A Calculus for Concurrent Processes with Constraints [*]

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

May 20, 1999

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 augment the π-calculus with a constraint store and add the notion of *constraint agent* to the standard π-calculus concept of agent. We call this extension the π^+-calculus. We also extend the notion of barbed bisimulation to define behavioral equivalence for the π^+-calculus and use it to characterize some equivalent behaviors derived from constraint agents. The paper discusses examples 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 integration of what appears to be fundamentally different notions of programming has come up as a real need in some domains. A relevant example is the realm of computer supported musical composition. Composers define complex hierarchical structures representing multiple musical dimensions which are to evolve according to either (or both) predetermined trajectories or to satisfaction of a set of compositional rules supplying partial structural information. Both *Object-Oriented* and *Constraint programming* paradigms easily come to mind as relevant for devising music composition tools capable of effectively handling such interactions.

Constraint Programming is a simple and powerful model of computation obtained by considering the notion of computation as deduction over (first-order) systems of partial information. The crucial issue in this paradigm is to replace the notion of *store-as-valuation* central to von Neumann computing with the notion of store as a *constraint*, that is, as pieces of partial information about the values that variables can take [Sar93]. The traditional notions of *read* and *write* are respectively replaced by the new primitives *ask* and *tell*. An *ask* operation is executed to check whether or not the current constraint store *entails* a given constraint (i.e., whether every valuation allowed by the store is also allowed by the constraint). A *tell* operation is executed to add a some constraint to the store. A *tell* operation does not change the value of a variable but may rule out certain values that were previously possible for it. In this sense the store is *monotonically refined* by using tell operations. Thus, in Constraint Programming, computation progresses by accumulating constraints in the store, and by checking whether the store entails constraints.

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

[†]Avispa and Gedi research groups, **Universidad del Valle, Colombia**. E-mail: jdiaz@borabora.univalle.edu.co.

[‡]Director, Avispa research group, **Pontificia Universidad Javeriana Cali, Colombia**.

[§]Avispa research group, **Pontificia Universidad Javeriana Cali, Colombia**. Currently a doctorate student at BRICS, Denmark. E-mail: fvalenci@brics.dk.

The need to establish a firm base for the integration of programming models 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 it [JM95]. A more direct approach is to include the notions of the integrated paradigms in a new calculus [Vas94]. In our quest for the foundations of a computer music language we favor the strategy of relying 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 [RMW92, Mil91], a well known, elegant and simple model of concurrent computation.

Having concurrency in the calculus appeals to us because it is at the heart of the musical craft and also because we believe it is fundamental to both Object-Oriented and Constraint Programming. Indeed, several *tell* and/or *ask* operations can be executed simultaneously and synchronized by the so called *blocking ask* mechanism. Ask agents can be blocked until enough information to entail them is available.

To our knowledge there has been no attempt to encode first-order constraints into the π-calculus nor to orthogonally extend the π-calculus to include them. The κ-calculus [Smo94b] considers only equational constraints, whereas [VP96] shows that equational constraints can be encoded into the π-calculus.

In this paper we define the π^+-calculus, an orthogonal extension of the (polyadic) π-calculus [Mil91]. In the π^+-calculus the notion of *constraint agent* interacting with a *store* is added to the standard π-calculus concept of agent. In this way communication between processes can be parametric in a constraint system. Constraint agents perform the basic *Ask* and *Tell* operations of Concurrent Constraint Programming (CCP) languages, thus adding to the π-calculus synchronization of processes via *blocking ask.* The semantics of the extension is defined operationally in a similar way as the one given for the cc-model in [Sar93]. We illustrate the definition of recursive processes in the π^+-calculus and discuss the representation of *cells* providing a notion of state compatible with concurrency and constraints.

We also extend the concept of barbed bisimulation [MS92] to define behavioral equivalence and to characterize equivalent behaviors derived from constraint agents.

In sections 2, 3, and 4 we present the syntax and semantics of the π^+-calculus and discuss some examples. The proposed syntax adds constraint agents to the standard π-calculus agents.

In section 5 we define a notion of behavioral equivalence for our calculus, in a similar way as it was done for the π-calculus [RMW92, Mil91]. Finally, section 6 shows conclusions and section 7 presents 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.

In what follows, we describe agents informally. In an agent of the form $\pi.P$, the prefix π represents an *atomic action* and P denotes 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.

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

Table 1: π^+-calculus syntax

Constraint agents are new kind of agents whose behavior depends on a global *store.* A store contains information supplied 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".

