

# A Calculus for Concurrent Processes with Constraints <sup>\*</sup>

Juan Francisco DIAZ FRIAS <sup>†</sup>   Camilo RUEDA <sup>‡</sup>   Frank D. VALENCIA POSSO <sup>§</sup>

May 20, 1999

## Abstract

The  $\pi$ -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  $\pi$ -calculus with a constraint store and add the notion of *constraint agent* to the standard  $\pi$ -calculus concept of agent. We call this extension the  $\pi^+$ -calculus. We also extend the notion of barbed bisimulation to define behavioral equivalence for the  $\pi^+$ -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,  $\pi$ -calculus,  $\pi^+$ -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.

---

<sup>\*</sup>This work is supported in part by grant 1251-14-041-95 from Colciencias-BID.

<sup>†</sup>Avispa and Gedi research groups, **Universidad del Valle, Colombia**. E-mail: jdiaz@borabora.univalle.edu.co.

<sup>‡</sup>Director, Avispa research group, **Pontificia Universidad Javeriana Cali, Colombia**.

<sup>§</sup>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  $\pi$ -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  $\pi$ -calculus nor to orthogonally extend the  $\pi$ -calculus to include them. The  $\kappa$ -calculus [Smo94b] considers only equational constraints, whereas [VP96] shows that equational constraints can be encoded into the  $\pi$ -calculus.

In this paper we define the  $\pi^+$ -calculus, an orthogonal extension of the (polyadic)  $\pi$ -calculus [Mil91]. In the  $\pi^+$ -calculus the notion of *constraint agent* interacting with a *store* is added to the standard  $\pi$ -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  $\pi$ -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  $\pi^+$ -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  $\pi^+$ -calculus and discuss some examples. The proposed syntax adds constraint agents to the standard  $\pi$ -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  $\pi$ -calculus [RMW92, Mil91]. Finally, section 6 shows conclusions and section 7 presents future work.

## 2 Syntax

The syntax of the  $\pi^+$ -Calculus is given in Table 1. There are only two kinds of entities in the  $\pi$ -calculus: *Channels* and *Agents* (or *Processes*). The  $\pi^+$ -calculus adds *Constraint* agents and agents declaring variables to the standard  $\pi$ -calculus agents. In the  $\pi$ -calculus *names* denote channels. The  $\pi^+$ -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  $\pi$  represents an *atomic action* and  $P$  denotes the continuation of  $\pi.P$ . When  $\pi$  is a writing prefix  $C![C_1 \dots C_n]$ ,  $\pi.P$  means “send  $C_1, \dots, C_n$  along channel  $C$  and then activate  $P$ ”. When  $\pi$  is a reading prefix  $C?[x_1 \dots x_n]$ ,  $\pi.P$  means “receive the arguments, say  $x_1, \dots, x_n$ , along channel  $C$ , use them in  $P$  and then activate  $P$ ”. In both cases  $C$  is called the *subject* of  $\pi$ .

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

Table 1:  $\pi^+$ -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  $\phi$  to the store and then activate  $P$ ”. The *ask* agent  $?\phi.P$  means “Activate  $P$  if constraint  $\phi$  is a logical consequence of the information in the store”.

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

Agent  $P | Q$  means that  $P$  and  $Q$  are concurrently active, so they can act independently (and possibly communicate).  $*P$ , “Bang  $P$ ”, means  $P | P \dots$  ( 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  $\pi$ .

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  $\pi^+$ -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]<sup>1</sup>. A Constraint

<sup>1</sup>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





















