A Markov Model for Probabilistic Concurrent Constraint Programming

Alessandra Di Pierro and Herbert Wiklicky

Abstract

This paper introduces a new approach towards the semantics of Concurrent Constraint Programming (CCP), which is based on operator algebras. In particular, we show how stochastic matrices can be used for modelling both a quantitative version of nondeterminism (in the form of a probabilistic choice) and synchronisation. We will assume Probabilistic Concurrent Constraint Programming (PCCP) as the reference paradigm. The presented model subsumes CCP as a special case.

The model is obtained by a fixpoint construction on a space of linear operators. For the purpose of this paper, we consider only finite constraint systems. This allows us to define our model by using finite dimensional linear operators, i.e. matrices. We show that our model is equivalent to a notion of observables which extends the standard one in CCP (input/output behaviour) by including information about the likelihood of the computed results. The model also captures infinite computations.

Keywords: Concurrent Constraint Programming, Probability, Markov Chains, Operator Algebras, Denotational Semantics, Synchronisation.

1 Introduction

Probabilistic Concurrent Constraint Programming (PCCP) was introduced in [5, 4] as a probabilistic extension of Concurrent Constraint Programming (CCP) [11]. A PCCP program has a natural operational interpretation as a simple stochastic process similar to the random walk on a graph, the latter being the graph associated to the transition system. This kind of processes are intensively studied in mathematics and several models have been introduced for them, mainly based on the theory of stochastic matrices and Markov chains. It is therefore not surprising that we look at these structures for providing PCCP with a denotational semantics.

To this purpose, we look at the free vector space on the underlying set of constraints, i.e. the set of (formal) linear combinations of constraints. In this setting we define a semantics of PCCP: The domain of denotations is defined as the algebra of linear operators on the free vector space on the constraint system. For finite constraint systems we can represent these operators simply by matrices. The semantics is then given via (one of) the fixpoint(s) of an inductively defined operator Ψ on this matrix algebra. The

City University, Northampton Square, London EC1V OHB. E-mail: adp@cs.city.ac.uk and herbert@cs.city.ac.uk.

Work supported by Marie Curie grants No.ERBFMBICT960884 and No.ERBFMBICT960907.

$$
\begin{array}{lcl}
P & ::= & D.A \\
D & ::= & \epsilon \mid D.D \mid p(x) : -A \\
A & ::= & \mathbf{stop} \mid \mathbf{tell}(c) \mid \square_{i=1}^{n}\ c_i|p_i \rightarrow A_i \mid A \parallel A \mid \exists_x A \mid p(x)
\end{array}
$$

Table 1: The syntax for PCCP.

definition of Ψ encodes the structure of the underlying constraint system in an elegant way. Finally, we show that the denotation of each agent is equivalent to a notion of observables defined as the exact results of all (finite and infinite) computations together with their associated probabilities.

From the Markov model for PCCP we can obtain a model for CCP by simply abstracting away from concrete probabilities and considering 0/1 entries instead of real numbers in $[0, 1]$ for the matrices denoting the programs. One of the main difficulties in constructing a model for CCP is due to the combination of nondeterminism and synchronisation. Previous approaches towards the semantics of CCP are based on complicated structures like reactive sequences [3, 2], or bounded trace operators [11], or categorical constructions [9]. The model we present is based on the use of less conventional structures (linear operators) and relies on well-established mathematical theories and results. Moreover, it is able to capture also infinite computations. This aspect has not always been taken into account in the previous approaches and only [2] and [9] present a solution to this problem. We demonstrate that the presence of additional information, in the form of "quantities" associated to the transition steps, is essential for the convergence of the fixpoint operator and then for defining a semantical model for infinite computation. We will discuss examples in which the incorporation of probabilities enables us to construct fixed points where classical approaches fail to do so.

The adoption of functional analytical and operator algebraic methods constitutes the main novelty of this work. It also creates a natural link between (a certain field of) computer science and different disciplines such as physics, by the use of the theory of stochastic processes.

The plan of the paper is as follows: In Section 2 we recall the syntax and the operational semantics of PCCP. In Section 3 we present a denotational semantics for PCCP and show its correspondence with the previously introduced operational one. Finally in Section 4 we give some examples showing the construction of the Markov model for some PCCP programs. Section 5 contains a summing up of the results presented in the paper and some directions for further work.

