Probabilistic Concurrent Constraint Programming

Vineet Gupta [⋆] Radha Jagadeesan [⋆⋆] Vijay Saraswat [⋆⋆⋆]

Abstract. We extend cc to allow the specification of a discrete probability distribution for random variables. We demonstrate the expressiveness of pcc by synthesizing combinators for default reasoning. We extend pcc uniformly over time, to get a synchronous reactive probabilistic programming language, Timed pcc. We describe operational and denotational models for pcc (and Timed pcc). The key feature of the denotational model(s) is that parallel composition is essentially set intersection. We show that the denotational model of pcc (resp. Timed pcc) is conservative over cc (resp. tcc). We also show that the denotational models are fully abstract for an operational semantics that records probability information.

1 Introduction

Concurrent constraint programming (CCP, [Sar93]) is an approach to computation which uses constraints for the compositional specification of concurrent systems. It replaces the traditional notion of a store as a valuation of variables with the notion of a store as a constraint on the possible values of variables. Computation progresses by accumulating constraints in the store, and by checking whether the store entails constraints. A salient aspect of the cc computation model is that programs may be thought of as imposing constraints on the evolution of the system. cc provides four basic constructs: (tell) a (for a a primitive constraint), parallel composition (A, B), positive ask (**if** a **then** A) and hiding (**new** X **in** A). The program a imposes the constraint a. The program (A, B) imposes the constraints of both A and B — logically, this is the conjunction of A and B. **new** X **in** A imposes the constraints of A, but hides the variable X from the other programs — logically, this can be thought of as a form of existential quantification. The program **if** a **then** A reduces to A as and when the store entails a — logically, this is an (intuitionistic) implication.

A primary domain of applicability for CCP ideas is the compositional modeling of physical systems [FBB+94, FS95, GSS95]. To handle reactive systems, such as controllers for reprographics system components such as paper-feed mechanisms, CCP was extended in [SJG94] to handle discrete time. The basic idea was to adapt to CCP the Synchrony Hypothesis of Berry et al: Time is measured by the sequence of instantaneous interactions that the program has with its environment. The need to handle interrupts instantaneously (e.g., a paper jam causing power to be interrupted) rather than at the next occasion when the environment interacts with the system was handled by introducing a

[⋆] Xerox PARC, 3333 Coyote Hill Road, Palo Alto Ca 94304; vgupta@parc.xerox.com
[⋆⋆] Department of Mathematics, Loyola University-Lake Shore Campus, Chicago IL 60626. radha@math.luc.edu
[⋆⋆⋆] AT & T Research, 180 Park Avenue, Florham Park, NJ 07932; vj@research.att.com

new idea [SJG]: a combinator (**if** a **else** A) which triggers A on the *absence* of e, that is under the assumption that e has not been produced in the store, and will not be produced throughout system execution at the present time instant. This causes the input-output relation of processes to be possibly non-monotonic but [SJG] presents a simple denotational model. The resulting theory is considerably smoother and enables the extension to hybrid systems in [GJS], which is necessary to handle continuously varying components such as variable-speed motors.

This entire conceptual framework is based, however, on the assumption that enough information will be available to model a given physical system in as accurate detail as is needed so that appropriate causal, determinate constraint-based models can be constructed. However, this assumption is violated in many physical situations — it becomes necessary to work with approximate, incomplete or uncertain information. We consider three paradigmatic examples. Consider first a telephony system that has to respond to alarms being generated from another complicated system that may only be available as a black-box. A natural model to consider for the black-box is a stochastic one, which represents the timing and duration of the alarm by random variables with some given probability distribution. Consider next situations in which it becomes necessary to model the system stochastically: for instance, not enough may be known about the mechanisms controlling wear-and-tear of rollers so that the actual departure time of a sheet of paper from a worn roller may have to be approximated by a suitably chosen distribution of a random variable. It becomes necessary then to compute system response in a setting in which system models and inputs may behave stochastically. Third, consider model-based diagnosis settings. Often information is available about *failure models* and their associated probabilities, for instance from field studies and studies of manufacturing practices. Failure models can be incorporated by assigning a variable, typically called the *mode* of the component to represent the physical state of the component, and associating a failure model with each value of the mode variable. Probabilistic information can be incorporated by letting the mode vary according to the given probability distribution [dKW89]. The computational task at hand becomes the calculation of the most probable diagnostic hypothesis, given observations about the current state of the system.

In this paper we develop the underlying theory for (timed) *probabilistic* cc — (timed) cc augmented with the ability to describe stochastic inputs and system components – thereby making it possible to address the phenomena above within CCP. Our basic move is to allow the introduction of (discrete) random variables (RVs) with a given probability distribution. A run of the system will choose a value for an RV, with the given probability; these probability values accumulate as more and more choices are made in the course of the run. Alternate choices lead to alternate runs, with their own accumulated probability values. Inconsistencies between chosen values of RVs and constraints in the store lead to some runs being dropped. Now the possibly multiple consistent (normalized) outcomes of the various runs can be declared as the probabilistic outputs of the program. The extension to timed pcc is done in exactly the same way as the extension of cc to tcc [SJG94], using the Synchrony Hypothesis.

