A Banach Space Based Semantics for Probabilistic Concurrent Constraint Programming

Alessandra di Pierro and Herbert Wiklicky

City University London
Northampton Square, London EC1V OHB
{adp,herbert}@cs.city.ac.uk

Abstract. The aim of this work is to provide a probabilistic foundation for nondeterministic computations in Concurrent Constraint Programming (CCP).
Differently from common approaches in (imperative) high-level programming languages, which rely on some kind of `random()` function, we introduce randomness in the very definition of the language by means of a probabilistic choice construct. This allows a program to make stochastic moves during its execution. We call the resulting language Probabilistic Concurrent Constraint Programming (PCCP).
We present an operational and denotational semantics for PCCP both capturing a notion of observables which combines results of computations and the probability of those results being computed. For the denotational semantics, we consider linear (Banach) structures and the mathematics behind them, as appropriate candidates for the definition of the domain of denotations. This approach differs from the ones typically adopted in the semantics of constraint programming, which are based on order-theoretic or metric structures.
We compare this semantics to other approaches and discuss it in some examples.

1 Introduction

This paper presents a new probabilistic programming language, which results from the introduction of a notion of randomness in the Concurrent Constraint Programming (CCP) paradigm [27]. As a probabilistic extension of CCP, we call the new language Probabilistic CCP (PCCP).

The use of a random element (a "coin flipping" device), in an algorithm is nothing new: such algorithms for calculating π or more generally the integration of complicated functions have been known in mathematics for a long time (e.g. Monte Carlo algorithms). Algorithms where the flow of information is determined by an element of random choice have recently gained more attention in particular from a complexity theoretic point of view. Various randomised algorithms and procedures have been investigated, of which we mention just a few examples: simulated annealing in combinatorial optimisation [1], genetic algorithms [7], probabilistic primality tests for use in crypto-systems [21], randomised proof procedures (e.g. for linear logic [17]), and approximation algorithms [11].

In our approach we incorporate this element of randomness in the form of a *probabilistic choice* directly within the semantics. This is different from other approaches where randomness is introduced either in an ad-hoc way by referring some "external" function or procedure call (like `random()` in imperative languages such as C), or by means of random variables [15, 10].

We define an operational semantics for PCCP by using the inductive style of Plotkin's SOS, i.e. by means of a transition system which describes the evolution of a program in a structural way [23]. Randomness is expressed by labels representing the probability that a transition takes place. Based on this we define a notion of observables which captures the (finite) results of computation together with their probability of being computed.

We then introduce a denotational semantics by assigning to each program a distribution (or a measure), i.e. a function from (the Borel sets of) the constraint system to the real interval $[0, 1]$. We will then use the Brouwer Fixed-Point Theorem and its infinite dimensional generalisation, the Schauder-Tychonoff Theorem to define a fixed-point semantic for PCCP. This semantics clearly reflects our observables. We also show by discussing two examples that this semantics lends itself to be extended so as to model infinite computation.

Since our intent is to randomise programs rather than data, we do not need to base our language on any kind of fuzzy or belief system [14]. Therefore, it is sufficient for our purpose to maintain the same standard (cylindric) constraint system as in CCP.

Up to now probabilistic algorithms have been usually written using imperative languages extended with some random facility (e.g. pseudo-random number generators) and have been analysed largely only by ad-hoc methods. However, real problems are normally quite complicated and intuition is very likely to fail. Therefore, a formal deductive system would be much more handy in such cases. The new language we propose and the semantics we provide is intended to constitute a general semantical framework in which "randomised algorithms" and in general probabilistic programs can be formalised, such that intuition can be replaced by rigorous mathematical tools for analysing and reasoning about such programs.

2 Nondeterminism

Nondeterminism is inherent in concurrent systems whose components work asynchronously. Therefore it cannot be easily ignored when studying models of concurrent and distributed systems. The difficulty in handling this notion lies basically in that pure nondeterminism is not in itself a computational concept: it just indicates all possible transitions in a machine, a deductive system, etc. but says nothing on how to make an actual choice among the various alternatives.

Of course in any physical realisation of a nondeterministic machine one has to deal with the problem of actually making choices. As a consequence, a "realistic" interpretation of a nondeterministic system is usually that nondeterministic choices, though beyond the control of the system itself, are supposed to be resolved by a scheduler. The scheduler is traditionally thought of as a fictitious entity called "adversary", for it is supposed to make the worst nondeterministic decisions (as a worst case context is normally considered in the study and analysis of algorithms).

Alternatively, the problem of nondeterminism can be approached as the absence of information which needs to be specified for the system be able to continue its computation. In this case the solution is to "refine" the notion of nondeterminism so as that some additional requirement on how the choice has to be made is available together with the possibility itself to make a choice.

One example of such a "refinement" is the notion of a *fair choice* where it is required that all the possible choices are actually made sometime. Note that this only specifies something (additional) about the computation we might expect to observe, but does not change the (number of) possibilities of how to continue with that computation.

Another example of providing more information of how possible branches of computation will be realised is the notion of a *probabilistic choice*. Here we put some requirement on the frequency of choices. We specify how likely it is (repeating the same computation 'sufficiently' often) that a certain possible continuation of the computation is actually performed.

By restricting the attention to CCP, the traditional distinction is between *don't care* nondeterminism (aka *indeterminism*) and *don't know* nondeterminism (aka *angelic* nondeterminism). Both forms of nondeterminism are expressed in terms of guarded constructs of the form $g \to A$, where, roughly speaking, the guard g represents a condition and A a computational process or agent. A choice is made among the agents whose associated guards are verified, and the other agents are discarded. However, in the indeterministic version once a choice is made the system has to commit with that choice and no other alternatives will be considered. In the angelic case it is instead possible to redo the choice (by backtracking) so that only those agents which lead to a failure are discarded.