2 Probabilistic Concurrent Constraint Programming

In order to allow the formulation of various randomised algorithms within the declarative framework of CCP we introduce in [4, 5] a new language we call Probabilistic Concurrent Constraint Programming (PCCP).

The syntax of PCCP is an extension of classical CCP as given in Table 1. The intuitive semantics is identical to the common one in CCP, except that we replace pure nondeterminism by a probabilistic choice by means of the construct

$$\bigsqcap_{i=1}^{n} c_i \mid p_i \rightarrow A_i.$$

This makes the execution of a program similar to a random walk on the transition graph. The intended meaning of the probabilistic choice construct is as follows: First, check whether constraints c_i are entailed by the store. All those agents A_i such that c_i is entailed are called *enabled*. After that we have to *normalise* the probability distribution by considering only the enabled agents. This can be done by replacing p_i for enabled agents by a normalised transition probability:

$$\tilde{p}_i = \frac{p_i}{\sum_{\vdash c_j} p_j}$$

where the sum is over all enabled agents. This guarantees that $\sum \tilde{p}_i = 1$. Finally, one of the enabled agents is chosen according to this new probability distribution $\tilde{p}_i$.

We also replace pure nondeterminism associated with the parallel construct, $A \parallel A$, by a probabilistic version. Basically, we assume that the scheduler treats all interleavings equally and chooses any of them with the same probability.

Like all CCP languages, PCCP is defined parametrically w.r.t. to a given *constraint system*. For PCCP we do not require any substantial modification of the standard cylindric constraint systems as defined in [10, 11]. In other words, we do not intend to introduce randomness on the data, which would require to restructure the constraint system as a fuzzy or belief system. Since the only randomisation is on the programs (i.e. on the choice construct) we can safely assume a standard constraint system as underlying our language.

The operational semantics of PCCP is given by means of a probabilistic transition system as defined in Table 2. The meaning of $\longrightarrow_p$ is a single transition step which takes place with probability p. We denote the transitive closure of this transition relation by $\longrightarrow_p^*$. The label p associated to $\longrightarrow_p^*$ represents the probability of getting from one configuration to another one in several steps and is determined by the product of the probabilities associated to the single steps.

We are interested in modelling the input/output behaviour of a PCCP program. This corresponds to the classical notion of observables in CCP extended with information about the probabilities associated to each possible outcome.

Given a program P, we define the result $\mathcal{R}$ of an agent A and an initial store d as the set of all pairs $<c, p>$, where c is the least upper bound of the partial constraints accumulated during a computation starting from d; and p is the probability of reaching that result.

$$\begin{array}{rcl}\mathcal{R}(A,d) & = & \{<c,p> \mid <A,d>\longrightarrow_p^*<B,c>\not\longrightarrow\} \cup \\ & & \{<\bigsqcup_i d_i, \prod_i p_i> \mid <A,d_0>\longrightarrow_{p_0} \ldots\}.\end{array}$$

The first term describes the results of finite computations where the least upper bound of the partial store corresponds to the final store. The second term covers the infinite results.

The probability of obtaining a certain result depends on the probabilities p associated to the possible paths which lead to it. To capture the true behaviour of an agent we have to identify different computational paths leading to the same result as well as to collect

$$\textbf{R1} \quad < \textbf{tell}(c), d > \longrightarrow_1 < \textbf{stop}, c \sqcup d >$$

$$\textbf{R2} \quad < \Box_{i=1}^{n} \ c_i \mid p_i \rightarrow A_i, d > \longrightarrow_{\tilde{p_j}} < A_j, d > \qquad j \in [1, n] \ and \ d \vdash c_j$$

$$\textbf{R3} \quad \frac{< A, c > \longrightarrow_p < A', c' >}{\begin{array}{c} < A \parallel B, c > \longrightarrow_p < A' \parallel B, c' > \\ < B \parallel A, c > \longrightarrow_p < B \parallel A', c' > \end{array}}$$

$$\textbf{R4} \quad \frac{< A, d \sqcup \exists_x c > \longrightarrow_p < B, d' >}{< \exists_x^d A, c > \longrightarrow_p < \exists_x^{d'} B, c \sqcup \exists_x d' >}$$

$$\textbf{R5} \quad < p(y), c > \longrightarrow_1 < \exists_\alpha (\delta_{y\alpha} \wedge \exists_x (\delta_{\alpha x} \wedge A)), c > \quad p(x) : -A \in P$$