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

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 $*\pi.P$. This represents a resource that can only be replicated when a requester communicates via π.

We often write $\pi.$ instead of $\pi.O$. We also omit $.O$ in constraint agents $!\phi.O$ and $?\phi.O$.

We describe formally the behavior of agents in the next section.

3 Operational Semantics

3.1 Constraint System

The π^+-calculus is parameterized in a Constraint System. For our purposes it will suffice to found the notion of constraint system on first-order Predicate Logic, as it was done in [Smo94b, Smo94a] [1]. A Constraint

[1]There exist more general and foundationally less heavy alternatives for setting up the notion of a constraint system (e.g [Sar93]); however, by taking Predicate Logic as the starting point, we can build on well-established intuitions, notions and

System consists of:

- 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, ψ 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 $\phi \models\!\dashv_\Delta \psi$, iff $\phi \models_\Delta \psi$ and $\psi \models_\Delta \phi$. We say that ϕ is *satisfiable* in Δ iff $\phi \not\models \perp$. We use $\perp$ 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 underlying 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 $S \models_\Delta \perp$, S is said to be the unsatisfiable store.*

The operational semantics of the π^+-calculus will be defined in terms of an equivalence relation (written $\equiv_{\pi^+}$) on configurations describing computation states and a one-step reduction relation (written $\longrightarrow$) 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, $(\nu a)P$, declares a new name a in P. There are two binding operators for variables: $(\nu x)P$ that binds x in P and $C?[x_1 \ldots x_n].P$ that declares formal parameters $x_1, \ldots, x_n$ in P. *Free names $fn(P)$*, *bound names $bn(P)$*, *free variables $fv(P)$*, *bound variables $bv(P)$* of a process P are defined as usual. 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 satisfying 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.*

notations, and proceed quickly to the issues we want to bring across.

- $*P \equiv P \mid *P$.
- $(\nu a)O \equiv O,\ (\nu x)O \equiv O,\ (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P,\ (\nu x)(\nu y)P \equiv (\nu y)(\nu x)P,\ (\nu a)(\nu x)P \equiv (\nu x)(\nu a)P$.
- *If* $a \notin fn(P)$ *then* $(\nu a)(P \mid Q) \equiv P \mid (\nu a)Q$.
- *If* $x \notin fv(P)$ *then* $(\nu x)(P \mid Q) \equiv P \mid (\nu x)Q$.
- *If* $\phi \models\!\!\!\dashv_\Delta \psi$ *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* $P_1 \equiv P_2,\ S_1 \models\!\!\!\dashv_\Delta S_2,\ fn(S_1) = fn(S_2)$ *and* $fv(S_1) = fv(S_2)$*. Relation* $\equiv_{\pi^+}$ *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, $\langle P; S\rangle \longrightarrow \langle P'; S'\rangle$, 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. $fv(P) = fn(P) = \emptyset$). Notice that this closes the outermost term, but not necessarily terms inside that one. We define transitions on configurations next.

3.3 Reduction relation

The reduction relation ,$\longrightarrow$, over configurations is the least relation satisfying the following rules:

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

COMM describes the communication between two normal processes $C?[x_1 \dots x_n].Q$ and $C'![C_1 \dots C_n].P$ appearing in a summation, which are sending and receiving along the same channel. The current store is used to decide whether they are indeed using the same channel. In this sense we can say that the store controls all communications in the π^+-calculus. Agent $Q\{C_1, \dots, C_n/x_1, \dots, x_n\}$ is obtained by replacing, in parallel, every free occurrence of $x_1, \dots, x_n$ by $C_1, \dots, C_n$, respectively. Notice that the remaining normal processes (M and N,) are discarded since at most one component of a summation is allowed to execute.

Rules ASK and TELL describe the interaction between constraint agents and the store.

$$\text{TELL: } \langle !\phi.P; S\rangle \longrightarrow \langle P; S \wedge \phi\rangle$$

$$\text{ASK: } \frac{S \models_\Delta \phi}{\langle ?\phi.P; S\rangle \longrightarrow \langle P; S\rangle} \quad , \quad \frac{S \models_\Delta \neg\phi}{\langle ?\phi.P; S\rangle \longrightarrow \langle O; S\rangle}$$

TELL is the way of adding information to the store. It says that $!\phi.P$ adds constraint ϕ 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 $x![C_1 \dots C_n].Q$ that its communication channel is now fixed to a.