In this paper we investigate the possibility to introduce a probabilistic nondeterminism in CCP. This notion is orthogonal to the ones discussed above and, as such, can be combined with both *don't care* and *don't know* nondeterminism.

In the next section we explain the idea behind the concept of a probabilistic choice by discussing an example well-known in probability theory, namely the *random walk* [8, 9].

3 Probabilistic Choice

To illustrate the difference between pure nondeterminism and our approach to a probabilistic version of nondeterminism let us look at a (directed or non-directed) graph $\Gamma = (N, E)$. We can try to find (nondeterministically) some paths trough Γ leading from one node to another. This just allows us to distinguish between nodes where a path between them exists and nodes which are not *connected*.

The only information we can get this way are thus related to concepts like connectedness, reachability, etc.

More information can be obtained by assigning to each of the edges of Γ a probability $p(v)$ of choosing it (in which case it is necessary to require that the sum of all probabilities assigned to the edges leaving one node is one). A so-called *random walk* on Γ then can be described as follows: We start at an initial node and (according to the assigned probabilities) choose one adjacent node to go to. From there we repeat this procedure. We obtain this way some (random) paths through the graph. In this setting we might not only ask if two nodes are connected, i.e. if a path exists, but also how likely it is to find a path from one node to another.

A very simple example of such a process is obtained by considering a random walk on a graph where each node corresponds to an integer, and two nodes are connected if they differ only by ± 1. If we further assume that the probability to go from one integer to its successor or to its predecessor is each 50% we get a process usually referred to as a *symmetric random walk* in one dimension, which perhaps resembles a drunken sailor's way staggering along a narrow street.

We can calculate the probability p of how likely it is that after one step, or two steps, etc. we will be at position n in this graph. Let us express this by an ordered pair $<n,p>$. Suppose we start at the origin ± 0, that is the initial situation is given by $<\pm 0, 1>$: with certainty we are at the origin. After one step, with probability $\frac{1}{2}$ we will be either at position $+1$ or position -1. If we continue this walk we get the following picture (where impossible positions, i.e. with $p = 0$ are omitted):

$$\begin{array}{c}
\{<\pm 0, 1>\} \\
\{<-1, \frac{1}{2}>, <+1, \frac{1}{2}>\} \\
\{<-2, \frac{1}{4}>, <\pm 0, \frac{1}{2}>, <+2, \frac{1}{4}>\} \\
\{<-3, \frac{1}{8}>, <-1, \frac{3}{8}>, <+1, \frac{3}{8}>, <+3, \frac{1}{8}>\} \\
\{<-4, \frac{1}{16}>, <-2, \frac{4}{16}>, <\pm 0, \frac{6}{16}>, <+2, \frac{4}{16}>, <+4, \frac{1}{16}>\} \\
\{<-5, \frac{1}{32}>, <-3, \frac{5}{32}>, <-1, \frac{10}{32}>, <+1, \frac{10}{32}>, <+3, \frac{5}{32}>, <-5, \frac{1}{32}>\} \\
\{<-6, \frac{1}{64}>, <-4, \frac{6}{64}>, <-2, \frac{15}{64}>, <\pm 0, \frac{20}{64}>, <+2, \frac{15}{64}>, <+4, \frac{6}{64}>, <+4, \frac{1}{64}>\} \\
\ldots
\end{array}$$

In principle this is just Pascal's triangle where additionally going a step further means also that we have to halve all the probabilities involved.

It is now possible to investigate further properties of this simple random walk [8, 9]. For example, we could ask how likely it is to return to given point, e.g. the origin. The probability of returning some time to the origin is the probability of returning after one step, plus the probability of returning after two steps, etc. It is a well-known result that the recurrence probability is one. That means that although there exist paths which will never bring us back their probability vanishes and it is *practically certain* to return to the starting point. Note that in a purely nondeterministic setting we could only conclude that *it is possible* to return to the starting point.

4 Probabilistic CCP

In order to allow the formulation of various randomised algorithms – some of which we mentioned in the above introduction – within the declarative framework of constraint programming we introduced a new language, which we called Probabilistic Concurrent Constraint Programming (PCCP).

4.1 Syntax and Intuitive Semantics

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 the nondeterministic choice construct by a probabilistic one of the form:

$$\mathop{\Box}_{i=1}^{n} c_i | p_i \rightarrow A_i .$$

$P ::= D.A$

$D ::= \epsilon \mid D.D \mid p(x) : -A$

$A ::= \mathbf{stop} \mid \mathbf{tell}(c) \mid \Box_{i=1}^{n}\ c_i|p_i \rightarrow A_i \mid A \parallel A \mid \exists_x A \mid p(x)$

Table 1. The syntax for PCCP.

The intended meaning of this construct is as follows: First, check whether the constraints c_i are entailed by the store. All those agents A_i such that c_i are 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:

$$\tilde{p_i} = \frac{p_i}{\sum_j p_j}$$

where the sum is over all enabled agents. This way we can guarantee $\sum_i \tilde{p_i} = 1$.

Finally, one of the enabled agents is chosen according to this new probability distribution $\tilde{p_i}$.

Note that *deterministic* agents can be obtained by imposing $n = 1$. This means that only one agent is enabled and therefore the normalised probability has to be 1.

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 [26, 27]. However, as in [5], some additional assumptions could be made on the constraint system in order to deal with the probabilistic features of our language more explicitly.