Probabilistic cc. In more detail we proceed as follows. We add a single combinator **new** (r, f) **in** A where r is a variable, f is its probability mass function, and A is an agent. Intuitively, such an expression is executed by making a choice for the r according to the mass function f.

Example 1. Consider the program:

$$\textbf{new}\ (r, f : f(0) = f(1) = f(2) = f(3) = 0.25)\ \textbf{in}\ [\textbf{if}\ r = 0\ \textbf{then}\ a, \textbf{if}\ r = 1\ \textbf{then}\ b]$$

On input `true`, this program can produce outputs a, b or `true`. The probability of a being the output is $.25$; similarly for b. The probability of the output `true` is $.5$.

Random variables, like any other variables, may be constrained. These constraints may cause portions of the space of the joint probability distribution to be eliminated due to inconsistency.

Example 2. Consider the program below, with the same f as before, and with the constraint $r \in \{0, 1\}$ interpreted as " r is 0 or 1" [HSD92].

$$\textbf{new}\ (r, f)\ \textbf{in}\ [\textbf{if}\ r = 0\ \textbf{then}\ a, \textbf{if}\ r = 1\ \textbf{then}\ b, r \in \{0, 1\}]$$

On input `true`, this program will produce outputs a or b; `true` is not a valid output, because of the constraint $r \in \{0, 1\}$. Each of the two execution paths are associated with the number 0.25 because of f; however to compute the probability of each path, we must normalize these numbers with the sum of the numbers associated with all successful execution paths. This yields the probability $0.5 = (0.25/(0.25+0.25))$ for each path.

We make the following further assumptions on RVs (such as r in the above program)

1. RVs are uniquely associated with probability distributions — this is achieved syntactically by requiring that the probability distribution be imposed when the variable is declared.
2. Distinct RVs are assumed to be independent — correlations between RVs, *i.e.* joint probability distributions, are achieved by constraints.

Example 3. Consider the program (with f as before):

$$[\textbf{new}\ (r, f)\ \textbf{in}\ x = r], [\textbf{new}\ (r', f)\ \textbf{in}\ [\textbf{if}\ r' \in \{0, 1\}\ \textbf{then}\ x = r']]$$

The first agent causes the generation of four execution paths, each with associated number 0.25. For each of these, the second generates four more; however paths corresponding to six choices for $r \times r'$ are ruled out. Consequently the following results (with associated probabilities) are obtained: $x = 0(0.3)$, $x = 1(0.3)$, $x = 2(0.2)$, $x = 3(0.2)$.

The presence of RVs such as above allows us to construct program combinators reminiscent of combinators that detect "negative information" in synchronous programming languages [Hal93, BB91, Har87, SJG].

Example 4. Consider the program

$$\textbf{new } (r, g : g(0) = \epsilon, g(1) = (1-\epsilon)) \textbf{ in } [\textbf{if } a \textbf{ then } r = 0, \textbf{if } r = 1 \textbf{ then } b]$$

This program on input `true` produces constraint b with probability $(1-\epsilon)$ and `true` with probability ϵ. On input a the program results in a with probability 1. Thus, this program can be thought of as **if** a **else** $_{(1-\epsilon)}$ b *i.e.* if a is not present, produce b with probability $(1-\epsilon)$. Note however that if a is present, b is *not* produced. Note that the same result can be obtained by running n-fold parallel composition of $\epsilon = 0.5$ components, for arbitrarily large n.

In essence, we have used the RV to set up a very high expectation that b will be produced; however, this expectation can be categorically denied on the production of a since the entire probability mass is shifted to the hitherto low end possibility. The construct can be generalized to arbitrary agents A by:

$$\textbf{if } a \textbf{ else }_{(1-\epsilon)} A \stackrel{d}{=} \textbf{new } X \textbf{ in } [\textbf{if } a \textbf{ else }_{(1-\epsilon)} X, \textbf{if } X \textbf{ then } A]$$

Note however, that the program **if** a **else** $_{1-\epsilon} b$, **if** b **else** $_{1-\epsilon} a$ is determinately probabilistic [MMS96] rather than *indeterminate*: assuming ϵ is very small, it produces on input `true` the distribution $(a, 0.5), (b, 0.5)$, rather than producing either a or b indeterminately.

Example 5 Probabilistic Or. The probabilistic choice operator of [JP89], $P +_r Q$, can be defined as follows:

$$\textbf{new } (X, f : (f(0) = r, f(1) = (1-r))) \textbf{ in } (\textbf{if } X = 0 \textbf{ then } P, \textbf{if } X = 1 \textbf{ then } Q)$$

$P +_r Q$ reduces to P with probability r and to Q with probability $(1-r)$. [JP89] require that this operator satisfies the laws of commutativity, associativity and absorption ($P +_r P = P$); this will be the case in the model we now develop.