ASK is the way of "reading" information from the store. The rule says that P can be activated whenever the current store S entails ϕ, or discarded when S entails $\neg\phi$. For instance, agent $?(x = a \vee x = b).x![C_1 \dots C_n].P$ is able to send $C_1 \dots C_n$ along channel x, just in case x represents either channel a or channel b. The reason for discarding P is the monotonicity of the constraint store. Of course, an inconsistent store entails both ϕ

and $\neg\phi$, so that the rules introduces nondeterminism in this case. However, there is no way to augment an inconsistente store so this indeterminism really adds nothing computationally.

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 "awakened" and reduced in some augmentation of S. In the previous example $?(x = a \vee x = b).x![C_1 \dots C_n].P$ is suspended by the empty store, but if a *tell* agent adds $x = a$ to this store then it can be reduced.

$$\text{PAR: } \frac{\langle P;S\rangle \longrightarrow \langle P';S'\rangle}{\langle Q \mid P;S\rangle \longrightarrow \langle Q \mid P';S'\rangle}$$

$$\text{DEC-V: } \frac{x \notin fv(S) \qquad \langle P;S \gg \{x\}\rangle \longrightarrow \langle P';S'\rangle}{\langle (\nu x)P;S\rangle \longrightarrow \langle P';S'\rangle} \quad , \quad \text{DEC-N: } \frac{a \notin fn(S) \qquad \langle P;S \gg \{a\}\rangle \longrightarrow \langle P';S'\rangle}{\langle (\nu a)P;S\rangle \longrightarrow \langle P';S'\rangle}$$

PAR says that reduction can occur underneath composition. DEC-V is the way of introducing new variables. By $S \gg \{C_1, \dots, C_n\}$ we mean the store $S \wedge C_1 = C_1 \wedge \dots \wedge C_n = C_n$, that 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 subsequent 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. $(\nu x)P \equiv (\nu z)P\{z/x\}\ if\ z \notin fv(P)$). DEC-N is defined in a similar way.

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

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

It is easy to see that the π-calculus is a special case of the π^+-calculus parameterized in a minimal constraint system (i.e. a constraint system including only names in the domain under consideration), without resorting to constraint agents.

In what follows, $\Longrightarrow$ will denote the reflexive and transitive closure of $\longrightarrow$. Finally, we will say that $\langle P';S'\rangle$ is a *derivative* of $\langle P;S\rangle$ iff $\langle P;S\rangle \Longrightarrow \langle P';S'\rangle$.

Runtime failure. In the cc-model [Sar93], the invariant property of the store is that it is satisfiable. This property can be maintained in the π^+-Calculus by performing a transition from $\langle !\phi.P;S\rangle$ iff $S \wedge \phi$ is satisfiable and otherwise reducing to a distinguished configuration called *fail* denoting a runtime failure. This runtime failure is propagated thereafter in the usual way. For the sake of 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 the store is augmented, the potentiality of reduction, that is, the number of possible transitions from a configuration, increases. 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 $D \models_\Delta C$ and $\langle P_1;C\rangle \longrightarrow \langle P_2;C'\rangle$ then $\langle P_1;D\rangle \longrightarrow \langle P_2;D'\rangle$ and $D' \models_\Delta C'$.*

Proof: We proceed by structural induction on the form of reduction derivation expressions (RDE). RDE's are inductively (and implicitly) defined by the reduction rules given above. A RDE is just a sequence of reduction rules used in proving membership to the reduction relation. COMM, TELL, ASK are thus base forms in the inductive definition of RDE's, whereas PAR, DEC-V, DEC-N and EQUIV are constructors.

1. Base forms:
   ASK: Let P_1 be $?\phi.P$. Since $D \models_\Delta \phi$ whenever $D \models C$ and $C \models \phi$, we have that $\langle P_1;D\rangle \longrightarrow \langle P;D\rangle$ whenever $\langle P_1;C\rangle \longrightarrow \langle P;C\rangle$

COMM: A similar reasoning applies, since $D \models_\Delta (I = I')$ whenever $C \models_\Delta (I = I')$.

TELL: Let P_1 be $!\phi.P$. Since there are no premises, we have $\langle P_1; D\rangle \longrightarrow \langle P; D \wedge \phi\rangle$ and also $\langle P_1; C\rangle \longrightarrow \langle P; C \wedge \phi\rangle$. Since $D \wedge \phi \models_\Delta C \wedge \phi$ whenever $D \models_\Delta C$, we have the desired result.

