On Probabilistic CCP

Alessandra Di Pierro and Herbert Wiklicky

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

Abstract

This paper investigates a probabilistic version of the concurrent constraint programming paradigm (CCP). The aim is to introduce the possibility to formulate so called "randomised algorithms" within the CCP framework. Our approach incorporates randomness directly within the (operational) semantics instead of referring to an "external" function or procedure call.

We define the operational semantics of probabilistic concurrent constraint programming (PCCP) by means of a probabilistic transition system such that the execution of a PCCP program may be seen as a stochastic process, i.e. as a random walk on the transition graph. The transition probabilities are given explicitly.

1 Introduction and Motivation

This paper investigates a probabilistic version of the concurrent constraint programming paradigm. The aim is to introduce the possibility to formulate so called "randomised algorithms" within the concurrent constraint programming (CCP) framework.

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).

It might be important to note that an algorithm based on some element of chance does not necessarily produce random outputs or only approximations of the true result [14]. This is because although the computation might proceed along different, randomly chosen paths, the overall input/output behaviour might still be deterministic (as all paths lead to the same final result).

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 [5], probabilistic primality tests in particular for use in crypto-systems [14], and randomised proof procedures (e.g. for linear logic [10]).

In our approach we will incorporate this element of randomness in the form of a probabilistic choice directly within the semantics, instead of referring to it as some "external" function or procedure call, e.g. a (pseudo-)random number generator `random()` in an imperative language like C, as one can find it in many of the above mentioned approaches.

This embedding of randomness within the semantics of a well structured programming paradigm, like CCP, also aims at providing a sound framework for formalising and reasoning about randomised algorithms and programs.

2 Conceptual Framework

Before introducing the probabilistic concurrent constraint (PCCP) language we have to discuss a basic question: What is a probabilistic choice and how is it different from a non-deterministic choice?

The basic idea here is not to look just at what is possible, but to repeat a choice and look how often each possibility is realized.

Consider, for example, a Turing Machine (TM). In the case of a deterministic TM, at each stage of computation there is only one possible way to continue (there is a well-defined transition *function*). In the non-deterministic case we have stages during the computation where it is not known or specified how to continue with the computation (a transition *relation* is specified in this case). One could say – from an internal point of view – that a choice has to be made each time when we reach such a stage of how to continue; or note – from an external point of view – that only partial information of how the computation will continue was specified.

However, sometimes it is not enough to consider just the collection of possible continuations of the computation. Some additional information is needed about how the computation will continue.

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

Another example of providing some 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.

3 Probability Theory

The standard approach in science (from physics to the social sciences) to describe a (choice) process where we have only, in some sense limited, knowledge of what really is going to happen is via probability theory.

Probability is a concept intrinsically related to experiments and observations. One could argue that probability theory is not about the real facts, objects, etc. but rather about "partial knowledge" or "information" about these things, and sometimes one has to take into account that the available information will change from time to time.

As with all mathematical theories, one has to separate between the formal, theoretical framework – which is a well defined mathematical theory – and possible applications of this theory to physical or real world phenomena. The mathematical theory can't say anything about "real" entities; it just deals with models (which might be wrong but still consistent).

The basic mathematical structure of probability theory [6] – as a theory providing a theoretical model for reasoning about stochastic processes – is usually referred to as measure theory. This is similar to topology in that it identifies a useful sub-family of sub-sets of a given set (universe). On this family some *measure* is defined, which assigns certain sizes (probabilities, weights, etc.) to each of the allowed sub-sets.

Definition 3.1 *Given a set X. We call a family $\mathcal{S} \subseteq \mathcal{P}(X)$ of subsets of X a σ-algebra iff*

(i) $\emptyset \in \mathcal{S}$

(ii) $A \in \mathcal{S} \Rightarrow \complement A \in \mathcal{S}$

(iii) $A_i \in \mathcal{S} \Rightarrow \bigcup_{i=1}^{\infty} A_i \in \mathcal{S}$

The interpretation of this is the following: X is a set of (simple) events, representing the possible outcomes of an experiment (in our case possible choices). We are then interested in sets of (elementary) events, because we might have a situation where our experiments are not "fine" enough to distinguish between individual events. On the other hand, we also can't assume that our experiments are accurate enough for all possible subsets of X. Therefore we just look at a certain subset of the power set $\mathcal{S} \subseteq \mathcal{P}(X)$. All this defines how precise our measurements or experiments can be. The function p (or μ) finally assigns some probability (or measure) to all sets in $\mathcal{S}$.

