta} C{=}C'}{\langle((M \ + \ C?[x_1 \ldots x_n].Q) \mid (N{+}C'![C_1 \ldots C_n].P)); S\rangle \xrightarrow{\pi^+} \langle(Q\{C_1,\ldots,C_n/x_1,\ldots,x_n\} \mid P); S\rangle}$$

COMM describes the communication between two normal processes $C?[x_1 \ldots x_n].Q$ and $C'![C_1 \ldots C_n].P$ appearing in a summation, which are sending and receiving along the same channel. We decide from the store whether they are using the same channel. In this sense we can say that the store controls the communication in

the π^+-calculus. Agent $Q\{C_1,\ldots,C_n/x_1,\ldots,x_n\}$ is obtained by replacing, in parallel, every free occurrence of $x_1,\ldots,x_n$ by $C_1,\ldots,C_n$, respectively. Notice that the remaining normal processes, M and N, are discarded, since at most one component of a summation is allowed to execute.

The rules ASK and TELL describe the interaction between constraint agents and the store.

$$\text{TELL: } \langle !o.P;\, S\rangle \xrightarrow{\pi^+} \langle P;\, S\wedge\phi\rangle$$

$$\text{ASK: } \frac{S\models_\Delta o}{\langle ?\phi.P;S\rangle\xrightarrow{\pi^+}\langle P;S\rangle} \quad , \quad \frac{S\models_\Delta \neg\phi}{\langle ?\phi.P;S\rangle\xrightarrow{\pi^+}\langle O;S\rangle}$$

TELL is the way of adding information to the store. It says that $!\phi.P$ adds the constraint o to store S and then activates its continuation P. Such augmentation of the store is the major mechanism in CCP languages for an agent to influence other agents in the system [Sar93]. For example, agent $!(x=a).P$ tells agent $x![C_1\ldots C_n].Q$ that its communication channel is now fixed to a.

ASK is the way of obtaining information from the store. It says that P can be activated if the current store S entails o, or discarded when S entails $\neg o$. For instance, agent $?(x=a\vee x=b).x![C_1\ldots C_n].P$ is able to send $C_1\ldots C_n$ along a channel x, just in case x is either channel a or channel b.

An Ask agent that cannot be reduced in the current store S is said to be *suspended* by S. An agent suspended by S might be reduced in some augmentation of S. In the previous example $?(x=a\vee x=b).x![C_1\ldots C_n].P$ is suspended by the empty store, but if a Tell agent adds $x=a$ to this store, it can be reduced.

$$\text{PAR: } \frac{\langle P;S\rangle\xrightarrow{\pi^+}\langle P';S'\rangle}{\langle Q\mid P;S\rangle\xrightarrow{\pi^+}\langle Q\mid P';S'\rangle}$$

$$\text{DEC-V: } \frac{x\notin fv(S),\ \langle P;S\gg\{x\}\rangle\xrightarrow{\pi^+}\langle P';S'\rangle}{\langle(vx)P;S\rangle\xrightarrow{\pi^+}\langle P';S'\rangle} \quad , \quad \text{DEC-N: } \frac{a\notin fn(S),\ \langle P;S\gg\{a\}\rangle\xrightarrow{\pi^+}\langle P';S'\rangle}{\langle(va)P;S\rangle\xrightarrow{\pi^+}\langle P';S'\rangle}$$

PAR says that reduction can occur underneath composition. DEC-V is the way of introducing new variables. By $S\gg\{C_1,\ldots,C_n\}$ we mean the store $S\wedge C_1=C_1\wedge\ldots\wedge C_n=C_n$, which is obviously equivalent to S. Thus, we add variable $x\notin fv(S)$ to the store by $S\gg\{x\}$ ensuring that x will not be used in following declarations. In the case that $x\in fv(S)$ we can rename x with a new variable $z\notin fv(S)\cup fv(P)$ by using the first item of Definition 3.2 (i.e. $(vx)P\equiv(vz)P\{z/x\}\ if\ z\notin fv(P)$). DEC-N is defined in a similar way.