2. Constructors:

   EQUIV: Suppose the proposition is true for the RDE establishing the reduction stated in the premise of the EQUIV rule. Let a particular instance of this rule be

$$\text{(H):}\ \frac{\langle Q_1;S_1\rangle \equiv_{\pi^+} \langle P_1;C\rangle \qquad \langle Q_2;C_2\rangle \equiv_{\pi^+} \langle P_2;C'\rangle \qquad \langle Q_1;S_1\rangle \longrightarrow \langle Q_2;C_2\rangle}{\langle P_1;C\rangle \longrightarrow \langle P_2;C'\rangle}$$

   By the inductive hypothesis, there is a reduction $\langle Q_1; D_1\rangle \longrightarrow \langle Q_2; D_2\rangle$ with $D_1 \models_{\pi^+} S_1$ and $D_2 \models_{\pi^+} C_2$. By the congruences in the premises of (H), we also have $D_1 \models_{\pi^+} C$ and $D_2 \models_{\pi^+} C'$. Now, let D, D' be such that $D \equiv_{\pi^+} D_1$ and $D' \equiv_{\pi^+} D_2$. The EQUIV rule can thus be applied:

$$\frac{\langle Q_1;D_1\rangle \equiv_{\pi^+} \langle P_1;D\rangle \qquad \langle Q_2;D_2\rangle \equiv_{\pi^+} \langle P_2;D'\rangle \qquad \langle Q_1;D_1\rangle \longrightarrow \langle Q_2;D_2\rangle}{\langle P_1;D\rangle \longrightarrow \langle P_2;D'\rangle}$$

   and the stated property follows.

3. DEC-V: Suppose the proposition is true for the RDE establishing the reduction stated in the premise of the DEC-V rule. Let $\langle P; C \gg \{x\}\rangle \longrightarrow \langle P'; C'\rangle$ be such reduction. Now, let D be a store such that $x \notin fv(D)$ and $D \models_{\pi^+} C$. Obviously, $D \gg \{x\} \models_{\pi^+} C \gg \{x\}$. By the inductive hypothesis, we must have $\langle P; D \gg \{x\}\rangle \longrightarrow \langle P'; D'\rangle$, with $D' \models_{\pi^+} C'$. We have then the necessary premises for applying rule DEC-V and conclude $\langle P; D\rangle \longrightarrow \langle P'; D'\rangle$, as needed.

   Constructor DEC-N is proved analogously.

4. PAR: Suppose the proposition is true for the RDE establishing the reduction stated in the premise of the PAR rule. Let $\langle P; C\rangle \longrightarrow \langle P'; C'\rangle$ be such reduction. Let let D be a store such that $D \models_{\pi^+} C$. By the inductive hypothesis, there exists a reduction $\langle P; D\rangle \longrightarrow \langle P'; D'\rangle$, with $D' \models_{\pi^+} C'$. But then rule PAR can be applied to conclude $\langle Q \mid P; D\rangle \longrightarrow \langle Q \mid P'; D'\rangle$.

□

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.*

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

Since 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 in turn 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 $\top \gg \{x, y, r\}$. Since there is no other agent sending along channel r, $\langle P_4; \top \gg \{x, y, r\}\rangle$ can only be reduced by applying TELL combined with PAR and EQUIV. Thus,*

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

Now using ASK combined with PAR and EQUIV,

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

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,

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

Using $\equiv_{\pi^+}$ processes can be rewritten so they 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 $\equiv_{\pi^+}$,

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

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

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

We first give an example to illustrate the difference. Let $P_1 \equiv (\nu x)(\nu y)(?\neg(x = y).Q)$ and $P_2 \equiv (\nu a)(\nu b)(?\neg(a = b).Q)$ and let Δ be the set of sentences valid in the natural numbers. It is easy to see that

$$\langle P_2; \top \rangle \longrightarrow \langle Q; \top \gg \{a, b\} \rangle \,.$$

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 stating 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 the two conditions for a consistent theory mentioned in the definition of a 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 successor function):
$D_1(x, y, z) \overset{def}{=} (?y > 0.(\nu x_1)(\nu y_1)(!(x_1 = succ(x)) \mid !(succ(y_1) = y) \mid D_1(x_1, y_1, z))) \mid ?y = 0.z![x]$.

Recursively-defined processes have the form $D(x_1, \ldots, x_n) \overset{def}{=} P$, 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 $D(x_1, \ldots, x_n) \overset{def}{=} P$. That is, we use D without arguments as a "reference" to a previously given process definition. Intuitively, definitions behave like "macros" that are to be "expanded" at invocations when needed.