4.2 Operational Semantics

In [5] we define the operational semantics in the style of SOS [23], i.e. by means of a (probabilistic) transition system which describes the evolution of the system of concurrent agents. A *configuration* represents the state of the system at a certain moment, namely the agent A which has still to be executed, and the common store d. We denote a configuration by $<A, d>$.

In order to express the probability of each transition step we introduce a *labelled transition system*, which consists of a pair $(\mathit{Conf}, \longrightarrow_p)$, where *Conf* is a set of *configurations* and $\longrightarrow_p \subseteq \mathit{Conf} \times \mathbb{R} \times \mathit{Conf}$ is the transition relation defined by the rules in Table 2. The label p represents the transition probability between two configurations.

We denote the transitive closure of $\longrightarrow_p$ by $\longrightarrow_p^*$. As we deal with a labelled transition system, we also have to define the label p associated to $\longrightarrow_p^*$. It is obvious that the probability of getting from one configuration to another one in several steps is determined by the product of the probabilities associated to the single steps. In case we have several possible paths from one configuration to another one we have to sum the probabilities along them to obtain the correct probability [5].

The agent **stop** represents successful termination. The basic actions are given by 'telling' the common store a constraint and by 'asking' if a constraint is entailed by the store. The first action is expressed by the **tell**(c) construct. Given a store d, as shown by rule **R1**, the execution of **tell**(c) updates the store to $c \sqcup d$ and then stops. This transition, as it is the only possible one, is done with probability 1.

The ask action is incorporated in the choice construct in the form $c|p \rightarrow A$. In case the number of alternatives is just one, the choice construct corresponds to the **ask**$(c) \rightarrow A$ of CCP. The effect of the probabilistic choice in general was described informally in the previous section. Rule **R2** gives a formal description. One of the agents A_j is enabled if the store entails c_j . After normalising the probabilities to

$$\tilde{p_j} = \frac{p_j}{\sum_{\vdash c_i} p_i}$$

as described before, a probabilistic choice (according to $\tilde{p_j}$) is made among the enabled agents. This means that if this choice is repeated (under the same condition and sufficiently often) the relative frequency of executions of an agent A_j is exactly $\tilde{p_j}$.

Rules **R3** and **R4** describe the usual semantics of parallel execution of agents and the hiding operation. The former is modelled as interleaving. As for the latter we use the notation $\exists_x^d A$ for the agent A with local store d containing information on x which is hidden in the external store (see [4, 26] for further details). Obviously, the transition probability p is not changed by parallel composition and hiding.

The execution of a procedure call, $p(y)$, is modelled by the recursion rule **R5** which replaces $p(y)$ by the body of its definition in the program P, after the link between the actual parameter y and the formal parameter x has been established. Following the method introduced in [27], we express this link between formal and actual parameters by the context $\exists_\alpha(\delta_{y\alpha} \wedge \exists_x(\delta_{\alpha x} \wedge \ldots))$, where α is a variable which does not occur free in any program P or goal G (i.e. $\exists_\alpha c = c$ for all constraints c occurring in P or G), and α does not occur as a parameter (either as a formal parameter or as an actual parameter). Note that throughout the whole computation only one variable α is needed. As in rule **R1** the associated transition probability is 1.

4.3 The Observables

The *observables* we consider correspond to the input/output behaviour of an agent. Given a program P, we define the observables $\mathcal{O}_P$ 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. For finite computations this corresponds to the set:

$$\mathcal{O}_P(A, d) = \{<c, p> \mid <A, d> \longrightarrow_p^* <\mathbf{stop}, c>\}.$$

Note that this notion of observables differs from the classical notion of input/output behaviour in CCP. In the classical case a constraint c belongs to the input/output observables of a given agent A if at least one path leads from the initial configuration d to the final result c. In the probabilistic case we have to consider *all* possible paths leading to the same result c and combine the associated probabilities.

5 Denotational Semantics

Let us now introduce a denotational semantics for PCCP. To do this we have to assign to each agent $A \in \mathcal{A}$ uniquely a semantics $[\![A]\!]$, i.e. a mathematical object in some (structured) space. Furthermore, the definition of this semantics should be not only well-defined but also compositional. Finally, we are obviously also interested in a way to explicitly construct the semantics for each agent in PCCP.

Basically we will define the meaning or semantics of an agent as the set of its possible results plus some information about the likelihood of actually obtaining each result. By a result of an agent we roughly mean the constraint representing the final store after executing the agent. The information about the likelihood of a result will be formally expressed by assigning to each PCCP agent a (probability) measure on the constraint system $\mathcal{C}$. In our case we will

R1 $<\mathbf{tell}(c), d> \longrightarrow_1 <\mathbf{stop}, c \sqcup d>$	
R2 $<\square_{i=1}^{n}\ c_i \mid p_i \rightarrow A_i, d> \longrightarrow_{\tilde{p_j}} <A_j, d>$	$j \in [1, n]$ *and* $d \vdash c_j$
R3 $\dfrac{<A, c> \longrightarrow_p <A', c'>}{\begin{array}{l}<A \parallel B, c> \longrightarrow_p <A' \parallel B, c'> \\ <B \parallel A, c> \longrightarrow_p <B \parallel A', c'>\end{array}}$	
R4 $\dfrac{<A, d \sqcup \exists_x c> \longrightarrow_p <B, d'>}{<\exists_x^d A, c> \longrightarrow_p <\exists_x^{d'} B, c \sqcup \exists_x d'>}$	
R5 $<p(y), c> \longrightarrow_1 <\exists_\alpha(\delta_{y\alpha} \wedge \exists_x(\delta_{\alpha x} \wedge A)), c>$	$p(x) : -A \in P$

Table 2. The transition system for PCCP.

represent this measure by a distribution (function) on the constraint system, that is a function from the set of constraints $\mathcal{C}$ into the real interval $[0, 1]$ (see [25, 8]).