Definition 3.2 *Given a set X together with a σ-algebra $\mathcal{S}$ on X. We call a function $p : \mathcal{S} \mapsto \mathbb{R}$ a (normed) probability measure on $(X, \mathcal{S})$ iff*

(i) $p(X) = 1$

(ii) $p(A) \geq 0$ *for all* $A \in \mathcal{S}$

(iii) $p(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} p(A_i)$ *for all* $A_i \in \mathcal{S}$ *with* $A_i \cap A_j = \emptyset, i \neq j$

The determination of a certain probability is not part of this mathematical theory. Instead, this has always to be based on some assumption or some physical experiment. The mathematical theory just is about the reasoning with probabilities, not about their determinations (in as far as we are not deducing some probability from others).

The standard way to determine and to interpret probabilities is by identifying them with relative frequencies, this way linking the mathematical model to some physical, observable "reality". In this we look at the number of times we observe some event A and compare it to the overall number of observations when more and more experiments are performed. Then we get as the probability of A

$$p(A) = \lim \frac{\text{no of observations of A}}{\text{no of all observations}}$$

4 Probabilistic Constraint Systems

CCP languages are defined parametrically w.r.t. to a given *constraint system*. Therefore, in order to define our language we need to specify its underlying constraint system.

We will consider cylindric constraint systems as defined in [17, 18]. In order to deal with the probabilistic features of our language we have to make some additional assumptions which allow us to use real numbers for expressing (transition) probabilities. In particular, we will require the presence of elements δ_{Xr}, $r \in \mathbb{R}, X \in \mathit{Var}$ within the constraint system, i.e. of equality constraints involving real numbers.

P	::=	$D.A$
D	::=	$\epsilon \mid D.D \mid p(x) :- A$
A	::=	$stop \mid tell(c) \mid \Box_{i=1}^{n}\ c_i \mid p_i \rightarrow A_i \mid A \parallel A \mid \exists x A \mid p(x)$

Table 1: The syntax for PCCP.

To illustrate the role of these additional constraints we can think of the store as being composed of two different parts: one corresponds to the classical store in CCP, the other one only contains information about (transition) probabilities. Constraints of the form δ_{Xr} belong to the latter, which we will refer to as "probability store".

As we will explain later on, the computational mechanism depends crucially on the information provided by the probability store. The probability that certain computational steps will be made is determined by the constraints δ_{Xr} in this part of the store.

We can distinguish two major ways of how the probability store might evolve: a static one and a dynamic one.

In the static case, all (transition) probabilities are fixed initially. This means that the constraints expressing these probabilities are already in the probability store before even computations take place and that they are not changed in the course of the execution.

In the dynamic case, the information about the probabilities guiding the transition steps (e.g. in the choice agents) might be added during the computation, for example by the agent $tell(\delta_{Xr})$. This would not harm monotonicity of CCP. One could also think about the possibility of changing probabilities depending on the circumstances (current store, configuration, etc.). This would imply replacing constraints in the probability store and therefore losing the monotonicity. We leave the last possibility as future work, and concentrate in this paper on the definition of a monotonic probabilistic extension of the CCP paradigm, that we call Probabilistic Concurrent Constraint (PCCP).

5 Probabilistic CCP

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 introduce a probabilistic choice (instead of a purely non-deterministic one) as

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

To be precise this syntactic construct expresses two possible expressions: p_i may either denote a constant or a variable. In both cases p_i stands for an equality constraint of the form δ_{Xr}. In case p_i is a constant, e.g. 0.5, then the corresponding constraint is $\delta_{X0.5}$, where X is a variable never referred to anywhere else. In case p_i is a variable, e.g. X, the associated equality constraint is of the form δ_{Xr}.

The intended meaning of this construct is as follows: First, check whether (the primary) constraints c_i and the constraints associated to p_i are entailed by the store (in case p_i is a constant we assume that the corresponding constraint was told to the store

beforehand). All those agents A_i such that c_i and (the constraint associated to) p_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:

$$\tilde{p_i} = \frac{p_i}{\sum_{\vdash c_j \sqcup \delta_{X_j p_j}} p_j}$$

where the sum is over all enabled agents such that we get that $\sum \tilde{p_i} = 1$.

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.

6 Operational Semantics for PCCP

We define the operational semantics in the style of SOS [15], i.e. by means of a transition system which describes the evolution of the system of concurrent agents in a structural way.

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 $\langle A, d \rangle$.

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

The label p simply 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.