Table 2: The transition system for PCCP.

the accumulated probabilities associated with different interleavings. In order to do this in a precise way we define two operations. The first collects all the probabilities along different paths and in different interleavings associated to the same resulting constraint:

$$\mathcal{K}(\{< c_{ij}, p_{ij} >\}_{i,j}) \quad = \quad \{< c_i, P_{c_i} > \mid \ P_{c_i} = \textstyle\sum_j p_{ij}\}_i,$$

where c_{ij} denotes the jth occurrence of the constraint c_i in the multi-set of all results.

The second operation normalises the probabilities. This is necessary as the probabilities in each interleaving add up to one such that the overall sum of probabilities is exactly the number of possible interleavings. This process of re-normalisation effectively implies that all interleavings are equally likely.

$$\mathcal{N}(\{< c_i, p_i >\}_i) \quad = \quad \{< c_i, \tfrac{p_i}{P} > \mid \ P = \textstyle\sum_i p_i\}.$$

With these two operations we can define the observables associated to an agent A and an initial store simply as:

$$\mathcal{O}(A, d) \quad = \quad \mathcal{N}(\mathcal{K}(\mathcal{R}(A, d))).$$

3 An Operator Semantics for PCCP

We define a denotational semantics for PCCP by assigning to each agent a linear operator (matrix) on a suitable linear space. This operator encodes all the possible computational paths (traces) and will be constructed as the fixpoint of an inductively defined operator Ψ on the operator algebra on the underlying linear space.

3.1 A Linear Space on the Constraint System

Starting from the constraint system $\mathcal{C}$ we construct the free (real) vector space as the set of all formal linear combinations of constraints:

$$\mathcal{V}(\mathcal{C}) = \{\sum x_c c \mid x_c \in \mathbb{R}, \ c \in \mathcal{C}\}.$$

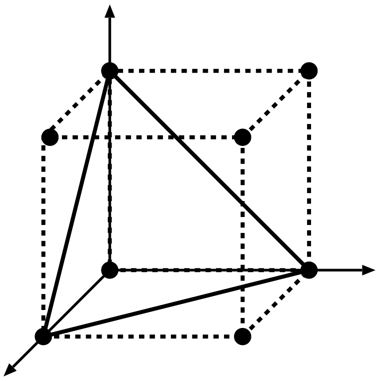

Figure 1: The embedding of computational states in $\mathcal{V}(\mathcal{C})$.

By fixing an enumeration of the constraint system, we can look at these linear combinations as tuples of real numbers. Intuitively, these vectors represents the possible states of computation. In the CCP case a state is a set of constraints (store). In PCCP a state is a distribution, i.e. a function ρ from the set of constraints to the real interval $[0, 1]$, such that

$$\sum_{c \in \mathcal{C}} \rho(c) = 1.$$

In both cases we can identify the set of possible states with a subset of the free vector space $\mathcal{V}(\mathcal{C})$. In CCP, this corresponds to the set of the vertices of the unit cube in $\mathcal{V}(\mathcal{C})$ or, more formally, to the vectors whose co-ordinates are in $\{0, 1\}$. In PCCP the set of states corresponds to the so-called simplex. Both embeddings are depicted for a simple three-element constraint system in Fig. 1.

In this work we will consider finite constraint systems. This assumption allows us to use the above defined vector space without any additional structure for both finite and infinite computation. The generalisation to infinite constraint systems requires the consideration of a suitable topology on $\mathcal{V}(\mathcal{C})$, which can be given by means of a norm. This can be done by turning $\mathcal{V}(\mathcal{C})$ into a Banach space. As the purpose of this paper is to show the applicability of an operator algebraic approach to the denotational semantics of (P)CCP, we will concentrate for the time being on the finite dimensional case. The results we present can nevertheless extended to the (countable) infinite case in a straight forward way.