Our approach generalises the approach in [4], as – in order to deal with recursion – we define a similar operator Φ which transforms interpretations into interpretations. By constructing the appropriate limit of a sequence of interpretations, i.e. a fixed point of Φ, we are then able to define the semantics of an agent. The basic difference between our approach and the one in [4] is that in our case interpretations are not just sets of constraints but instead we consider distributions.

5.1 Interpretations of Agents

An interpretation $I(A)$ of an agent A is a function from the set of constraints $\mathcal{C}$ into the real interval between zero and one such that the sum of all values assigned to all constraints adds up to one. The intention is, of course, that these numbers represent the probability associated to each constraint. More formally, an *interpretation* $I(A)$ of an agent A is a map from the set of constraints $\mathcal{C}$ into the real numbers $I(A) : \mathcal{C} \mapsto [0, 1]$ satisfying the following "normalisation" condition:

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

One way of concretely denoting the distribution function $I(A)$ of an agent A is via a collection of pairs of the form: $\{<\text{constraint}, \text{probability}>\}$, i.e. we have the following notation to represent an interpretation of an agent:

$$I(A) = \{<c_i, p_i>\} \text{ with } c_i \in \mathcal{C} \text{ and } p_i \in [0, 1] \text{ such that } \sum_i p_i = 1.$$

In the following, whenever we will denote interpretations in this way, we will usually omit those pairs where the probability vanishes. Note that in the case of an uncountable constraint system it would be appropriate to change this notation slightly, replacing for instance sums by integrals, etc.

The set of interpretations $\mathcal{I}(A)$ of each agent forms a subset of a *vector space*. The multiplication by a constant and the addition of distributions are defined *pointwise* simply as (with $q, p \in \mathbb{R}$):

$$qD = q\left\{<c_i, p_i>\right\}_i = \left\{<c_i, qp_i>\right\}_i$$

and

$$D + E = \left\{<c_i, p_i>\right\}_i + \left\{<c_i, q_i>\right\}_i = \left\{<c_i, p_i + q_i>\right\}_i .$$

Additionally, in order to deal with hiding, we also define a further operator on distributions as follows:

$$\exists_x D = \exists_x \left\{<c_i, p_i>\right\}_i = \left\{<\exists_x c_i, p_i> \mid <c_i, p_i> \in D\right\}$$

A (complete) interpretation $I : \mathcal{A} \mapsto \mathcal{I}$ for all agents is a function from the set of agents $\mathcal{A}$ into the set of *distributions* $\mathcal{I}$ on the constraint set $\mathcal{C}$, assigning to each agent $A \in \mathcal{A}$ a concrete interpretation $I(A) \in \mathcal{I}$.

5.2 The Fixpoint Operator

We treat recursion by constructing the meaning, i.e. the denotational semantics $[\![A]\!]$, of an agent A – in particular of a procedure – as the fixed-point of an approximation operator Φ. One way to concretely construct this fixed-point will be as the (pointwise) limit of a sequence of interpretations corresponding to the stepwise execution of the agent, starting from store *true*; but we will also discuss other possible approaches.

To start, we first *compositionally* define an operator Φ on the interpretations $\mathcal{I}$ as in Table 3. We restrict for the time being to the fragment of PCCP where all guards in a choice construct are *true*, that is where synchronisation is not present. Due to the latter, the definition of Φ for full PCCP is more complicated and we postpone it to a future work.

The definition of Φ is quite straight forward. The most interesting clause perhaps is the third one, which expresses the (convex) combinations of interpretations when a probabilistic choice is made. For a choice between two agents A and B whose associated constraints are both *true* this clause simplifies to

$$\Phi(I)(p \to A \;\Box\; q \to B) = p\Phi(I)(A) + q\Phi(I)(B).$$

The denotational semantics of an agent A is now defined as a fixed point of this operator $\Phi : \mathcal{I} \mapsto \mathcal{I}$. We have, therefore, to show that Φ indeed has fixed points.

$\Phi(I)(\mathbf{stop})$	$= \{<true, 1>\}$
$\Phi(I)(\mathbf{tell}(c))$	$= \{<c, 1>\}$
$\Phi(I)(\Box_{i=1}^{n} \ c_i \mid p_i \rightarrow A_i)$	$= \sum_{i=1}^{n} p_i \Phi(I)(A_i)$
$\Phi(I)(A \parallel B)$	$= \{<c \sqcup d, p> \mid <c, p> \in \Phi(I)(A), <d, p> \in \Phi(I)(B)\}$
$\Phi(I)(\exists_x A)$	$= \exists\!\!\exists_x \Phi(I)(A)$
$\Phi(I)(p(x))$	$= I(\exists_\alpha(\delta_{y\alpha} \wedge \exists_x(\delta_{\alpha x} \wedge A)))$

Table 3. The compositional definition of $\Phi : \mathcal{I} \rightarrow \mathcal{I}$.

5.3 The Finite Case

Let us assume first that the set of agents $\mathcal{A}$ and the set of constraints $\mathcal{C}$ are both *finite*, in particular let the number of agents be $|\mathcal{A}| = n$ and the number of constraints be $|\mathcal{C}| = m$. We later will generalise our setting to the case of an infinite number of constraints, but for the time being this assumption will simplify our presentation.