What is a model for pcc*?* We briefly review the model for cc programs, referring the reader to [SRP91] for details. An observation of a cc program A is a store u in which it is quiescent, i.e. running A in the store u adds no further information to the store. Formally we define the relation $A \downarrow^u$, read as A *converges on* u or A *is quiescent on* u, with the evident axioms:

$$\frac{a \in u}{a \downarrow^u} \quad \frac{A_1 \downarrow^u \quad A_2 \downarrow^u}{(A_1, A_2) \downarrow^u} \quad \frac{c \notin u}{(\textbf{if } c \textbf{ then } A) \downarrow^u} \quad \frac{A \downarrow^u}{(\textbf{if } c \textbf{ then } A) \downarrow^u} \quad \frac{A \downarrow^v \quad \exists_X u = \exists_X v}{(\textbf{new } X \textbf{ in } A) \downarrow^u}$$

The denotation of a program A can be taken to be the set of all u such that $A \downarrow^u$. The semantics is compositional since the axioms above are compositional, i.e. the convergence of A on u depends only upon the convergence of the sub-programs of A on u. The output of A on any given input a is now the least u containing a on which A converges. This immediately tells us that the set denoting A must be closed under $\sqcap$. Such a set is called a *closure operator*. It is easy to adjust this view in the presence of divergence, where divergence may arise because of infinite execution sequences or inconsistency of generated constraints [SRP91]. In essence, the denotation becomes a *partial* closure operator, characterized by sets of constraints closed under *non-empty* glbs.

We turn now to pcc. We make two crucial observations. First, note that an RV-valuation (i.e., an assignment of values to hidden RVs) reduces a pcc program to a cc program, which can be represented as a set of its quiescent points. Consider Example 4. The choice of r as 0 yields the fixed-point set $\{\mathtt{true}, a, b, a \wedge b\}$. Similarly the choice of r as 1 yields $\{b\}$ — this is the fixed-point set of a partial closure operator undefined on a.

One might consider, then, the denotation of a pcc process to be a set of pairs (c, p) where c is a fixed-point, and p is the sum of the probabilities associated with the RV-valuations which realize c, that is, for which c is a fixed-point. However, from just this information it is not possible to recover the fixed-point *set* generated by an RV-valuation; hence it is necessary to record correlations explicitly. That is, for an arbitrary set of fixed-points X, record (the sum of the probabilities associated with) those valuations on RVs for which all elements of X are fixed-points. Technically, this is best done by associating with the set S of fixed-points a probability lattice, defined formally in Section 2.2, which associates a probability with each element in the freely-generated complete and completely distributive lattice (= free profinite lattice, [Joh82]) on S. A process P can then be taken to be a pair (P_A, P_C) where P_A is a set of constraints, and P_C a probability lattice on P_A; P must satisfy conditions that ensure that it is "generable" from a program, intuitively, that it is possible to recover from it a (finite) set of (partial) closure operators, namely those generated by the reduction of the given program under different RV-valuations.

Thus for instance the program in Example 4 has the fixed-point set $\{\mathtt{true}, a, b, a \wedge b\}$. The probability lattice associated with this process yields 0.5 for $\mathtt{true}$, a, $a \wedge b$ and 1 for b; 0.5 for the collection $\{\mathtt{true}, a, a \wedge b, b\}$ (since each element in this set is a fixed-point for the RV-valuation $X = 0$, which has probability 0.5); 0.5 for the collection $\{a, b\}$ etc.

The resulting model has the following key properties: (1) parallel composition is essentially set intersection (Section 2.3); (2) there is a natural embedding of the space of cc processes into the space of pcc processes (Section 2.3); (3) it is fully abstract with respect to a notion of observation that includes probabilities (Section 2.5).

Timed pcc arises from pcc by the integration of a notion of time. This allows the representation of (possibly probabilistic) timed reactive systems. We use the same mechanism to extend pcc that was used to extend Default cc to Default tcc in [SJG]. At each time step the computation executed is a pcc program. Computation progresses in cycles: input a constraint from the environment, compute to quiescence and compute the program to be executed at subsequent time instants.

As for Default tcc we add to the untimed pcc a single temporal control construct: **hence** A. Declaratively, **hence** A imposes the "constraints" of A at every time instant after the current one. Operationally, if **hence** A is invoked at time t, a new copy of A is invoked at each instant in $t' > t$. As shown in [SJG] **hence** can combine with ask operations to yield rich patterns of temporal evolution. In particular, Timed pcc satisfies probabilistic variants of following key features of synchronous programming languages: (1) The notion of time is *multiform* — any signal can serve as the notion of time. (2) All the ESTEREL-style combinators, including (the probabilistic versions of) the strong preemption combinators such as "`do A watching a`", are expressible.