However, "definition-making" is not a primitive since it can easily be encoded using replication as in [Milner, 91] and [Turner, 95]. The idea is to replace each definition $D(x_1, \ldots, x_n) \overset{def}{=} P$ (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 process where, as explained above, D_1 names the process definition given at the beginning of the section, and $D_1(x,1,r)$ denotes a call to it. Intuitively, we expect that*

$$\langle Q; \top \rangle \Longrightarrow \langle D_1 \mid Q_1\{x_1/z\}; \ldots \wedge x = 5 \wedge x_1 = succ(x) \wedge \ldots \rangle\,.$$

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

Process 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 $\langle Q'; \top \rangle \Longrightarrow \langle P \mid Q_1\{x_1/z\}; \top \gg \{d_1, r, x_1, y_1\} \wedge x = 5 \wedge x_1 = succ(x) \wedge succ(y_1) = y \rangle\,.$

4.2 Encoding Cells

Cells are useful for modeling mutable data structures. They are syntactic 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:*

$$D_2(x,y) \overset{def}{=} x?[x_1 x_2].(x_1![y].D_2(x,y) + x_2?[z].D_2(x,z)).$$

A cell $a : C$ is obtained by an invocation to the cell generator; i.e.

$$[\![a : C]\!] \overset{def}{=} D_2 \mid D_2(a, C).$$

A cell containing the value 5 at location a is $[\![a : 5]\!]$, a cell containing a value greater than 10 at 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 requests 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, 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:

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

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:

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

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, *non atomic 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:
$D_3(x,y,z) \overset{def}{=} (\nu r)(\nu u)(x![ru].r?[z_0].!(y=z_0).x![ru].u![z])$

Thus, the non atomic *exchange* operation $[\![ayC]\!]$ is defined as:
$[\![ayC]\!] \overset{def}{=} D_3 \mid D_3(a,y,C)$

For instance, agent $P \equiv ([\![a:5]\!] \mid (\nu x)\,[\![ax4]\!])$ transforms $[\![a:5]\!]$ into $[\![a:4]\!]$ and records its old contents in a new variable x. Note that:

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

Atomic *exchange* could similarly be defined by considering cells with a single "read-and-update" operation. Namely,

$$D_2(x,y) \overset{def}{=} (\nu z)\ x?[x_1].(x_1![yz].D_2(x,z).$$

5 Behavioral equivalence

In the π-calculus, barbed bisimulation [MS92] equates agents that can match each other's interaction and can communicate on the same channels at each step. For each channel C, the latter is expressed by means of an observation predicate $\downarrow_C$ detecting the possibility of performing a communication with the external environment along C. We take from [Mil91] the notions of *unguardness* and *observability* and modify them by parameterizing in a store (the store controls all communication in the π^+-calculus, anyhow).

Definition 5.1 (Observability in the π^+-calculus) *An agent P occurs unguarded in Q iff it has some occurrence in Q that is not under a prefix or within a constraint agent. An agent Q is observable at C in a store S, written $Q\downarrow^S_C$, iff some $\pi.P'$ occurs unguarded in Q and $S \models_\Delta C=C'$, where C' is the subject of π and it is not bound by a binding operator (νx) or (νa). More precisely, given a channel C:*

$$\begin{array}{ll} O \not\downarrow^S_C . & (\nu a)P\downarrow^S_C\ iff\ P\downarrow^S_C\ and\ S\models_\Delta C\neq a. \\ !\phi.P \not\downarrow^S_C . & (\nu x)P\downarrow^S_C\ iff\ P\downarrow^S_C\ and\ S\models_\Delta C\neq x. \\ ?\phi.P \not\downarrow^S_C . & (P \mid Q)\downarrow^S_C\ iff\ P\downarrow^S_C\ or\ Q\downarrow^S_C . \\ C'![x_1\ldots x_n].P\downarrow^S_C\ iff\ S\models_\Delta C=C'. & (P\ +\ Q)\downarrow^S_C\ iff\ P\downarrow^S_C\ or\ Q\downarrow^S_C . \\ C'?[x_1\ldots x_n].P\downarrow^S_C\ iff\ S\models_\Delta C=C'. & *P\downarrow^S_C\ iff\ P\downarrow^S_C . \end{array}$$

Definition 5.2 (Strong π^+- reduction equivalence) *(Strong) π^+- reduction equivalence (or π^+-barbed bisimulation), $\dot\sim_{\pi^+}$, is the largest equivalence relation $\mathcal{R}$ over configurations such that $(\langle P_1;S_1\rangle,\langle P_2;S_2\rangle)\in\mathcal{R}$ implies:*