Rule EQUIV simple says that π^+-equivalent configurations have the same reductions.

$$\text{EQUIV: } \frac{\langle P_1;S_1\rangle\equiv_{\pi^+}\langle P_1';S_1'\rangle,\langle P_2;S_2\rangle\equiv_{\pi^+}\langle P_2';S_2'\rangle,\langle P_1;S_1\rangle\xrightarrow{\pi^+}\langle P_2;S_2\rangle}{\langle P_1';S_1'\rangle\xrightarrow{\pi^+}\langle P_2';S_2'\rangle}$$

In what follows, $\stackrel{\pi^+}{\Longrightarrow}$ will denote the reflexive and transitive closure of $\xrightarrow{\pi^+}$. Finally, we will say that $\langle P';S'\rangle$ is a *derivative* of $\langle P;S\rangle$ iff $\langle P;S\rangle\stackrel{\pi^+}{\Longrightarrow}\langle P';S'\rangle$.

Runtime failure. In the cc-model [Sar93], the invariant property of the store is that it is satisfiable. This can be done in the π^+-Calculus by performing a transition from $\langle !\phi.P;S\rangle$ iff $S\wedge\phi$ is satisfiable, otherwise reducing to a distinguished configuration called *fail*, which denotes a runtime failure. This runtime failure is

propagated thereafter in the usual way. For simplicity we do not consider runtime failures, but we can add these rules orthogonally, as in [Tur95], without affecting any of our results.

Potentiality of reduction. Whenever we augment the store, we may increase the potentiality of reduction, that is, the number of possible transitions from a configuration. The following proposition states that any agent P' obtained from a configuration $\langle P; S_1\rangle$ can be obtained from a configuration $\langle P; S_2\rangle$, S_2 being an augmentation of S_1.

Proposition 3.4 *If* $S_2 \models_\Delta S_1$ *and* $\langle P_1; S_1\rangle \xrightarrow{\pi^+} \langle P_2; S_1'\rangle$ *then* $\langle P_1; S_2\rangle \xrightarrow{\pi^+} \langle P_2; S_2'\rangle$ *and* $S_2' \models_\Delta S_1'$.

Proof: Straightforward from rules TELL, COMM and ASK:

1. *Transitions using* ASK *or* COMM: Since $S_2 \models_\Delta S_1$, for any constraint ϕ such that $S_1 \models_\Delta \phi$ we have $S_2 \models_\Delta \phi$. Thus, any P_2 obtained by using ASK or COMM in $\langle P_1; S_2\rangle$ can be obtained by using the same rule for $\langle P_1; S_2\rangle$. Neither ASK nor COMM modify the store, therefore $S_2' \models_\Delta S_1'$.
2. *Transitions using* TELL: TELL does not consider the store as premise, but it modifies it. In this case $P_1 \equiv !\phi.Q$. For the transition $\langle !\phi.Q; S_1\rangle \xrightarrow{\pi^+} \langle Q; S_1 \wedge \phi\rangle$ we have $\langle !\phi.Q; S_2\rangle \xrightarrow{\pi^+} \langle Q; S_2 \wedge \phi\rangle$. Thus, $S_2' \models_\Delta S_1'$.
3. *Transitions using* EQUIV: If $\langle P_1; S_1\rangle \equiv_{\pi^+} \langle P_3; S_3\rangle$ and $\langle P_3; S_3\rangle \xrightarrow{\pi^+} \langle P_3'; S_3'\rangle$, where $\langle P_3'; S_3'\rangle \equiv_{\pi^+} \langle P_2; S_1'\rangle$, then from 3.3 we have $S_2 \models_\Delta S_3$ and $P_1 \equiv P_3$. The desired result is obtained by applying inductively items 1,2,3,4,5.
4. *Transitions using* DEC-V *or* DEC-N: In this case $P_1 \equiv_{\pi^+} (\upsilon x)Q$. Suppose first that $x \notin fv(S_2)$. If $x \notin fv(S_1)$ we have $S_2 \gg \{x\} \models_\Delta S_1 \gg \{x\}$. If $x \in fv(S_1)$, then using 3.2 (i.e. $(\upsilon x)Q \equiv (\upsilon z)Q\{z/x\}$ *if* $z \notin fv(Q)$, x was replaced in P_1 by a new variable, z, such that $z \notin fv(S_1)$. For any z we have $S_2 \gg \{x\} \models_\Delta S_1 \gg \{z\}$. Similarly, if $x \in fv(S_2)$ it is easy to see that for any y, $S_2 \gg \{y\} \models_\Delta S_1$ and for any z, $S_2 \gg \{x\} \models_\Delta S_1 \gg \{z\}$. Thus, the desired result is obtained by applying inductively items 1,2,3,4,5.
5. *Transitions using* PAR: PAR does not consider the store (directly) as a premise, therefore the desired result is obtained by applying inductively items 1,2,3,4,5.