For finite constraint sets each function $I(A)$ can be simply represented by a vector in $\mathbb{R}^m$, i.e. an m-tuple of real numbers:

$$(p_1, p_2, \dots, p_m)$$

such that for any enumeration of constraints $\mathcal{C} = \{c_1, c_2, \dots, c_m\}$ we get

$$\begin{array}{rcl} p_1 & = & I(A)(c_1) \\ p_2 & = & I(A)(c_2) \\ & \dots & \\ p_m & = & I(A)(c_m) \end{array}$$

As already mentionned, interpretations of an agents form a sub-space of a vector space, indeed they form sub-set of a very special type:

Lemma 1. *For each agent A the set of possible interpretations*

$$\mathcal{I}(A) = \{I(A) = (I(A)(c_1), I(A)(c_2), \dots, I(A)(c_m))\}$$

is a (non-empty) convex compact subset of $\mathbb{R}^m$.

To be even more specific the set of all possible interpretations $\mathcal{I}(A)$ of an agent A forms a so called *simplex* in $\mathbb{R}^n$. Although it is impossible to picture a general simplex, for a system with only three constraints it is quite instructive to

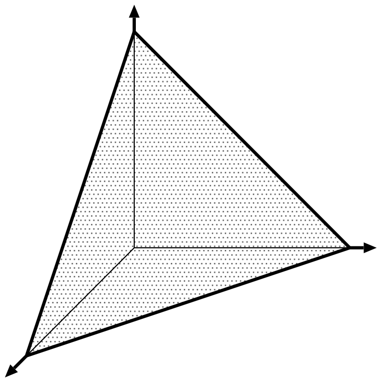

Fig. 1. A geometric representation of interpretations.

draw a picture as in Fig. 1. The set of all vectors fulfilling the above normalisation condition, i.e. $\mathcal{I}(A)$, simply form a triangle.

We can combine the interpretations $I(A_i)$ of all agents $\{A_1, A_2, \ldots, A_n\}$ and represent any interpretation as an $n \times m$-tuple in $\mathbb{R}^{n\times m}$. With an interpretation of an agent A_i as

$$I(A_i) = (p^i_1, p^i_2, \ldots, p^i_m)$$

an element in this set is concretely in the following form:

$$(p^1_1, p^1_2, \ldots, p^1_m, p^2_1, p^2_2, \ldots, p^2_m. \ldots, p^n_1, p^n_2, \ldots, p^n_m)$$

It is easy to see that the combination of all these $\mathcal{I}(A)$, which we can represent as the direct sum:

$$\mathcal{I} = \bigoplus_{j=1}^{n} \mathcal{I}(A_j)$$

again forms a (non-empty) compact convex sub-set in $\mathbb{R}^{n\times m}$.

The operator Φ is now operating on this set $\mathcal{I}$. Furthermore, we can prove that Φ is continuous from $\mathcal{I}$ to $\mathcal{I}$, and then apply Brouwer's Fixed-point Theorem to guarantee the existence of a fixed-point of Φ. Proofs of this theorem can be found for example in [6, Theorem 18.1] or [29, Theorem 7.5.3].

Theorem 2. *(Brouwer's Fixed-Point Theorem, 1912) Let $f : K \mapsto K$ be a continuous mapping from a non-empty compact convex set K in $\mathbb{R}^n$ into itself. Then there exists a fixed-point of f, i.e. a point $c \in K$ such that $f(c) = c$.*

5.4 The General Case

The general infinite case can be analysed in exactly the same way; basically we only have to replace the finite dimensional spaces $\mathbb{R}^n$ and $\mathbb{R}^{n\times m}$ by an infinite dimensional Banach space [18].

For (countable) infinite sets of constraints a convenient standard space we will consider is given by $l_1 = l_1(\mathbb{N})$ of infinite sequences of real numbers whose sum exists, formally:

$$l_1 = \{(x_i)_{i=1}^{\infty} | x_i \in \mathbb{R} \text{ and } \sum_{i=1}^{\infty} x_i < \infty\}$$

Again the set of possible interpretations $\mathcal{I}(A)$ of an agent:

$$\mathcal{I}(A) = \{(x_i)_{i=1}^{\infty} | x_i \in [0,1] \text{ and } \sum_{i=1}^{\infty} x_i = 1\}$$

and also their combination as a direct product forms a convex compact subset of this Banach space. Furthermore, we can show in the same way as above that Φ is continuous. Therefore we can apply the infinite-dimensional generalisation of Brouwer's Fixed-Point Theorem, namely:

Theorem 3. *(Schauder-Tychonoff Fixed-Point Theorem, 1930/35) Let $f : K \mapsto K$ be a continuous mapping from a non-empty compact convex set K in a Banach (or locally convex) space into itself. Then there exists a fixed-point of f, i.e. a point $c \in K$ such that $f(c) = c$.*

Proofs, applications and discussions of this theorem can be found e.g. in [6, Theorem 18.10], [24, Theorem 5.28] or [22, Section 4.2]. It might be interesting to note that besides metric and topological characteristics one also has to use some kind of additional "geometric" structure in order to prove this theorem (cf. Remarks in [6, p 188]).

5.5 Constructing a Fixed-Point

The fixed-point in the BrouwerbTheorem and in the Schauder-Tychonoff Fixed-Point Theorem respectively is not unique; only the existence of a fixed-point is guaranteed. The problem therefore arises of how to construct a fixed-point or, in other words, of how to construct a concrete semantics $[\![A]\!]$ for each agent A.

Unfortunately, Φ is not a *contraction* such that we can apply Banach's Fixed-Point Theorem; instead it is a *isometry*: it maps vectors or interpretations of length (norm) one into such of the same length. Not very much is known about the construction of fixed-points for isometries or *non-expansive maps* in general [6]. In our case it is nevertheless possible to introduce various methods of constructing a fixed-point for Φ.