Suppose, for example, we have

$$A_1 \longrightarrow_{p_{12}} A_2 \text{ and } A_2 \longrightarrow_{p_{23}} A_3$$

then

$$A_1 \longrightarrow_{p_{13}}^* A_3$$

with

$$p_{13} = p_{12}\ p_{23}.$$

If there are several paths leading from configuration A to B, then the probability label p for $A \longrightarrow_p^* B$ is the sum of all the probabilities associated to these different paths.

The agent *stop* represents successful termination. The basic actions are given by 'telling' a constraint to the common store 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 $c \mid p \rightarrow A$. 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

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

Table 2: The transition system for PCCP.

previous section. Rule **R2** gives a formal description. One of the agents A_j is enabled if the store entails both the (primary) constraint c_j and $\delta_{X_j p_j}$, (this implies that a definitive real value for the transition probability p_j is known). After normalising the probabilities to

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

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, 17] 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 [18], 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.

7 The Observables

From the operational model described above we can abstract various semantics depending on what we want to *observe* of a process. The classical notion of observables in CCP are the results computed by an agent for a given initial store, where we can look at finite as well as infinite computations. Fair computations are to be considered when infinite

computations are taken into account. The result of a fair computation is defined as the least upper bound of all the stores occurring in the computation. This definition is possible because for the monotonic properties of CCP the stores of a computation form a chain.

Our observables correspond to the input/output behaviour of an agent. We will consider (for the time being) only finite computations which are obviously fair. In our case the result of a computation does not depend only on the initial store. In fact, given an input the results obtained by executing a certain program depend also on the probability distribution associated with the transition system.

The probability of obtaining a certain result depends on the probabilities p associated to the possible paths which lead to it. It is therefore the sum of the probabilities associated to all these paths, as explained in Section 6.

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 (which corresponds to the classical final store); and p is the probability of reaching that result (which represents the information contained in the probability store). More formally,

$$\mathcal{O}_P(A, d) \quad = \quad \{(c, p) \mid \langle A, d\rangle \longrightarrow_p^* \langle stop, c\rangle\}.$$

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.

As already noted in the introduction, it might happen that although the computation is proceeding randomly, there is nevertheless only one final result c which (by combing the probabilities along several possible paths leading to it) will be reached with probability $p = 1$. Such an agent therefore exhibits a completely deterministic input/output behaviour despite the apparent probabilistic nature of the computation.

8 Classification of Transition Systems

A transition system may be seen from at least three different points of view. According to the most common practice in semantics, it is a relation on the set of configuration $\longrightarrow \subseteq Conf \times Conf$. Alternatively, it can be seen as a (directed) graph $\Gamma = (Trans, Conf)$ where two nodes are connected iff they are related by $\longrightarrow$. Another possibility is to see it as a matrix, T, indexed by the set of configurations such that there is a non-zero entry $T_{s_1 s_2}$ iff $s_1 \longrightarrow s_2$. The same views can be adopted also for a labelled transition system, except that we have to either extend the relation to $\longrightarrow^l \subseteq Conf \times Label \times Conf$, or associate some labels to the edges in the transition graph, or take matrix entries from *Label*.

A probabilistic transition system is nothing more than a labelled transition systems where the labels are taken to be (normalised) transition probabilities. It is often represented in its matrix form as a so called stochastic matrix [19]. We make it "operational" – for example if we look at the graph representation – in the following way: At each step (of computation) we simply randomly choose one of the edges leading away from a configuration (according to the assigned, or normalised probabilities) and follow it to the

next configuration to repeat this process; this leads to what is called a stochastic process, a random walk on the transition graph [19, 12, 2].

Given any representation of a transition system we may classify it *statically* by looking at what transitions will be *possible*: We call a transition system (a) *deterministic* if the transition relation is many-to-one, or if at most one edge leads away from each node, or if only one entry is non-zero in the matrix per row (or column). Otherwise we call it *non-deterministic*.

But we can also look at a *dynamic* interpretation of a transition system by asking not what is possible, but what *will happen* when we look at the executions, paths etc. of the transitions. If we look at a statically deterministic transition system we find that (at most) only one path is possible from each node, thus what is possible will happen. But if we look at a non-deterministic system where more than one path is possible we can distinguish further, for example: (when repeating the execution) all possible paths are taken sometime we can call the system *fair*, or – in the case of a *probabilistic* transition system – (when repeating the execution) the possible paths are taken according to the respective probabilities.