1. *If $\langle P_1;S_1\rangle \longrightarrow \langle P_1';S_1'\rangle$ then $\langle P_2;S_2\rangle \longrightarrow \langle P_2';S_2'\rangle$*
   for some $\langle P_2';S_2'\rangle$ such that $(\langle P_1';S_1'\rangle\,,\langle P_2';S_2'\rangle)\in\mathcal{R}$.
2. *For each channel C if $P_1\downarrow^{S_1}_C$ then $P_2\downarrow^{S_2}_C$*

In other words, the first condition says that any transition from P_1 in a store S_1 can be simulated by a transition from P_2 in a store S_2, such that derivatives $\langle P_1';S_1'\rangle$ and $\langle P_2';S_2'\rangle$ remain in the simulation. Note that in the second condition we do not require S_1 and S_2 to be equivalent, but we just require that they allow communication with the external environment along the same channels.

However, π^+-reduction equivalence is not preserved by agent constructions. For example,

$$\langle x?[z] \,|\, y?[z]; S\rangle \dot{\sim}_{\pi^+} \langle x?[z] + y?[z]; S\rangle$$

, but communicating the agents along x with $x![r]$, we get

$$\langle x?[z] \,|\, y?[z] \,|\, x![r]; S\rangle \not\dot{\sim}_{\pi^+} \langle (x?[z] + y?[z]) \,|\, x![r]; S\rangle.$$

This motivates the definition appearing below:

Definition 5.3 (Strong π^+-context reduction equivalence) *We say that P_1 in a store S_1 is (strong) π^+-context reduction equivalent to P_2 in a store S_2, written, $\langle P_1; S_1\rangle \sim_{\pi^+} \langle P_2; S_2\rangle$, iff for any agent context $\mathcal{C}[.]$, $\langle \mathcal{C}[P_1]; S_1\rangle \dot{\sim}_{\pi^+} \langle \mathcal{C}[P_2]; S_2\rangle$. An agent context $\mathcal{C}[.]$ is an agent term with a single hole, such that placing an agent in the hole yields a well-formed process. $\sim_{\pi^+}$ is said to be the (strong) π^+-context reduction equivalence relation.*

The weak version of the equivalences, where a reduction from a configuration can be simulated by several reductions from the other, is obtained in the standard way: Let $\Downarrow^S_C$ be $\Longrightarrow\downarrow^S_C$, the composition of the two relations. Weak π^+- *reduction equivalence*, written $\dot{\approx}_{\pi^+}$, is defined by replacing in Definition 5.2 the reduction $\langle P_2; S_2\rangle \longrightarrow \langle P_2'; S_2'\rangle$ with $\langle P_2; S_2\rangle \Longrightarrow \langle P_2'; S_2'\rangle$ and $P_2 \downarrow^{S_2}_C$ with $P_2 \Downarrow^{S_2}_C$. Weak π^+- *context reduction equivalence,* written $\approx_{\pi^+}$, is defined by replacing in Definition 5.3 $\dot{\sim}_{\pi^+}$ with $\dot{\approx}_{\pi^+}$.

The standard congruence theorems (congruence of bisimulation wrt. parallel composition, substitution, etc.) can be shown in a similar way as it was done in the π-calculus [RMW92, Mil91]. In what follows we only show some equivalent behaviors derived from constraint agents.

The following proposition shows an obvious result: There is no need to replicate a Tell agent:

Lemma 5.4 *For any Q, $\langle Q \mid *!\phi.P; S\rangle \dot{\approx}_{\pi^+} \langle Q \mid !\phi. * P; S\rangle$*

Proof: Consider a binary relation R on configurations defined recursively as follows:

1. $(\langle Q \mid *!\phi.P; S\rangle, \langle Q \mid !\phi. * P; S\rangle) \in R$
2. $(\langle Q' \mid *!\phi.P; S_1'\rangle, \langle Q' \mid T_2; S_2'\rangle) \in R$ whenever
   $(\langle Q \mid *!\phi.P; S_1\rangle, \langle Q \mid T_2; S_2\rangle) \in R$ and
   - $\langle Q; S_1\rangle \longrightarrow \langle Q'; S_1'\rangle$
   - $\langle Q; S_2\rangle \longrightarrow \langle Q'; S_2'\rangle$
   - $S_1 \equiv S_2$
3. $(\langle Q \mid P \mid *!\phi.P; S \wedge \phi\rangle, \langle Q \mid P \mid T_2; S_2'\rangle) \in R$ whenever
   $(\langle Q \mid *!\phi.P; S\rangle, \langle Q \mid T_2; S_2\rangle) \in R$ and
   - $S \equiv S_2$
   - $\left\{\begin{array}{ll} S_2 = S_2' & if\ T_2 = *P\ and\ (S_2 \wedge \phi \equiv S_2 \\ (S_2' = S_2 \wedge \phi) \wedge (T_2' = *P) & if\ T_2 = !\phi.*P \end{array}\right\}$