Finite Computations. Constructing a fixed-point of Φ via an iterated sequence is a common technique. In trying to guarantee the convergence of this process, difficulties mainly arise because of *infinite computations*. If we consider computations which stop after a finite number of steps, then it is clear that this corresponds to a finite number of iterations of Φ before it reaches a fixed-point.

For finite computations our probabilistic version of CCP will therefore give more or less the same results as in the traditional CCP approach. We just have to repeat the constructions there with the only difference that in PCCP interpretations are not just sets of constraints but that we also keep track of how probabilities change during the computation. Nevertheless, we will get the same *support* sets for interpretations of agents, i.e. the set of constraints where the probability distribution does not vanish. (cf. example 6.1).

Infinite Computations: Method A. Considering infinite computations, a first method to construct a fixed-point of Φ is simply via the iterative application of Φ (staring with an extremal interpretation like $I_0 = \{<true, 1>\}$) and by constructing the pointwise limit of $I_n = \Phi^n I_0$. The latter is defined for a sequence of distributions $I_n = \{<c_i, p_{i,n}>\}$ as:

$$\lim_n I_n = \left\{<c_i, \lim_n p_{i,n}>\right\}_i .$$

If the limit exists then, it is a fixed-point of Φ because of the continuity of Φ. The convergence of this method can be guaranteed by exploiting the Banach lattice structure of $\mathcal{I}$ and the monotonicity of Φ [18, Proposition II.1.a.8].

Infinite Computations: Method B. Another approach would be to look at the average interpretation (instead of a pointwise limit) occurring during the computation. We then had to construct the limit of interpretations in the average

$$\lim_n I_n = \left\{<c_i, \lim_n \frac{1}{n} \sum_{j=1}^{n} p_{i,j}>\right\}_i .$$

In this case we could construct a fixed-point using some "Ergodic Theorem" (see [30, Theorem VIII.3.2,p 213], [6, p 232], or [28]) to guarantee the convergence.

Infinite Computations: Method C. An alternative general method to construct the fixed-point of a non-expanding map like Φ would be to look a some kind of "moving average", e.g. $I_{n+1} = \frac{1}{2} I_n + \frac{1}{2} \Phi I_n$, and look at the limit of this sequence:

$$\lim_n I_n = \left\{<c_i, \lim_n \frac{1}{2} p_{i,n} + \frac{1}{2} p_{i,n-1}>\right\}_i$$

and apply the following [6, Corollary 9.1,p 101].

Theorem 4. *Let K be a closed convex subset of a Banach space X, and $\Phi : K \mapsto K$ be a non-expansive map for which $\Phi(K)$ is compact. Then for each $\alpha \in (0, 1)$ the iterates of the mapping $\Phi_\alpha = (1 - \alpha)I + \alpha\Phi$ converges to a fixed-point of Φ.*

6 Examples

In this section we discuss two examples: the first one illustrates the differences between various notions of nondetermism in CCP, the second one demonstrates the suitability of the denotational semantics defined in Section 5 for modelling infinite computations.

6.1 Nondeterministic Counting

Consider the following PCCP program for computing the natural numbers.

$$\begin{array}{ll} nat(x) :- & true|\frac{1}{2} \rightarrow \mathbf{tell}(x = 0) \\ & \Box\ true|\frac{1}{2} \rightarrow \exists_y(\mathbf{tell}(x = s(y)) \parallel nat(y)) \end{array}$$

We can use this example to illustrate the difference between pure nondeterminism, angelic nondeterminism and probabilistic nondeterminism as we discussed it in Section 2.

With a purely nondeterministic choice $\Box$ (and ignoring the guards) we could not predict what the results of running $nat(x)$ were. It might or might not terminate. Even if it terminates we could not guarantee that it produces as results all natural numbers; it might be that the first branch is always taken and therefore only '0' would be obtained as output.

If we look at the formulation of $nat(x)$ in CCP (which is the same as the PCCP one but for the probability components of the guards) we obtain a more useful semantics. Since the nondeterminism occurring in the CCP formulation is angelic [16] we are guaranteed that $nat(x)$ generates all natural numbers. Note that in the CCP case "fairness" of the choice construct, i.e. all possible branches are sometime executed, is guaranteed by backtracking.

In our PCCP approach, we require that the chances of selecting either of the two branches are equal (namely $\frac{1}{2}$). Table 4 depicts the sequence of interpretations of the call $nat(x)$ obtained by means of the operator Φ introduced in Section 5. In the PCCP case we can deduce even more information about the expected behaviour of $nat(x)$ than in the CCP case: we see that not only all natural numbers are produced, but also that the likelihood for a larger number to be output decreases (exponentially).

	$nat(x)$
I_0	$\{<true, 1>\}$
I_1	$\{<true, 1>\}$
I_2	$\{<x = 0, \frac{1}{2}>, <\exists_y(x = s(y)), \frac{1}{2}>\}$
I_3	$\{<x = 0, \frac{1}{2}>, <x = s(0), \frac{1}{4}>, <\exists_y(x = s(y)), \frac{1}{4}>\}$
$\vdots$	$\vdots$
I_{n+1}	$\{<x = 0, \frac{1}{2}>, <x = s(0), \frac{1}{4}>, \ldots, <x = s^{n-1}(0), \frac{1}{2^n}>, <\exists_y(x = s(y)), \frac{1}{2^n}>\}$
$\vdots$	$\vdots$
$\lim_n I_n$	$\{<x = 0, \frac{1}{2}>, <x = s(0), \frac{1}{4}>, \ldots, <x = s^n(0), \frac{1}{2^{n+1}}>, \ldots, <\omega, 0>\}$

Table 4. The interpretations I_n converge pointwise.