3.2 The Algebra of Denotations

On the space $\mathcal{V}(\mathcal{C})$ we define linear operators which correspond to the computational steps. Since we consider finite constraint systems these operators can be represented by matrices. Each entry $m_{c,d}$ of such a matrix $M = (m_{c,d})_{c,d \in \mathcal{C}}$ encodes the possibility and/or probability to reach a certain state d starting from another state c. Concretely, we have for the CCP case the following convention:

$$m_{c,d} = \begin{cases} 1 & \text{iff } c \longrightarrow d \\ 0 & \text{otherwise} \end{cases}$$

where $c \longrightarrow d$ means that there is a transition step leading from store c to store d. This is a convenient encoding of a generator of reactive sequences [2, 3, 11].

In the case of PCCP the entries in M simply correspond to the probabilities associated to the transition step leading from store c to store d:

$$m_{c,d} = \begin{cases} p & \text{iff } c \longrightarrow_p d \\ 0 & \text{otherwise} \end{cases}$$

In this case we get a so called *stochastic matrix*, i.e. a matrix where the sum of all entries in each row is one. These matrices generate a well known class of stochastic processes, namely Markov chains [12, 13].

The matrices representing each agent form an algebra $\mathcal{L}(\mathcal{V}(\mathcal{C}))$ (in the sense of linear algebra [8]). On $\mathcal{L}(\mathcal{V}(\mathcal{C}))$ the definition of the fixpoint operator Ψ is given compositionally by means of the basic algebra operations on matrices (multiplication, sum and scalar product) and the direct sum, defined as follows:

$$M \oplus N = \begin{pmatrix} M & 0 \\ 0 & N \end{pmatrix}$$

where M and N are referred to as the *factors* of $M \oplus N$. In the following we will use the shorthand notation $M \cdot (N_1 \oplus N_2 \oplus \ldots \oplus N_n)$ for $M \cdot N_1 \oplus M \cdot N_2 \oplus \ldots \oplus M \cdot N_n$.

Furthermore, we will make use of some auxiliary operations on matrices in order to model the normalisation and compactification operations defined in Section 2. The first operation N normalises the entries in each row of a matrix so as to obtain a stochastic one:

$$N(m_{c,d}) = (\frac{m_{c,d}}{\sum_{d \in \mathcal{C}} m_{c,d}}).$$

A second operation K sums all the factors in a direct sum in order to identify transitions which may occur in different interleaving but have the same result. This compactification operation is defined by:

$$K(\bigoplus_{i=1}^{n} M_i) = \sum_{i=1}^{n} M_i.$$

3.3 The Basic Agents

Let us first look at the semantic matrices representing the primitive **tell**(c) and the guards. In order to do this we need to introduce special matrices:

$$E_{c,d} = \begin{cases} 1 & \text{at entry } c, d \\ 0 & \text{otherwise} \end{cases}$$

This represents a single transition from c to d. The set of all matrices of this type, for all constraints c and d, form an (algebraic) basis of the space of all linear operators on the vector space $\mathcal{V}(\mathcal{C})$.

The meaning of agent **tell**(c) is defined so as to encode its effect on all possible stores. When the initial store is d and we execute **tell**(c) the resulting store is given by $d \sqcup c$. This can be represented by the following matrix:

$$T(c) = \sum_{d} E_{d,d \sqcup c}.$$

One can easily see that these matrices $T(c)$ (considered for all constraints c) encode the least upper bound operation of the underlying constraint system. For any given c the matrix $T(c)$ represents a row in the table for the "$\sqcup$" operation.

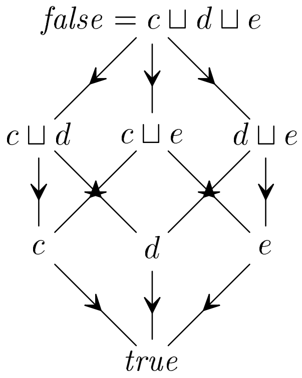

Figure 2: A constraint system.

The guard c in $c \mid p \rightarrow A$ will be represented by a diagonal matrix. This implements a projection onto the subspace of stores which entail c.

$$A(c) = \sum_{c \sqsubseteq d} E_{d,d}$$

Each guarded agent $c \mid p \rightarrow A$ in a choice construct can then be modelled by combining this projection operation with the matrix $M(A)$ representing the guarded agent A. This ensures that only those transitions described in $M(A)$ actually take place, which start from a store entailing c.