The construction of the denotational model of Timed pcc from pcc follows the idea that "processes are relations extended over time" ([Abr93, Abr94]). Formally, the construction follows the definition of the Default tcc model from the Default cc model described in [SJG]. Consequently, the model for Timed pcc "inherits" the good formal properties of pcc (Section 3): (1) Parallel composition is essentially set intersection in this model; (2) The Timed pcc model is conservative over the `tcc` model of [SJG94]; (3) The Timed pcc model is fully abstract.

Related work The role of probability has been extensively studied in the context of several models of concurrency. Typically, these studies have involved a marriage of a concurrent computation model with a model of probability. For example, stochastic Petri nets [Mar89, VN92] add Markov chains to the underlying Petri net model. Similarly probabilistic process algebras add a notion of randomness to the underlying process algebra model. This theory is well developed and is primarily about the interaction between probability and non-determinism, see for example [HJ90, vGSST90, JY95, LS91, HS86, CSZ92]. These studies have been carried out in the traditional framework of semantic theories of process algebras, *e.g.* theories of (probabilistic) testing, relationship with (probabilistic) temporal logics etc.

We start with the underlying concurrent model being cc. Inspired by [BLFG95], we build an integrated treatment of probability and the underlying concurrent programming paradigm. This is revealed in the dual roles played by the underlying cc paradigm. In addition to being utilized in the specification of system (this is similar to the use of the Petri nets/process algebras in the above approaches), the cc paradigm is exploited to build and specify joint probability distributions of several variables (as illustrated in earlier examples). Furthermore, our model remains determinately probablistic.

The development of probabilistic frameworks in knowledge representation has been extensive [Pea88]. It is easy to see how to express the joint probability distributions of Bayesian networks within pcc, e.g. by associating a random variable for each row in the joint conditional dependence matrix for each node in the network. In this sense, pcc provides a simple but powerful notation for Bayesian networks. It also seems feasible to represent probabilistic Dempster-Shafer rules of the form "If it is Sunday, John will go to the baseball game" with a given strength (say 0.8) as the agent:

$$\textbf{if } sunday(today) \textbf{ then new } (x, f : f(1) = 0.8, f(0) = 0.2) \textbf{ in} \\ \textbf{if } x = 1 \textbf{ then } will\_go(john, bgame)$$

However, pcc does not allow the direct manipulation of conditional probability assertions e.g. `p(fly(tweety) | bird(tweety)) = 0.9` as in the logics of [Nil86, FJM90].

2 Model for Probabilistic cc

2.1 Constraint systems with discrete random variables

A constraint system with discrete random variables extends the usual notion of constraint systems with special variables called random variables, that can take values from non-negative integers $\mathbb{N}$. Such constraint systems have an explicit notion of inconsistencies of random variables.

Such a constraint system $\mathcal{D}$ is a system of partial information, consisting of a set of primitive constraints (first-order formulas) or *tokens* D, closed under conjunction and existential quantification, and an inference relation (logical entailment) $\vdash$ that relates tokens to tokens. We use $a, b, \ldots$ to range over tokens. The entailment relation induces through symmetric closure the logical equivalence relation, $\approx$.

Definition 1. A *constraint system with random variables* is a structure $\langle D, \vdash, \mathbf{Var}, \mathbf{RandVar} \subseteq \mathbf{Var}, \{\exists_X \mid X \in \mathbf{Var}\}, \mathtt{false} \in D\rangle$ such that:

1. D is closed under conjunction($\wedge$); $\vdash \subseteq D \times D$ satisfies:
   (a) $(\forall a)\,[\mathtt{false} \vdash a]$
   (b) Identity: $a \vdash a, \quad a \vdash a'$ and $a' \wedge a'' \vdash b$ imply that $a \wedge a'' \vdash b$.
   (c) Conjunction: $a \wedge b \vdash a,\ a \wedge b \vdash b,\ a \vdash b_1$ and $a \vdash b_2$ implies that $a \vdash b_1 \wedge b_2$.
2. $\mathbf{Var}$ is an infinite set of *variables*, such that for each variable $X \in \mathbf{Var}$, $\exists_X : D \to D$ is an operation satisfying usual laws on existentials:

   $$a \vdash \exists_X a,\ \exists_X(a \wedge \exists_X b) \approx \exists_X a \wedge \exists_X b,\ \exists_X \exists_Y a \approx \exists_Y \exists_X a,\ a \vdash b \Rightarrow \exists_X a \vdash \exists_X b.$$

3. $\mathbf{RandVar}$ is a set of *random variables* satisfying:
   (a) $(\forall r \in \mathbf{RandVar})\,(\forall i \in \mathbb{N})\,[r = i \in D]$.
   (b) $(\forall r \in \mathbf{RandVar})\,(\forall i, j \in \mathbb{N})\,[i \neq j$ implies $(r = i) \wedge (r = j) \vdash \mathtt{false}]$
4. $\vdash$ is decidable.