6.2 A Difficult Toy Example

The following declarations have been used in [4] (in their CCP formulation) as an example of the inapplicability of the metric approach for modelling a particular notion of observables for the sublanguage of CCP corresponding to Constraint Logic Programming (i.e. CCP without synchronisation). The observables considered in [4] are defined as the set of the exact results of all finite and infinite computations of a given program.

$$
\begin{aligned}
&p(x) :- q(x)\\
&q(x) :- \mathit{true}|\tfrac{1}{2} \to p(x) \,\square\, \mathit{true}|\tfrac{1}{2} \to (r(x) \parallel \mathbf{tell}(c))\\
&r(x) :- \mathbf{tell}(\mathit{false}).
\end{aligned}
$$

In [4] it is shown that the problem in giving a denotational semantics for these observables is essentially the failure of constructing a convergent sequence of interpretations in any of the classical methods used in denotational semantics (powerdomains, metrics, etc.)

As already observed in Section 5, our semantical construction is similar to the one in [4]. However, as shown in Table 5, in our case the sequence of interpretations defining the meaning of the calls $p(x)$, $q(x)$ and $r(x)$ converges.

The evolution of the initial interpretation $\{<\mathit{true}, 1>\}$ subject to iterated applications of Φ for the agents $p(x)$ and $q(x)$ is pictured in Fig. 2.

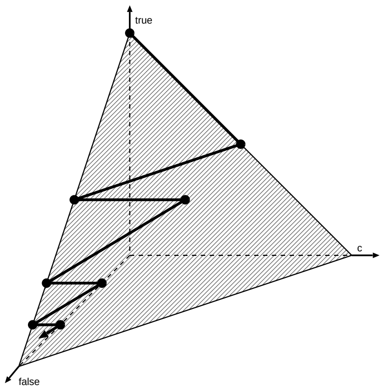

Fig. 2. The geometry of converging interpretations.

The reason why this example works in our case is related to the introduction of randomness for guiding the choice. This allows us to use a notion of pointwise convergence, which abstracts from the actual links between partial constraints on each branch of the computation. In other words, our notion of convergence is not based on the order between constraints (i.e. which constraint generates which other), but rather on the convergence of the probabilities accumulated during the computation.

Table 5. Pointwise convergence of distributions

	$p(x)$	$q(x)$	$r(x)$
I_0	$\{<\mathit{true}, 1>\}$	$\{<\mathit{true}, 1>\}$	$\{<\mathit{true}, 1>\}$
I_1	$\{<\mathit{true}, 1>\}$	$\{<\mathit{true}, 1>\}$	$\{<\mathit{true}, 1>\}$
I_2	$\{<\mathit{true}, 1>\}$	$\left\{<\mathit{true}, \frac{1}{2}>, <c, \frac{1}{2}>\right\}$	$\{<\mathit{false}, 1>\}$
I_3	$\left\{<\mathit{true}, \frac{1}{2}>, <c, \frac{1}{2}>\right\}$	$\left\{<\mathit{true}, \frac{1}{2}>, <\mathit{false}, \frac{1}{2}>\right\}$	$\{<\mathit{false}, 1>\}$
I_4	$\left\{<\mathit{true}, \frac{1}{2}>, <\mathit{false}, \frac{1}{2}>\right\}$	$\left\{<\mathit{true}, \frac{1}{4}>, <c, \frac{1}{4}>, <\mathit{false}, \frac{1}{2}>\right\}$	$\{<\mathit{false}, 1>\}$
I_5	$\left\{<\mathit{true}, \frac{1}{4}>, <c, \frac{1}{4}>, <\mathit{false}, \frac{1}{2}>\right\}$	$\left\{<\mathit{true}, \frac{1}{4}>, <\mathit{false}, \frac{3}{4}>\right\}$	$\{<\mathit{false}, 1>\}$
I_6	$\left\{<\mathit{true}, \frac{1}{4}><\mathit{false}, \frac{3}{4}>\right\}$	$\left\{<\mathit{true}, \frac{1}{8}>, <c, \frac{1}{8}>, <\mathit{false}, \frac{3}{4}>\right\}$	$\{<\mathit{false}, 1>\}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
I_{2n}	$\left\{<\mathit{true}, \frac{1}{2^{n-1}}>, <c, 0><\mathit{false}, \frac{2^{n-1}-1}{2^{n-1}}>\right\}$	$\left\{<\mathit{true}, \frac{1}{2^n}>, <c, \frac{1}{2^n}><\mathit{false}, \frac{2^{n-1}}{2^n}>\right\}$	$\{<\mathit{false}, 1>\}$
I_{2n+1}	$\left\{<\mathit{true}, \frac{1}{2^n}>, <c, \frac{1}{2^n}>, <\mathit{false}, \frac{2^{n-1}}{2^n}>\right\}$	$\left\{<\mathit{true}, \frac{1}{2^n}>, <c, 0>, <\mathit{false}, \frac{2^n-1}{2^n}>\right\}$	$\{<\mathit{false}, 1>\}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\lim_n I_n$	$\{<\mathit{true}, 0>, <c, 0>, <\mathit{false}, 1>\}$	$\{<\mathit{true}, 0>, <c, 0>, <\mathit{false}, 1>\}$	$\{<\mathit{false}, 1>\}$

7 Conclusions and Related Works

A substantially different attempt to incorporate probabilistic aspects into CCP languages has been suggested in [10]. There are at least two main differences between that approach and ours. The first one is the different *motivation*: while we are interested in giving a semantics for probabilistic programming languages, the work in [10] is aimed at modelling stochastic (hybrid) systems. The second difference is about the way how to incorporate randomness: we introduce randomness via the *algorithms*, namely by using a random choice construct, while the approach of [10] is based on some kind of fuzzy *data*, i.e. on the use of random variables.