□

In the following example we will describe the behavior of an agent to clarify our semantics.

Example 3.5 *Agent* P_1 *sends along channel* r *the greater of two numbers* x *and* y, *which is then used by* Q. *Let* Δ *be the set of all sentences valid in the rational numbers.*

$P_1 \equiv (\upsilon x)P_2$; $P_2 \equiv (\upsilon y)P_3$; $P_3 \equiv (\upsilon r)P_4$; $P_4 \equiv (r?[z].Q \mid ?(x > y).r![x] \mid ?\neg(x > y).r![y] \mid !(x = y + 1))$, *i.e.*, $P_1 \equiv (\upsilon x)(\upsilon y)(\upsilon r)(r?[z].Q \mid ?(x > y).r![x] \mid ?\neg(x > y).r![y] \mid !(x = y + 1))$.

Since the variable declarations are different, by DEC-V, the derivatives of $\langle P_1; \top\rangle$ *are the derivatives of* $\langle P_2; \top \gg \{x\}\rangle$, *whose derivatives are the derivatives of* $\langle P_2; \top \gg \{x, y\}\rangle$. *By DEC-N (remember that* r *denotes a name) the derivatives of* $\langle P_2; \top \gg \{x, y\}\rangle$ *are the derivatives of* $\langle P_4; \top \gg \{x, y, r\}\rangle$ *if any. The Ask agents in* P_4 *are suspended by* $\top \gg \{x, y, r\}$, *and there is no other agent sending along channel* r, *so we can only reduce* $\langle P_4; \top \gg \{x, y, r\}\rangle$ *by applying TELL combined with PAR and EQUIV. Thus,*

$$\langle P_4; \top \gg \{x, y, r\}\rangle \xrightarrow{\pi^+} \langle (r?[z].Q \mid ?(x > y).r![x] \mid ?\neg(x > y).r![y] \mid 0); \top \gg \{x, y, r\} \wedge x = y + 1\rangle .$$

Now using ASK combined with PAR and EQUIV,

$$\xrightarrow{\pi^+} \langle (r?[z].Q \mid r![x] \mid ?\neg(x > y).r![y] \mid 0); \top \gg \{x, y, r\} \wedge x = y + 1\rangle .$$

We can eliminate the null process by using $\equiv_{\pi^+}$,

$$\equiv_{\pi^+} \langle (r?[z].Q \mid r![x] \mid ?\neg(x > y).r![y]); \top \gg \{x, y, r\} \wedge x = y + 1\rangle .$$

Using ASK combined with PAR,

$$\xrightarrow{\pi^+} \langle (r?[z].Q \mid r![x] \mid 0) ; \top \gg \{x, y, r\} \wedge x = y + 1 \rangle .$$

Using $\equiv_{\pi+}$ *we can rewrite the processes so they can have the correct format for COMM, and eliminate the null process,*

$$\equiv_{\pi+} \langle ((r?[z].Q + 0) \mid (r![x] + 0)) ; \top \gg \{x, y, r\} \wedge x = y + 1 \rangle .$$

Finally, applying COMM and $\equiv_{\pi+}$,

$$\xrightarrow{\pi^+} \langle (Q\{x/z\} \mid 0) ; \top \gg \{x, y, r\} \wedge x = y + 1 \rangle . \equiv_{\pi+} \langle Q\{x/z\} ; \top \gg \{x, y, r\} \wedge x = y + 1 \rangle$$

Thus, $\langle P_1 ; \top \rangle \xrightarrow{\pi^+} \langle Q\{x/z\} ; \top \gg \{x, y, r\} \wedge x = y + 1 \rangle .$

Behavioral equivalence. In our technical report [VDR97] we defined a reduction equivalence relation, called π^+-reduction equivalence. This relation equates configurations whose agents can communicate on the same channels at each transition. For each channel C, this is expressed by means of an observation predicate $\downarrow_C^S$ detecting the possibility of performing a communication with the external environment along C in a store S. Because of space restrictions we do not develop this here.

Names and Variables. In the π-calculus there is no difference between names and variables [Smo94b]. Names, conveniently used, provides a unique reference to concurrent objects and can also be used for data encapsulation as in [Tur95]. In the π^+-calculus Names and Variables are considered different because of the presence of constraints.