The matrices $A(c)$ encode the partial order "$\sqsubseteq$" in the constraint system. For every c the matrix $A(c)$ represents the upward closure of c.

Example 3.4 *Consider the constraint system in Fig. 2. and assume the following enumeration of the constraints* $<true, c, d, e, c \sqcup d, c \sqcup e, d \sqcup e, false>$ *. Then the matrices for* **tell**(c) *and the guard* c *are as follows:*

$$T(c) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad A(c) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

3.5 The Fixpoint Operator

We are now ready to define the fixpoint operator Ψ as in Table 3.

The **stop** agent is represented by the identity matrix

$$\mathbf{Id} = \sum_{c \in \mathcal{C}} E_{c,c}$$

The **tell** primitive was discussed before. The choice construct is modelled by a weighted sum, where the projection operator $A(c)$ selects only those transitions in $\Psi(M)(A_i)$ which are enabled and the normalisation N constructs the $\tilde{p}_i$ of the operational semantics. The

$$
\begin{array}{lcll}
\Psi(M)(\mathbf{stop}) & = & \mathbf{Id} & \\
\Psi(M)(\mathbf{tell}(c)) & = & T(c) & \\
\Psi(M)(\Box_{i=1}^{n}\ c_i \mid p_i \to A_i) & = & N(\sum_{i=1}^{n}\ p_i \cdot A(c_i) \cdot \Psi(M)(A_i)) & \\
\Psi(M)(A_1 \parallel A_2) & = & \frac{1}{2}\left(\Psi(M)(A_1) \cdot \Psi(M)(A_2) \oplus \Psi(M)(A_2) \cdot \Psi(M)(A_1)\right) & \\
\Psi(M)(\exists_x A) & = & \exists\!\!\exists_x \cdot \Psi(M)(A) & \\
\Psi(M)(p(x)) & = & M(\Delta_y^x A) & p(x) : -A \in P
\end{array}
$$

Table 3: The compositional definition of Ψ.

meaning of the parallel construct is given by means of a direct sum: different interleavings are represented by different factors in the direct sum. Hiding is modelled by means of a shift operation expressed by the matrix:

$$\exists\!\!\exists_x = \sum_{c \in \mathcal{C}} E_{c, \exists_x c}.$$

Finally, the matrix for the procedure call is the matrix associated to the agent defining it in the program (after renaming). $\Delta_y^x A$ is a shorthand notation for $\exists_\alpha(\delta_{y\alpha} \wedge \exists_x(\delta_{\alpha x} \wedge A))$.

Note that the binary parallel construct defined above is not associative. To overcome this we could replace it by an n-ary version whose meaning would be as follows:

$$\Psi(M)(\|_{i=1}^{n}\ A_i) = \frac{1}{n!} \bigoplus_{\sigma \in S(n)} \prod_{j=1}^{n} \Psi(M)(A_{\sigma(j)}),$$

where $S(n)$ is the set of all permutations of n indices.

Note also that the matrix multiplication can be used for modelling an extension of the language with a sequential combination of agents.

Proposition 3.6 *The operator Ψ is continuous on $\mathcal{L}(\mathcal{V}(\mathcal{C}))$ for finite $\mathcal{C}$.*

In order to construct a fixpoint of Ψ we will mimic the classical fixpoint construction: Starting with the initial matrix $M_0(A)$ assigning to each agent A the identity **Id** we iteratively apply Ψ. The (pointwise) limit $Fix(\Psi)$ of the sequence

$$
\begin{array}{rcl}
M_0(A) & = & \mathbf{Id} \\
M_i(A) & = & \Psi(M_{i-1})(A)
\end{array}
$$

will be used for constructing our model. First of all, the convergence is guaranteed by a generalisation of results already shown in [6]. The intuition behind the proof of convergence is that the application of Ψ can only create more precise information about the final result. The limit of this sequence is a fixpoint of Ψ because of Proposition 3.6.