The last condition is necessary to have an effective operational semantics.

A *constraint* is an entailment closed subset of D. For any set of tokens S, we let $\overline{S}$ stand for the constraint $\{a \in D \mid \exists\{a_1, \ldots, a_k\} \subseteq S.\, a_1 \wedge \ldots \wedge a_k \vdash a\}$. For any token a, $\overline{a}$ is just the constraint $\overline{\{a\}}$. The set of constraints, written $|D|$, ordered by inclusion($\subseteq$), forms a complete algebraic lattice with least upper bounds induced by $\wedge$, least element $\mathtt{true} = \{a \mid \forall b \in D.\, b \vdash a\}$ and greatest element $\mathtt{false} = \overline{\mathtt{false}} = D$. Reverse inclusion is written $\supseteq$. $\exists, \vdash$ lift to operations on constraints.

An example of such a system is the system FD [HSD92]. In this system, variables are assumed to range over finite domains. In addition to tokens representing equality of variables, there are tokens that that restrict the range of a variable to some finite set.

In the rest of this paper we will assume that we are working in some constraint system $\langle D, \vdash, \mathbf{Var}, \mathbf{RandVar}, \{\exists_X \mid X \in \mathbf{Var}\}, \mathtt{false}\rangle$. We will let $a, b \ldots$ range over D. We use $u, v, w \ldots$ to range over constraints.

2.2 Modeling probability information

We model probability information using probability lattices. Given a set A let $L(A)$ be the free profinite lattice generated by it. In the following definition, one should think of A as the fixed-point set of constraints, and think of C as mapping each $x \in L(A)$ to the sum of the probabilities associated with the RV-valuations realizing x, where an RV-valuation is considered to realize $\bigvee\{x_i\}$ if it realizes some x_i and realize $\bigwedge\{x_i\}$ if it realizes each x_i. For instance, with this interpretation, the modularity condition below is seen to correspond to taking the union of two overlapping sets.

Definition 2. A probability lattice C on A is a function from $L(A) \to [0,1]$ satisfying:

Monotonicity: $x \leq y \Rightarrow C(x) \leq C(y)$.
Join Continuity: If $\{x_i\}$ is a directed set then $C(\vee x_i) = \textbf{sup}\, C(x_i)$.
Meet Continuity: If $\{x_i\}$ is a filter then $C(\wedge_i x_i) = \textbf{inf}\, C(x_i)$.
Normality: $C(\vee A) = 1$.
Modularity: $C(x \vee y) = C(x) + C(y) - C(x \wedge y)$.
Extensionality: If $C(x) = C(x') = C(x \vee x')$, then $(\forall y)\ [C(x \vee y) = C(x' \vee y)]$.

In what follows we will find useful the following definitions: For a probability lattice C on A, define $u \preceq_C v$ if $C(u) = C(u \vee v)$; intuitively, the set of RV-valuations realizing u is contained in the set of RV-valuations realizing v. Let $\equiv_C$ be the associated symmetric closure. Note that by extensionality, $\equiv_C$ is an equivalence relation. Say that $S \subseteq A$ is *C-consistent* if $C(\bigwedge S) > 0$; intuitively there is at least one RV-valuation that jointly realizes every element of a consistent set. Note that to specify a probability lattice f on A, it is enough to specify the values of f on all finite meets (or joins) of A; such an f can be uniquely extended (via monotonicity, continuity, modularity and extensionality) to a map on $L(A)$.
We define the following operations on probability lattice.

Quotient. Suppose C is a probability lattice on A, and $h : A \to B$ is a function. If $C(\vee(h^{-1}(B))$ is greater than 0, define $\frac{C}{h}$, the *quotient* of C by h, as follows. For b in the image of h, let $h^{-1}(y) = \bigvee\{x \in A \mid h(x) = y\}$ and freely extend it to the lattice $L(h(A))$ by $h^{-1}(y \wedge y') = h^{-1}(y) \wedge h^{-1}(y')$ and so on. (Alternatively we could have extended h to a homomorphism between $L(A)$ and $L(h(A))$, and then defined h^{-1}.) We can *quotient* C to a probability lattice C' on $h(A)$, by defining $C'(b) = C(h^{-1}(b))$. C' inherits monotonicity, modularity and extensionality from h^{-1}. Multiplying all values of $C'(b)$ by $\frac{1}{C(\vee h^{-1}(B))}$ normalizes the resulting probability lattice.