We claim R is a weak π^+-reduction equivalence relation. First, it is straightforward to show that for all $(\langle P_1; S_1\rangle, \langle P_2; S_2\rangle) \in R$ we have $S_1 \equiv S_2$. Notice also that for any $(\langle P_1; S_1\rangle, \langle P_2; S_2\rangle) \in R$ we have that $P_2 \Downarrow^{S_2}_C$ for every channel C such that $P_1 \Downarrow^{S_1}_C$. Indeed, it can be easily shown that if some pair $(A, B) \in R$ did not have this property then no other pair could have it either, which contradicts the fact that the pair in item 1 of the definition of R does.

Now, suppose R is not a weak π^+-reduction equivalence relation. Then it must be the case that there exists a pair $(\langle Q \mid *!\phi.P; S_1\rangle, \langle Q \mid T_2; S_2\rangle)$ such that for some possible reduction $\langle Q \mid *!\phi.P; S_1\rangle \longrightarrow \langle Q'; S_1'\rangle$ there is no weak reduction $\langle Q \mid T_2; S_2\rangle \Longrightarrow \langle Q''; S_2'\rangle$ such that $(\langle Q'; S_1'\rangle, \langle Q''; S_2'\rangle)$. Consider all possible one step reductions from configuration $\langle Q \mid *!\phi.P; S_1\rangle$. These involve either Q or $*!\phi.P$ but not both since P is guarded in $*!\phi.P$. That is, either

- $Q' = Q_1 \mid *!\phi.P$ and $\langle Q; S_1\rangle \longrightarrow \langle Q_1; S_1'\rangle$, or
- $Q' = Q \mid P \mid *!\phi.P$ and $\langle *!\phi.P; S_1\rangle \longrightarrow \langle P \mid *!\phi.P; S_1 \wedge \phi\rangle$

In the first case, we must also have $\langle Q; S_2\rangle \longrightarrow \langle Q_1; S_2'\rangle$ since $S_1 \equiv S_2$. But then $(\langle Q_1 \mid *!\phi.P; S_1'\rangle, \langle Q_1 \mid T_2; S_2'\rangle) \in R$, contradicting the hypothesis. Now, notice that either $T_2 = !\phi.*P$ or $T_2 = *P$ since rule 2 in the definition of R does not modify T_2 and rule 3 either does not modify T_2 or assumes it is equal to $!\phi.*P$ and changes it to $*P$. But then there is always a reduction $\langle Q \mid T_2; S_2\rangle \Longrightarrow \langle Q \mid P \mid T_2'; S_2'\rangle$ with T_2, T_2', S_2, S_2' satisfying all conditions in item 3 of the definition of R. Thus $(\langle Q \mid P *!\phi.P; S_1 \wedge \phi\rangle, \langle Q \mid P \mid T_2'; S_2'\rangle) \in R$, contradicting the hypothesis. □

Theorem 5.5 $\langle *!\phi.P; S\rangle \approx_{\pi^+} \langle !\phi.*P; S\rangle$

Proof: To show that

$$\langle P_1; S\rangle \approx_{\pi^+} \langle P_2; S\rangle$$

we only need to consider contexts $\mathcal{C}[.]$, where both P_1 and P_2 can act, that is, with P_1 and P_2 occurring unguarded in $\mathcal{C}[P_1]$ and $\mathcal{C}[P_2]$, respectively . Since $*!\phi.P$ and $!\phi.*P$ are not normal processes, these contexts have the form

$$(\nu C^i)(Q \mid (\nu C'^j)\ .\ )$$

for any Q (here, C^k denotes a (possibly empty) sequence of variables and/or names, $C_1 \ldots C_k$, and (νC^k) denotes $(\nu C_1)\ldots(\nu C_k)$). Thus, assuming that there is no need of renaming, from DEC-V (or DEC-N) we should show that

$$\left\langle Q \mid *!\phi.P; S \gg \{C_1, \ldots, C_i, C'_1, \ldots, C'_j\}\right\rangle \dot{\approx}_{\pi^+} \left\langle Q \mid !\phi.*P; S \gg \{C_1, \ldots, C_i, C_1', \ldots, C_j'\}\right\rangle$$