The observables $\mathcal{O}$ defined in Section 2 can be retrieved from the model $Fix(\Psi)$ by means of the operations N and K introduced in Section 3.2. This is stated by the following theorem:

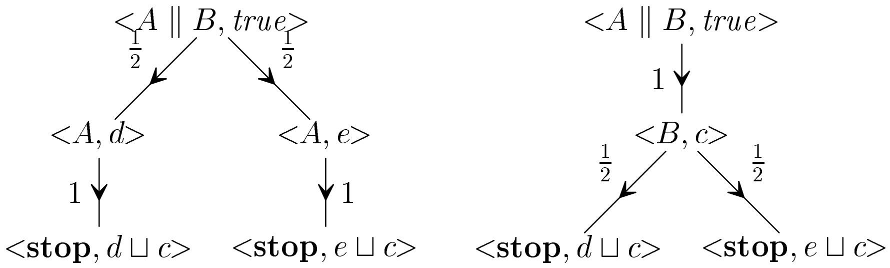

Figure 3: Two unguarded interleavings.

Theorem 3.7 *For any agent A and initial store d:*

$$N(K(Fix(\Psi)(A)))(d) = \mathcal{O}(A,d) = \mathcal{N}(\mathcal{K}(\mathcal{R}(A,d))).$$

On the left hand side we have the vector obtained by multiplication of the matrix $N(K(Fix(\Psi)(A)))$ with the vector $d \in \mathcal{V}(\mathcal{C})$. On the right hand side the set of pairs $\{<c_i, p_i>\}_i$ defining the observables can also be seen as a vector $\sum_i p_i c_i$ in $\mathcal{V}(\mathcal{C})$. By abuse of notation, d on the left hand side denotes a vector, whereas on the right hand side represents a constraint in $\mathcal{C}$.

4 Examples

In this section we give some examples of PCCP programs and show the construction of their Markov models as well as the correspondence with the observables.

Example 4.1 *Consider the constraint system of Example 3.4. Let us first introduce the following shorthand notation for two agents A and B, which implements an unguarded choice:*

$$\begin{array}{rcl} A & \equiv & \mathbf{tell}(c) \\ B & \equiv & true \mid \frac{1}{2} \rightarrow \mathbf{tell}(d) \\ & & \Box\ true \mid \frac{1}{2} \rightarrow \mathbf{tell}(e). \end{array}$$

We are interested in the parallel execution of these two agents, $A \parallel B$. We obtain two possible interleavings described by the transition trees in Fig. 3. The observables for $A \parallel B$ are therefore:

$$\begin{array}{rcl} \mathcal{O}(A \parallel B, true) & = & \mathcal{N}(\mathcal{K}(\{<d \sqcup c, \frac{1}{2}>, <e \sqcup c, \frac{1}{2}>, <d \sqcup c, \frac{1}{2}>, <e \sqcup c, \frac{1}{2}>\})) \\ & = & \mathcal{N}(\{<d \sqcup c, 1>, <e \sqcup c, 1>\}) \\ & = & \{<d \sqcup c, \frac{1}{2}>, <e \sqcup c, \frac{1}{2}>\} \end{array}$$

We now construct the denotational semantics of this example.

$$\begin{array}{rcl} M(A) & = & M(\mathbf{tell}(c)) \\ M(B) & = & N(\frac{1}{2}\, M(\mathbf{tell}(d)) \ + \ \frac{1}{2}\, M(\mathbf{tell}(e))), \end{array}$$

$$
\begin{tikzcd}
{<A \parallel B, \mathit{true}>} \arrow[d, "1"'] \\
{<A, d>} \arrow[d, "1"'] \\
{<\mathbf{stop}, d \sqcup c>}
\end{tikzcd}
\qquad
\begin{tikzcd}
& {<A \parallel B, \mathit{true}>} \arrow[d, "1"'] & \\
& {<B, c>} \arrow[dl, "\frac{1}{2}"'] \arrow[dr, "\frac{1}{2}"] & \\
{<\mathbf{stop}, d \sqcup c>} & & {<\mathbf{stop}, e \sqcup c>}
\end{tikzcd}
$$

Figure 4: Two guarded interleavings.