Expansion. Suppose C is a probability lattice on B, and $h : A \to B$ is a partial function. If $C(\vee h(A)) > 0$, define $C' = \texttt{Exp}(C, h)$, the *expansion* of C by h, as follows. C' is a probability lattice on $\texttt{dom}(h)$ (the domain of h) given by $C'(x) = C(h(x)), C'(\vee\{x_i\}) = C(\vee\{h(x_i)\}), C'(\wedge\{x_i\}) = C(\wedge\{h(x_i)\})$ etc. Multiplying all values of $C'(a)$ by $\frac{1}{C(\vee h(A))}$ normalizes the resulting probability lattice.

Product. The *product* of two probability lattices C_1 on A_1 and C_2 on A_2 is a probability lattice C on $A_1 \times A_2$. $C(\langle x, y\rangle) = C_1(x) \times C_2(y)$ for $\langle x, y\rangle \in A_1 \times A_2$. Similarly $C(\langle x, y\rangle \wedge \langle x', y'\rangle) = C_1(x \wedge x') \times C_2(y \wedge y')$; use modularity and continuity to define C on all elements of $L(A_1 \times A_2)$.

We can now define a process:

Definition 3 Process. A pcc *process* P is a pair (P_A, P_C) where $P_A \subseteq |D| - \{\texttt{false}\}$ and P_C is a probability lattice on P_A that satisfies the following conditions:

Consistency: For every $u \in P_A$, $\{u\}$ is P_C-consistent, i.e. $P_C(u) > 0$.
Glb-closure: For every P_C-consistent subset S of P_A, $\sqcap S \in P_A$ and $\sqcap S \preceq_{P_C} \bigwedge S$.
Finiteness: The number of $\equiv_{P_C}$-equivalence classes are finite.

The first condition ensures that every fixed-point is consistent (generated by at least one RV-valuation). The second forces every P_C-consistent set S of fixed-points to have a glb that is realized by at least every RV-valuation which realizes S; this ensures that every P_C-consistent set of fixed-points can be consistently extended to the range of a closure operator. The third condition forces finiteness of the total number of possible closure operators that can thus be generated.

2.3 Semantics of combinators

Tell. $\mathcal{P}[\![c]\!]_A \stackrel{d}{=} \{u \in |D| \mid c \in u\}$. $\mathcal{P}[\![c]\!]_C$ is the constant function 1.

Ask. $\mathcal{P}[\![\textbf{if } c \textbf{ then } A]\!]_A \stackrel{d}{=} \{u \in |D| \mid c \in u \Rightarrow u \in \mathcal{P}[\![A]\!]_A\}$. If $c \notin u$, then $\mathcal{P}[\![\textbf{if } c \textbf{ then } A]\!]_C(u) = 1$, otherwise $\mathcal{P}[\![\textbf{if } c \textbf{ then } A]\!]_C(u) = \mathcal{P}[\![A]\!]_C(u)$. This is extended to the rest of the lattice in the usual way.

Parallel Composition. $\mathcal{P}[\![A_1, A_2]\!]_A \stackrel{d}{=} \mathcal{P}[\![A_1]\!]_A \cap \mathcal{P}[\![A_2]\!]_A$. The probability lattice is defined as $\mathtt{Exp}(\mathcal{P}[\![A_1]\!]_C \times \mathcal{P}[\![A_2]\!]_C, \Delta)$ where $\Delta : |D| \rightarrow |D| \times |D|$ is the diagonal function.

Hiding. $\mathcal{P}[\![\textbf{new } X \textbf{ in } A]\!]_A \stackrel{d}{=} \{u \in |D| \mid \exists v \in \mathcal{P}[\![A]\!], \exists_X u = \exists_X v\}$. The probability lattice is defined as $\mathtt{Exp}(\frac{\mathcal{P}[\![A]\!]_C}{\exists_X}, \exists_X)$, where $\exists_X : |D| \rightarrow |D|$ is the existential function on constraints from the given constraint system.

Distributions. Let f be a probability lattice of the domain of X[4]. Define $\mathtt{Distr}(X, f)_A = \{u \in |D| \mid \exists r.(X = r) \in u, f(r) > 0\}$ and $\mathtt{Distr}(X, f)_C = \mathtt{Exp}(f, h)$, where $h : \mathtt{Distr}(X, f)_A \rightarrow f$, and $h(u) = r$ if $(X = r) \in u$.

Now $\mathcal{P}[\![\textbf{new } (X, f) \textbf{ in } A]\!] \stackrel{d}{=} \mathcal{P}[\![\textbf{new } X \textbf{ in } (\mathtt{Distr}(X, f), P)]\!]$.

Conservativity results. Let A be a cc program. Then, $\mathcal{P}[\![A]\!] = (Q, C)$, where Q is the set of quiescent points of A and C is the constant function 1. Conservativity of pcc over cc follows immediately.