We first give an example to illustrate the difference. Let $P_1 \equiv (\upsilon x)(\upsilon y)(?\neg(x = y).Q)$ and $P_2 \equiv (\upsilon a)(\upsilon b)(?\neg(a = b).Q)$ and let Δ be the set of sentences valid in the natural numbers. It is easy to see that $\langle P_2 ; \top \rangle \xrightarrow{\pi^+} \langle Q ; \top \gg \{a, b\} \rangle$. However, since $?\neg(x = y).Q$ is suspended by $\top \gg \{x, y\}$, there is no reduction for $\langle P_1 ; \top \rangle$.

Finally, we take from [Smo94b] a proposition which states that names are different from any other value that can be uniquely described by a formula:

Proposition 3.6 . *Let* ϕ *be a constraint such* $fv(\phi) = \{x\}$, *and such that* ϕ *determines* x, *that is,* $\Delta \models \exists! x \phi$. *Then* $\Delta \models \neg\phi\{a/x\}$ *for every name* a *not occurring in* ϕ.

Proof: The proof is based on conditions (1) and (2) of the underlined constraint system. See [Smo94b]. □

4 Using the π^+-calculus

4.1 Recursive process definitions

We often wish to define process recursively. For instance suppose you want to define the addition of the natural numbers x, y, returning the result along channel z. This can be done as follows (consider a constraint system providing equations, inequations, natural numbers and the succesor function):
$D_1(x, y, z) \stackrel{def}{=} (?y > 0.(\upsilon x_1)(\upsilon y_1)(!(x_1 = succ(x)) \mid !(succ(y_1) = y) \mid D_1(x_1, y_1, z))) \mid ?y = 0.z![x].$

Recursively-defined processes have the form $D(x_1, \ldots, x_n) \stackrel{def}{=} P$, where P may contain occurrences of D (perhaps with different arguments), $fv(P) \subseteq \{x_1, \ldots, x_n\}$ and $fn(P) = \emptyset$. When no confusion arises we write D (without arguments) instead of $D(x_1, \ldots, x_n) \stackrel{def}{=} P$.

However "definition-making" is not a primitive, since it can be easily encoded using replication as in [Milner, 91] and [Turner, 95]. The idea is to replace each definition $D(x_1,\ldots,x_n) \stackrel{def}{=} P$ (i.e. D) by the process $*d_1?[x_1\ldots x_n].P$ (where d is a new channel) and every call $D(C_1,\ldots,C_n)$ by the process $d![C_1\ldots C_n]$. We give an example to clarify this idea.

Example 4.1 *Let* $Q = (\nu r)(\nu x)(D_1 \mid D_1(x,1,r) \mid !(x = 5) \mid r?[z].Q_1)$ *be a context including the process definition of the previous example,* D_1, *and a call to it. Intuitively, we expect that* $\langle Q;\top\rangle \stackrel{\pi^+}{\Longrightarrow} \langle D_1 \mid Q_1\{x_1/z\};\ldots \wedge x = 5 \wedge x_1 = succ(x) \wedge \ldots\rangle$.

In the translation, D_1 *is replaced by an agent* P, *where*
$P \equiv *d_1?[xyz].(?y > 0.((\nu x_1)(\nu y_1)(!(x_1 = succ(x)) \mid !(succ(y_1) = y) \mid d_1![x_1y_1z])) \mid ?y = 0.z![x])$.

Context Q *is replaced by agent* Q', *where*
$Q' \equiv (\nu d_1)(\nu r)(\nu x)(P \mid d_1![x1r] \mid !(x = 5) \mid r?[z].Q_1)$.

Finally, note that $\langle Q';\top\rangle \stackrel{\pi^+}{\Longrightarrow} \langle P \mid Q_1\{x_1/z\};\top \gg \{d_1,r,x_1,y_1\} \wedge x = 5 \wedge x_1 = succ(x) \wedge succ(y_1) = y\rangle$.

4.2 Encoding Cells

Cells are useful for modelling mutable data structures. They are syntactical entities in the concurrent constraint calculi ρ and γ [Smo94b]. A cell $a : C$ can be thought of as a location a whose current contents is C. In the π^+-calculus, cells can be encoded as follows:

Definition 4.2 (Cell generator) *The cell generator agent is defined as:* $D_2(x,y) \stackrel{def}{=} x?[x_1x_2].(x_1![y].D_2(x,y) + x_2?[z].D_2(x,z))$. *A cell* $a : C$ *is obtained by an invocation to the cell generator; i.e.* $[\![a : C]\!] \stackrel{def}{=} D_2 \mid D_2(a,C)$.