therefore the resulting matrix $M(A \parallel B)$ is:

$$
\begin{pmatrix}
0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} \\
0 & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{4} \\
0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} \\
0 & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{4} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}
\end{pmatrix}
\oplus
\begin{pmatrix}
0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} \\
0 & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{4} \\
0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} \\
0 & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{4} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}
\end{pmatrix}
$$

The observables can be recovered from $M(A \parallel B)$ by considering the first row in the matrix:

$$
N(K(M(A \parallel B))) =
\begin{pmatrix}
0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \\
0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\
0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \\
0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
$$

Example 4.2 *Consider now a guarded version of the above example.*

$$
\begin{array}{rcl}
A & \equiv & \mathbf{tell}(c) \\
B & \equiv & \mathit{true} \mid \frac{1}{2} \ \rightarrow\ \mathbf{tell}(d) \\
& & \square\ c \mid \frac{1}{2} \ \rightarrow\ \mathbf{tell}(e).
\end{array}
$$

Operationally we still have two interleavings although the first one differs from the one above due to the presence of the guard c, (cf. Fig. 4). This is reflected in the observables:

$$
\begin{array}{rcl}
\mathcal{O}(A \parallel B, \mathit{true}) & = & \mathcal{N}(\mathcal{K}(\{<d \sqcup c, 1>, <d \sqcup c, \frac{1}{2}>, <e \sqcup c, \frac{1}{2}>\})) \\
& = & \mathcal{N}(\{<d \sqcup c, 1\frac{1}{2}>, <e \sqcup c, \frac{1}{2}>\}) \\
& = & \{<d \sqcup c, \frac{3}{4}>, <e \sqcup c, \frac{1}{4}>\}.
\end{array}
$$

The semantics of $A \parallel B$ also reflects this difference:

$$
\begin{array}{rcl}
M(A) & = & M(\mathbf{tell}(c)) \\
M(B) & = & N(\frac{1}{2}\ M(\mathbf{tell}(d))\ +\ \frac{1}{2}\ A(c) \cdot M(\mathbf{tell}(e))),
\end{array}
$$

which gives the matrix $M(A \parallel B)$:

$$
\begin{pmatrix}
0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} \\
0 & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{4} \\
0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} \\
0 & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{4} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}
\end{pmatrix}
\oplus
\begin{pmatrix}
0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\
0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} \\
0 & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{4} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}
\end{pmatrix}
$$

Again, the observables can be reconstructed by considering the first row in the following matrix:

$$
N(K(M(A \parallel B))) =
\begin{pmatrix}
0 & 0 & 0 & 0 & \frac{3}{4} & \frac{1}{4} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{3}{4} & 0 & 0 & \frac{1}{4} \\
0 & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & \frac{3}{4} \\
0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \\
0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
$$

Example 4.3 *In this example we show how our model deals with infinite computations.*

The following declarations have been used in [1] (in their CCP formulation) as an example of the inapplicability of metric and order-theoretic approaches to modelling the exact results observables in constraint programming.

$$
\begin{array}{lcl}
p(x) & :- & q(x) \\
q(x) & :- & true|\frac{1}{2} \rightarrow p(x) \\
 & & \Box\, true|\frac{1}{2} \rightarrow (r(x) \parallel \mathbf{tell}(c)) \\
r(x) & :- & \mathbf{tell}(\mathit{false}).
\end{array}
$$

Assume a constrain system with $true \sqsubseteq c \sqsubseteq false$.

The observables of p, q and r are:

$$\mathcal{O}(p(x), true) = \mathcal{O}(q(x), true) = \mathcal{O}(r(x), true) = \{<\mathit{false}, 1>\}.$$

The primitives we need to construct the Markov model are:

$$
M(\mathbf{tell}(c)) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad M(\mathbf{tell}(\mathit{false})) = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}.
$$

To simplify our presentation we replace the direct sum (used to model the parallel construct) by a sum. This is possible because we are dealing here with an unguarded version of PCCP. The sequence of the semantical matrices obtained by the iterative application of Ψ is as follows:

	$p(x)$	$q(x)$	$r(x)$
M_0	**Id**	**Id**	**Id**
M_1	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$
M_2	$\begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$
M_3	$\begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \frac{1}{4} & 0 & \frac{3}{4} \\ 0 & \frac{1}{4} & \frac{3}{4} \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
M_∞	$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$

We can now reconstruct the observables from M_∞. As we did the compactification during the construction of the chain, the observables correspond directly to M_∞.