2.4 Operational semantics

We define a transition relation to give the operational semantics of pcc programs, and then show that this operational semantics is equivalent to the denotational semantics. The transition relation is similar to the transition relation for cc. We assume that the program is operating in isolation —interaction with the environment can be coded as an observation and run in parallel with the program. A configuration is a multiset of agents Γ. $\sigma(\Gamma)$ is the lub of the tell constraints in Γ.

$$\frac{\sigma(\Gamma) \vdash a}{\Gamma, \textbf{if } a \textbf{ then } B \longrightarrow \Gamma, B} \qquad \Gamma, (A, B) \longrightarrow \Gamma, A, B$$

$$\Gamma, \textbf{new } X \textbf{ in } A \longrightarrow \Gamma, A[Y/X] \quad (Y \text{ not free in } \Gamma)$$

$$\frac{f(r) > 0}{\Gamma, \textbf{new } (X, f) \textbf{ in } A \longrightarrow \Gamma, Y = r, A[Y/X]} \quad (Y \text{ not free in } \Gamma)$$

[4] A probability mass function on a finite set extends to a probability lattice structure.

Consider $\{\Gamma_i \mid A, a \longrightarrow^* \Gamma_i \not\longrightarrow, \sigma(\Gamma_i) \not\approx \texttt{false}\}$. Define a probability lattice C on Γ_i by setting $C(\Gamma_i) = \Pi_{\mathbf{Y}} f_Y(r_Y)$, where the $\mathbf{Y}$ is the set of new variables introduced by the last rule, $Y = r_Y \in \Gamma_i$, and f_Y is the probability distribution corresponding to Y. $C(\Gamma_i \wedge \Gamma_j) = 0$, so we get a probability lattice by modularity and normalization.

The output of a process P on an input a, denoted $\texttt{OpsemIO}(P, a)$ is

$$\{\exists_{\mathbf{Y}} \sigma(\Gamma) \mid P, a \longrightarrow^* \Gamma \not\longrightarrow, \sigma(\Gamma) \not\approx \texttt{false}, \mathbf{Y} \text{ new vars in derivation}\}$$

with the probability lattice structure given by $\frac{C}{h}$, where $h(\Gamma) = \exists_{\mathbf{Y}} \sigma(\Gamma)$.

The operational semantics of a process P, denoted $\texttt{Opsem}(P)$ is a pair (P_A, P_C) where P_A is the set of quiescent points of the process P, *i.e*

$$P_A = \{c \mid P, a \longrightarrow^* \Gamma \not\longrightarrow, \exists_{\mathbf{Y}} \sigma(\Gamma) = c, \mathbf{Y} \text{ new vars in derivation}\}$$

P_C is the natural probability lattice structure on P_A induced by the valuations of (hidden) random variables correspoding to each quiescent point.

2.5 Correspondence Theorems

The key ingredient of the correspondence theorems relating the operational and denotational semantics is a representation theorem on pcc processes, sketched below. We show that each process corresponds to a set of closure operators with associated probability, and conversely, given such a set, we can recover a process.

Let P be a process. A *consistent closure operator* (cco) of P is any P_C-consistent subset of P closed under glbs of non-empty subsets. Note that to every cco S there corresponds at least one RV-valuation jointly realizing S. However, it could be that the valuation also realizes other constraints, that is, it realizes a cco $T \supset S$. Intuitively, the cco's exhibited by P are going to be those cco's S for which there is an RV-valuation realizing S and not any cco $T \supset S$. Such cco's can be determined as follows. Let $p(S) = P_C(\bigwedge S) - P_C(\bigvee\{\bigwedge T \mid T \supset S\})$. If $p(S) > 0$ then S is an exhibited cco. The probability of S is $p(S)$.

Conversely, given a set Z of closure operators, each associated with a probability $p : Z \rightarrow [0, 1]$, we can recover a process P as follows. $P_A = \bigcup\{S \mid S \in Z\}$. P_C is defined as follows. For any finite subset V of P_A, let $P_C(\wedge V) = \Sigma\{p(S) \mid S \in Z, S \supseteq V\}$. P_C can be extended to be a probability lattice in the usual way.

The input-output relation. The representation theorem permits the *semantic* recovery of the input output relation from a process P. Let c be an input. P will associate with c an output o iff there is closure operator f exhibited by P which maps c to o. The probability associated with o is the (normalized) sum of the probabilities associated with each closure operator that maps c to o.

Full abstraction. The operational and denotational semantics are equivalent. The proof exploits the representation theorem sketched above, and is omitted for lack of space.

Theorem 4. Computational Adequacy: *For any* pcc *program* A, $\texttt{IO}(\mathcal{P}[\![A]\!])(u) = \texttt{OpsemIO}(A, u)$

Full Abstraction: *For any* pcc *programs* A_1, A_2, *if* $\mathcal{P}[\![A_1]\!] \neq \mathcal{P}[\![A_2]\!]$ *then there is a context* C *such that* $\texttt{Opsem}(C[A_1]) \neq \texttt{Opsem}(C[A_2])$.