Related work on the denotational semantics of probabilistic imperative programming languages based on measure theoretic notions has been done by [25, 15, 13].

As for further work: Additional investigations should compare PCCP to other approaches to the semantics of probabilistic computation, e.g. probabilistic predicate transformers [20, 19], probabilistic logics and model checking [2, 12], or probabilistic process algebras [3].

A possible extension of the PCCP language would be to base it on or to combine it with a kind of "fuzzy" constraint system where strict entailment is replaced by a weighted justification of constraints, i.e. to base PCCP on a belief system or probabilistic entailment. This would be particularly interesting in the case of a distributed system where one had to weight the quality of certain entailments according to their plausibility. In this case the transition probability could be made dependent on the quality of the entailment of a certain constraint.

References

1. Emile Aarts and Jan Korst. *Simulated Annealing and Boltzmann Machines.* John Wiley & Sons, Chicester, 1989.
2. Christel Baier and Marta Z. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. Technical Report CSR-96-12, School of Computer Science, University of Birmingham, June 1996.
3. Marco Bernardo and Roberto Gorrieri. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. Technical Report UBLCS-96-17, Department of Computer Science, University of Bologna, January 1997.
4. Frank S. de Boer, Alessandra Di Pierro, and Catuscia 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.
5. Alessandra di Pierro and Herbert Wiklicky. On probabilistic CCP. In *Proceedings of APPIA-GULP-PRODE'97 Joint Conference on Declarative Programming*, June 1997.
6. Kazimierz Goebel and W.A.Kirk. *Topics in Metric Fixed Point Theory*, volume 28 of *Cambridge studies in advanced mathematics.* Cambridge University Press, Cambridge, 1990.

7. David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison–Wesley, Reading, Massachusetts, 1989.
8. Geoffrey R. Grimmett and D.R. Stirzaker. *Probability and Random Processes.* Clarendon Press, Oxford., second edition, 1992.
9. Charles M. Grinstead and J. Laurie Snell. *Introduction to Probability.* American Mathematical Society, Providence, Rhode Island, second revised edition, 1997.
10. Vineet Gupta, Radha Jagadeesan, and Vijay A. Saraswat. Probabilistic concurrent constraint programming. In *Proceedings of CONCUR 97.* Springer Verlag, 1997.
11. Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, Boston, MA, 1997.
12. Michael Huth and Marta Kwiatkowska. On probabilistic model checking. Technical Report CSR-96-15, School of Computer Science, University of Birmingham, August 1996.
13. Claire Jones. *Probabilistic Non-Determinism.* PhD thesis, University of Edinburgh, Edingburgh, 1993.
14. George J. Klir and Bo Yuan. *Fuzzy Sets and Fuzzy Logic.* Prentice Hall, New Jersey, 1995.
15. Dexter Kozen. Semantics for probabilistic programs. *Journal of Computer and System Sciences*, 22:328–350, 1981.
16. Marta Z. Kwiatkowska. Infinite Behaviour and Fairness in Concurrent Constraint Programming. In J. W. de Bakker, W. P.de Roever, and G. Rozenberg, editors, *Semantics: Foundations and Applications*, volume 666 of *Lecture Notes in Computer Science*, pages 348–383, Beekbergen The Nederland, June 1992. REX Workshop, Springer Verlag.
17. Patrick D. Lincoln, John C. Mitchell, and Andre Scedrov. Stochastic interaction and Linear Logic. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecture Note Series*, pages 147–166. Cambridge University Press, Cambridge, 1995.
18. Joram Lindenstrauss and Lior Tzafriri. *Classical Banach Spacses I and II.* Classics in Mathematics. Springer Verlag, Berlin – Heidelberg – New York, 1996.
19. Annabelle McIver and Carroll Morgan. Probabilistic predicate transformers: Part 2. Technical Report PRG-TR-5-96, Programming Research Group, Oxford University Computing Laboratory, 1996.
20. Carroll Morgan, Annabelle McIver, Karen Seidel, and J.W. Sanders. Probabilistic predicate transformers. Technical Report PRG-TR-4-95, Programming Research Group, Oxford University Computing Laboratory, 1995.
21. Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, Cambridge, England, 1995.
22. Nikolaĭ Kapitonovich Nikol'skiĭ, editor. *Functional Analysis I*, volume 19 of *Encyclopaedia of Mathematical Sciences.* Springer Verlag, Berlin – Heidelberg – New York, 1992.
23. Gordon Plotkin. A structured approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.
24. Walter Rudin. *Functional Analysis.* International Series in Pure and Applied Mathematics. McGraw-Hill, New York – Singapore, second edition, 1991.
25. N. Saheb-Djahromi. CPO's of measures for nondeterminism. *Theoretical Computer Science*, 12:19–37, 1980.
26. Vijay A. Saraswat and Martin Rinard. Concurrent constraint programming. In *Proceedings of POPL*, pages 232–245. ACM, 1990.

27. Vijay A. Saraswat, Martin Rinard, and Prakash Panangaden. Semantics foundations of concurrent constraint programming. In *Proceedings of POPL*, pages 333–353. ACM, 1991.
28. Peter Walters. *An Introduction to Ergodic Theory*, volume 79 of *Graduate Texts in Mathematics*. Springer Verlag, New York – Berlin – Heidelberg, 1982.
29. Roger Webster. *Convexity*. Oxford University Press, Oxford – New York – Tokyo, 1994.
30. Kôsaku Yosida. *Functional Analysis*. Springer Verlag, Berlin – Heidelberg – New York, sixth edition, 1980.