9 Examples

Let us now present three very easy examples of PCCP programs:

Example 9.1 A very simple program in PCCP would be the following one:

```
   c | 0.5 -> A
[] c | 0.5 -> B
```

where `A` and `B` are any two agents. If the constraint `c` is not entailed by the store this process will get stuck as none of the agents is enabled. On the other hand, in the case where `c` is entailed both agents are enabled and are taken with the same probability of 50%. That is, when this program is executed repeatedly the relative frequency of each of the two agents to be executed is the same.

Another very simple example is given by the following:

```
   c | 1.0 -> A
[] c | 1.0 -> B
```

Because of the normalisation of probabilities we introduced, this is semantically equivalent to the agent in the previous example.

Under the hypothesis that the constraint `d` is not entailed, another equivalent program is given by:

```
   c | 1.0 -> A
[] c | 1.0 -> B
[] d | 1.0 -> C
```

Example 9.2 Consider the following PCCP program:

```
   c | 1.0 -> A
[] c | 0.0 -> B
```

Again, if `c` is not entailed, then the whole process is stuck. But if `c` is entailed then the behaviour is as follows: With probability 1 agent `A` is executed and with probability 0 agent `B`. Note, that this does not mean that agent `B` is never executed: When we repeat "test-runs" of this program, then only the relative frequency of observing `B` being executed among all observed "test-runs" vanishes (in order to make this a deterministic execution we had to eliminate the second branch completely).

Example 9.3 A final example we will give here is an implementation of a well known stochastic process, namely a Random Walk (in one dimension).

```
walk(X,Y) :- X <> Y | 0.5 -> X' = X+1 || walk(X',Y)
          [] X <> Y | 0.5 -> Y' = Y+1 || walk(X,Y')
          []  X = Y | 1.0 -> A
```

The execution of `walk(1,2)` is semantically equivalent to the PCCP fragment we gave in the last example when we take as `B` a never terminating agent.

This procedure will call itself until both its parameters are equal and then execute the agent `A`. Each time it is called with different parameters it will increase either variable `X` or variable `Y` by one, with which one being determined by a probabilistic choice which gives both variables a 50% chance of being increased. From the theory of random walks or Markov chains we know that the probability that both variables once have the same value is one [6]. Therefore we know that the probability of enabling only the third agent `A` (which then will be executed) is one and the probability of an infinite loop is zero.

10 Conclusions and Further Work

We have introduced an operational semantics for a probabilistic version of CCP. This is based on a probabilistic transition system such that the execution of a PCCP program may be seen as a stochastic process, i.e. as a random walk on the transition graph. The transition probabilities are given explicitly.

A possible extension would be to base our treatment on 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.

Further work will also lead to the formulation of a denotational semantics for PCCP. It is natural to base this semantics on measure theoretic notions, e.g. on spaces of measures, cf. [16, 9, 8]. Additional investigations will relate PCCP to other semantic approaches towards probabilistic computation, e.g. probabilistic predicate transformers [13, 11], probabilistic logics and model checking [2, 7], and probabilistic process algebras [3].

References

[1] Emile Aarts and Jan Korst. *Simulated Annealing and Boltzmann Machines.* John Wiley & Sons, Chicester, 1989.

[2] Christel Baier and Marta 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] 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, 18-20 Nov.

[5] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison–Wesley, Reading, Massachusetts, 1989.

[6] Geoffrey R. Grimmett and D.R. Stirzaker. *Probability and Random Processes.* Clarendon Press, Oxford., second edition, 1992.

[7] Michael Huth and Marta Kwiatkowska. On probabilistic model checking. Technical Report CSR-96-15, School of Computer Science, University of Birmingham, August 1996.

[8] C. Jones. *Probabilistic Non-Determinism.* PhD thesis, University of Edinburgh, Edingburgh, 1993.

[9] Dexter Kozen. Semantics for probabilistic programs. *Journal of Computer and System Sciences*, 22:328–350, 1981.

[10] 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.

[11] Annabelle McIver and Carroll Morgan. Probabilistic predicate transformers: Part 2. Technical Report PRG-TR-5-96, Programming Research Group, Oxford University Computing Laboratory, 1996.

[12] Bojan Mohar and Wolfgang Woess. A survey on spectra of infinite graphs. *Bulletin of the London Mathematical Society*, 21:209–234, 1988.

[13] 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.

[14] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, Cambridge, England, 1995.

[15] Gordon Plotkin. A structured approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[16] N. Saheb-Djahromi. CPO's of measures for nondeterminism. *Theoretical Computer Science*, 12:19–37, 1980.

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

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

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