. The proof follows from the lemma 5.4 □

As we said before, in the cc-model a computation fails if a Tell agent attempts to add an inconsistent information to the store. In this sense two agents rendering the store unsatisfiable, and able to use its inconsistent information (i.e. they can use the inconsistent store to be observed at external channels) should be equivalent. In the π^+-calculus these agents will be weak reduction equivalent:

Theorem 5.6 *Suppose $\langle P_1; S_1\rangle \Longrightarrow \langle P_1'; S_1'\rangle$ and $\langle P_2; S_2\rangle \Longrightarrow \langle P_2'; S_2'\rangle$ where S_1' and S_2' are unsatisfiable stores and each P_i' (with $i \in \{1,2\}$) includes at least one unguarded agent. Then $\langle P_1; S_1\rangle \dot{\approx}_{\pi^+} \langle P_2; S_2\rangle$.*

Proof: It is straightforward from Definition 5.1. Both configurations can reduce (in zero or more steps) to a configuration having an unsatisfiable store with an agent including an unguarded agent. From Definition 5.1 this agent is observable at any channel, so it can simulate (weakly) any derivative from the other. □

In the previous section we gave an example of the behavior of $P_1 \equiv (\nu x)(\nu y)(?\neg(x = y).Q)$ and $P_2 \equiv (\nu a)(\nu b)(?\neg(a = b).Q)$ to illustrate the difference between names and variables. Assuming that Q is observable at some channel, the difference is preserved by our behavioral equivalence, that is, $\langle P_1; \top\rangle \not\approx_{\pi^+} \langle P_2; \top\rangle$.

6 Conclusions

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.

We believe that including constraints in concurrent processes calculi provides a powerful formal base for music composition tools. We think that the development of computational models and of tools for computer aided music composition should go hand in hand to benefit from insights at the user level while maintaining a coherent formal base. In recently proposed computer aided musical composition systems such as *Situation* [BR98] constraints and *Common Lisp* objects can be used to define complex musical structures. In the same spirit, but more closely integrated to the underlying Smalltalk language, *Backtalk* [PR95] provides a framework for handling constraint satisfaction within an object environment. Both systems have been successfully used in practical musical settings. In both applications, however, the constraint engine is a black box barely accessible to the user. Moreover, communicating data structures back and forth between the constraint and process (object) models is often awkward. In fact, processes containing partial information and "standard" fully instantiated processes are not really amenable to the same kind of computational treatment. In musical applications this lack of communication potential can be specially troublesome since the approach of the composer involves for the most part constant refinement and modification of compositional models based on the acoustical result of partial implementations.

The π^+-calculus is parameterized 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.

We extended to the π^+-calculus the notion of observability of an agent and defined equivalence of configurations under context reductions. These allowed us to prove that there is no need to replicate Tell agent and that configurations having different agents communicating through inconsistent stores are weak reduction equivalent, thus confirming the usual behavior in CCP of Tell agents attempting to add inconsistent information to the store.

7 Future Work

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. In particular, an object is represented as an agent whose algebraic structure reflects the internal structure of the object, and whose pattern of interaction captures the way in which the object may interact with others. The variables (which represent 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.

[BR98] A. Bonnet and C. Rueda. *Situation: Un Langage Visuel Basee sur les Contraintes pour la Composition Musicale.* HERMES, Paris, France, 1998. in: Recherches et Applications en Informatique Musicale. M. Chemillier and F. Pachet (Eds).

[JM95] N. Joachim and M. 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 1995. Springer-Verlag.

[Mil91] R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS-LFCS-91-180, Laboratory for Foundations of Computer Science,, October 1991. Also in *Logic and Algebra of Specification*, ed. F. L. Bauer, W. Brauer and H. Schwichtenberg, Springer Verlag , 1993.

[MS92] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proc. of 19th International Colloquium on Automata, Languages and Programming (ICALP '92)*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer-Verlag, 1992.

[PR95] F. Pachet and P. Roy. Mixing constraints and objects: A case study in automatic harmonization. In *Proceedings of TOOLS Europe'95*, pages 119–126, Versailles, France, 1995. Prentice-Hall.

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

[RMW92] J. Parrow R. Milner and D. Walker. A calculus of mobile processes, Parts I and II. *Journal of Information and Computation*, 100:1–77, September 1992.

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

[Smo94a] G. 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] G. 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, September 1994. Springer-Verlag.

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

[Vas94] V. 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.

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

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