A cell containing the value 5 in location a is $[\![a : 5]\!]$, a cell containing a value greater than 10 in location b is $(\nu x)([\![b : x]\!] \mid !(x > 10))$. The contents of the cell can be read along a channel x_1 or updated by sending a new value along a channel x_2. The summation operator in the cell generator ensures that read and update request cannot be executed concurrently. Thus, when an update request has been accepted, all subsequent read requests will be answered with the updated contents of the cell.

For instance the agent $[\![a : 5]\!] \mid (\nu r)(\nu u)(a![ru].r?[z].Q)$ reads the contents of the cell $[\![a : 5]\!]$ along channel r, which is then used by Q. Note that:

$$\begin{array}{rcl}\langle [\![a : 5]\!] \mid (\nu r)(\nu u)(a![ru].r?[z].Q;\top\rangle & \stackrel{\pi^+}{\Longrightarrow} & \langle D_2 \mid (r![5].D_2(a,5) + u?[z].D_2(a,z)) \mid r?[z].Q;\top \gg \{r,u\}\rangle \\ & \stackrel{\pi^+}{\Longrightarrow} & \langle D_2 \mid D_2(a,5) \mid Q\{5/z\};\top \gg \{r,u\}\rangle \\ & \equiv_{\pi+} & \langle [\![a : 5]\!] \mid Q\{5/z\};\top \gg \{r,u\}\rangle\end{array}$$

The process $[\![a : 5]\!] \mid (\nu r)(\nu u)(a![ru].u![4])$ updates (decreases) the contents of the cell $[\![a : 5]\!]$ by using channel u. Note that:

$$\begin{array}{rcl}\langle [\![a : 5]\!] \mid (\nu r)(\nu u)(a![ru].u![4];\top\rangle & \stackrel{\pi^+}{\Longrightarrow} & \langle D_2 \mid (r![5].D_2(a,5) + u?[z].D_2(a,z)) \mid u![4];\top \gg \{r,u\}\rangle \\ & \stackrel{\pi^+}{\Longrightarrow} & \langle D_2 \mid D_2(a,4) \mid 0;\top \gg \{r,u\}\rangle \\ & \equiv_{\pi+} & \langle [\![a : 4]\!];\top \gg \{r,u\}\rangle\end{array}$$

Finally, a common operation in cells, written ayC, is called *Exchange*. It records the current contents of the cell in a variable y and replaces it by C. Obviously Exchange can be implemented in three steps: reading the contents of the cell, recording this contents in y by using a Tell agent, and finally updating the contents with C. The definition of exchange is:

$D_3(x,y,z) \stackrel{def}{=} (\nu r)(\nu u)(x![ru].r?[z_0].!(y = z_0).x![ru].u![z])$

Thus, exchange operation $[\![ayC]\!]$ is defined as:

$[\![ayC]\!] \stackrel{def}{=} D_3 \mid D_3(a, y, C)$

For instance the agent $P \equiv ([\![a : 5]\!] \mid (\nu x)\,[\![axA]\!])$ transforms $[\![a : 5]\!]$ into $[\![a : 4]\!]$ and records its old contents in a new variable x. Note that:

$$\begin{array}{rll}
\langle P; \top\rangle & \stackrel{\pi^+}{\Longrightarrow} & \langle D_2 \mid D_3 \mid (r![5].D_2(a,5) + u?[z].D_2(a,z)) \mid r?[z].!(x = z).a![ru].u![4]; \top \gg \{x, r, u\}\rangle \\
& \stackrel{\pi^+}{\Longrightarrow} & \langle D_2 \mid D_3 \mid D_2(a,5) \mid !(x = 5).a![ru].u![4]; \top \gg \{x, r, u\}\rangle \\
& \stackrel{\pi^+}{\Longrightarrow} & \langle D_2 \mid D_3 \mid D_2(a,5) \mid a![ru].u![4]; \top \gg \{x, r, u\} \wedge x = 5\rangle \\
& \stackrel{\pi^+}{\Longrightarrow} & \langle D_2 \mid D_3 \mid D_2(a,4); \top \gg \{x, r, u\} \wedge x = 5\rangle \\
& \equiv_{\pi+} & \langle D_3 \mid [\![a : 4]\!]; \top \gg \{x, r, u\} \wedge x = 5\rangle
\end{array}$$

5 Conclusions and future work

We defined the π^+-calculus, an orthogonal extension of the π-calculus to handle constraints. We did this by adding variables and allowing agents to interact through constraints with a global store.