5 Conclusions

We introduced a denotational semantics for (P)CCP which is based on some methods unconventional in computer science. We exploited the existing theory of stochastic processes, and in particular Markov chains, for the definition of the domain of denotations. This offers a powerful and elegant way for modelling synchronisation.

In this paper we restricted the treatment to the case of a finite (or a finite effectively used portion of the) constraint system, as this simplifies the presentation. However, all the results can be generalised in a relatively straightforward way to the case of infinite constraint systems. This generalisation will basically consists in replacing matrices by more general, infinite dimensional operator algebras. Moreover, in order to deal with the convergence of the fixpoint construction, an appropriate topology has to be introduced on the underlying vector space of constraints and the operator algebra itself. We plan to develop this generalisation more formally in future work.

We finally presented some examples where we explicitly calculated the matrix semantics. This calculation has been done with the help of a high-level interactive language for numerical computations, called Octave [7]. In this way we successfully explored the correspondence between the denotational and the operational semantics, formally stated in Section 3.5 by Theorem 3.7.

References

[1] F. S. de Boer, A. Di Pierro, and C. Palamidessi. Nondeterminism and Infinite Computations in Constraint Programming. *Theoretical Computer Science*, 151(1), 1995. Selected Papers of the Workshop on Topology and Completion in Semantics, Chartres, France.

[2] F. S. de Boer and M. Gabbrielli. Infinite Computations in Concurrent Constraint Programming. *Electronic Notes in Theoretical Computer Science*, 6:16, 1997.

[3] F.S. de Boer and C. Palamidessi. A Fully Abstract Model for Concurrent Constraint Programming. In S. Abramsky and T.S.E. Maibaum, editors, *TAPSOFT/CAAP*, volume 493, pages 293–319. Springer Verlag, 1991.

[4] A. Di Pierro and H. Wiklicky. On probabilistic CCP. In A. Policriti, M. Falaschi, and M. Navarro, editors, *Proceedings of APPIA-GULP-PRODE'97 Joint Conference on Declarative Programming*, pages 225–234, June 1997.

[5] A. Di Pierro and H. Wiklicky. An operational semantics for Probabilistic Concurrent Constraint Programming. In P. Iyer, Y. Choo, and D. Schmidt, editors, *ICCL'98 – International Conference on Computer Languages*, pages 174–183. IEEE Computer Society and ACM SIGPLAN, IEEE Computer Society Press, May 1998.

[6] A. Di Pierro and H. Wiklicky. Probabilistic Concurrent Constraint Programming: Towards a Fully Abstract Model. In Gruska Brim and Zlatuska, editors, *Proc. of the 23rd International Symposium on Mathematical Foundations of Computer Science, MFCS'98*, Lecture Notes in Computer Science, page to appear, Brno, Czech Republic, 1998. Springer-Verlag.

[7] John W. Eaton. *Octave*. Boston, MA, 1997.

[8] Thomas W. Hungerford. *Algebra*, volume 73 of *Graduate Texts in Mathematics*. Springer Verlag, New York – Berlin – Heidelberg, 1980.

[9] S. O. Nyström and B. Jonsson. Indeterminate Concurrent Constraint Programming: A Fixpoint Semantics for Non-Terminating Computations. In D. Miller, editor, *Proc. of the 1993 International Logic Programming Symposium*, Series on Logic Programming, pages 335–352. The MIT Press, 1993.

[10] V. A. Saraswat and M. Rinard. Concurrent constraint programming. In *Proceedings of POPL*, pages 232–245. ACM, 1990.

[11] V. A. Saraswat, M. Rinard, and P. Panangaden. Semantics foundations of concurrent constraint programming. In *Proceedings of POPL*, pages 333–353. ACM, 1991.

[12] Eugene Seneta. *Non-negative Matrices and Markov Chains*. Springer Verlag, New York – Heidelberg – Berlin, second edition, 1981.

[13] Henk C. Tijms. *Stochastic Models – An Algorithmic Approach*. John Wiley & Sons, Chichester, 1994.