The π^+-calculus is parametrized in a constraint system and thus independent of a particular domain for constraints. We defined the operational semantic through an equivalence relation and a reduction relation on configurations of an agent and a store. We showed how the reduction relation essentially mimics that of the π-calculus but also that the π^+ is able to express the more general notion of potentiality of reduction by the presence of ASK and TELL rules interacting with the store. We described examples showing the transparent interaction of constraints and communicating processes, including the possibility to define mutable data.

Finally, we propose three main directions for future work on this topic:

- Among the most successful work in parallel object-oriented programming languages is that on the POOL family of languages [Ame89]. [Wal95] provides a semantics for a member of this family, via a phrase by phrase translation into the π-calculus. The attributes are translated into cells in the π-calculus, which are similar to those of the π^+-calculus, but without constraints. We believe that an extension of that language integrating OO, Concurrent and Constraints paradigms can be constructed successfully by using the π^+-calculus.
- The Turner's abstract machine [Tur95] is an efficient implementation of the π-calculus used in the programming language PICT [PT96]. Because of the orthogonality of our extension, it is feasible to think in an extension of this abstract machine for the π^+-calculus and also an extension of PICT to consider first-order constraints.
- We will analyze the possibility of incorporating in our calculus the type system for the π-calculus presented in [Tur95]. Moreover, we want to extend our calculus to consider objects (with classes) as a basic entity in a similar way as in [Vas94].

References

[Ame89] P. America. Issues in the design of a parallel object-oriented language. Formal Aspects of Computing, 1989.

[Mil80] Robin Milner. A calculus of communicating systems. Lecture Notes in Computer Science, LNCS 92, 1980.

[Mil91] Robin Milner. The polyadic π-calculus: a tutorial. Technical Report ECS-LFCS-91-180, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh,

UK, October 1991. Also in *Logic and Algebra of Specification*, ed. F. L. Bauer, W. Brauer and H. Schwichtenberg, Springer-Verlag, 1993.

[Mil92] Robin Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, Parts I and II. *Journal of Information and Computation*, 100:1–77, September 1992.

[NM95] Joachim Niehren and Martin Müller. Constraints for Free in Concurrent Computation. In Kanchana Kanchanasut and Jean-Jacques Lévy, editors, *Asian Computing Science Conference*, Lecture Notes in Computer Science, vol. 1023, pages 171–186, Pathumthani, Thailand, December 11–13 1995. Springer-Verlag.

[PRT93] Benjamin C. Pierce, Didier Rémy, and David N. Turner. A typed higher-order programming language based on the pi-calculus. In *Workshop on Type Theory and its Application to Computer Systems, Kyoto University*, July 1993.

[PT96] Benjamin C. Pierce and David N. Turner. Pict: A programming language based on the pi-calculus. Technical report in preparation; available electronically, 1996.

[Sar93] Vijay A. Saraswat. *Concurrent Constraint Programming.* The MIT Press, Cambridge, MA, 1993.

[Smo94a] Gert Smolka. A calculus for higher-order concurrent constraint programming with deep guards. Research Report RR-94-03, Deutsches Forschungszentrum für Künstliche Intelligenz, Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, February 1994.

[Smo94b] Gert Smolka. A foundation for higher-order concurrent constraint programming. In Jean-Pierre Jouannaud, editor, *1st International Conference on Constraints in Computational Logics*, Lecture Notes in Computer Science, vol. 845, pages 50–72, München, Germany, 7–9 September 1994. Springer-Verlag.

[Tur95] David N. Turner. *The Polymorphic Pi-Calculus: Theory and Implementation.* PhD thesis, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, UK, 1995.

[Vas94] Vasco T. Vasconcelos. Typed concurrent objects. In M. Tokoro and R. Pareschi, editors, *Proc. of 8th European Conference on Object-Oriented Programming (ECOOP'94)*, volume 821 of *Lecture Notes in Computer Science*, pages 100–117. Springer-Verlag, 1994.

[VDR97] Frank VALENCIA, Juan Francisco DIAZ, and Camilo RUEDA. The π^+-calculus. Technical Report No.2, AVISPA Team, 1997.

[VP96] Björn Victor and Joachim Parrow. Constraint as processes. In *Proc. of CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 389–405. Springer-Verlag, 1996.

[Wal95] David Walker. Objects in the π-calculus. *Journal of Information and Computation*, 116(2):253–271, 1995.