3 Adding time — Timed pcc

Timed pcc arises from pcc by adding a notion of discrete time. In adding time, we would also like to keep the characteristic properties of synchronous programming languages alluded to earlier. We ensure this by extending pcc to Timed pcc using the same method used to extend Default cc to Default tcc in [SJG, SJG94]. Concretely, we add a single temporal construct **hence** A — when this is invoked at time t, then a new copy of A is started at each time instant $t' > t$.

Notation. We will be working with sequences, *i.e.* partial functions on the natural numbers — their domains will be initial segments of the natural numbers of the form $0..n$. We let $s, t, s', s'', \ldots$ denote sequences. We use ϵ to denote the empty sequence. The concatenation of sequences is denoted by "$\cdot$"; for this purpose a singleton u is regarded as the one-element sequence $\langle u \rangle$. Given a subset of sequences S, and a sequence s, we will write S **after** s for the set $\{t \in \textbf{Obs} \mid s \cdot t \in S\}$. $s(n)$ denotes its n'th element of s. We also define $S(0) = \{u \mid \exists s.\ u \cdot s \in S\}$.

We define a *sequence algebra* with the signature $\langle S, \text{Pref}_i, \text{length}() \rangle$, where S is the set of all sequences, length$(s) : S \rightarrow \mathbb{N}$ and $\text{Pref}_i : S \rightarrow S$, $\text{Pref}_i(s) = s$ if length$(s) \leq i$, otherwise $\text{Pref}_i(s) = \langle s(0), s(1), \ldots, s(i-1) \rangle$, the sequence consisting of the first i elements of s. Homomorphisms on sequence algebras will preserve prefixes and lengths.

Denotational Model. An observation of a Timed pcc program is a *quiescent sequence* of a program. Let **Obs** be the set of all finite sequences of consistent(*i.e.* not `false`) constraints. A process is a collection of observations that satisfies the condition that instantaneous execution at any time instant is modeled by a pcc process. The probability information is kept as a probability lattice for each sequence s in the process — this information is interpreted as the conditional probability information associated with the process at the first instant after the history s. Formally, we proceed as follows.

Definition 5. P is a Timed pcc process if $P_A \subseteq$ **Obs**, and for each $s \in P_A$ we are given a probability lattice P_s, satisfying the following conditions:

Non-emptiness: $\epsilon \in P_A$,
Prefix-closure: $s \in P_A$ whenever $s \cdot t \in P_A$,
Point execution: $(\forall s \in P_A)$, $((P_A$ **after** $s)(0), P_s)$ is a pcc process.

We can combine the probability lattices into a single indexed set of probability lattices by defining $P_C(s \cdot z) \stackrel{d}{=} P_s(z)$, and similarly for joins and meets in P_s.

Probability Lattice operations on P_C. We generalize the definitions of operations on probability lattices to indexed sets of probability lattices.

Quotienting. Let $h : A \rightarrow B$ be a sequence algebra homomorphism. Let P be an indexed set of probability lattices on A, we want to define an indexed set Q on the image of h. We will do this in an inductive fashion on the tree of sequences. The basic idea is that we will at each stage collapse the sequences identified by h, and assume that for any two sequences $t, t' \in A$, if $t \neq t'$ then the probability lattices P_t and $P_{t'}$ are independent.

For each sequence $s \in B$ we will also define inductively a probability lattice R^s on the set $\{t \cdot v \in A \mid h(t) = s\}$. R^s will be quotiented to produce Q_s.

Define $R^\epsilon = P_\epsilon$ and $Q_\epsilon = \frac{R^\epsilon}{h}$.

Assume we have defined Q_s and R^s for $s \in B$. Let $s \cdot u \in B$. If $h(t \cdot v) = s \cdot u$, define $R^{s \cdot u}(t \cdot v \cdot w) = R^s(t \cdot v) \times P_{t \cdot v}(w)$. Also, $R^{s \cdot u}(t \cdot v \cdot w \wedge t \cdot v \cdot w') = R^s(t \cdot v) \times P_{t \cdot v}(w \wedge w')$. If $h(t' \cdot v') = s \cdot u$, and $t' \cdot v' \neq t \cdot v$ then $R^{s \cdot u}(t \cdot v \cdot w \wedge t' \cdot v' \cdot w') = R^s(t \cdot v \wedge t' \cdot v') \times P_{t \cdot v}(w) \times P_{t' \cdot v'}(w')$. This follows from the independence assumption. Now $Q_{s \cdot u} = \frac{R^{s \cdot u}}{h'}$, where $h'(t \cdot v \cdot w) = u'$ if $h(t \cdot v \cdot w) = s \cdot u \cdot u'$.

Expansion. Let $h : A \to B$ be a partial sequence algebra homomorphism, i.e. the domain of h is a subalgebra of A. Let Q be an indexed set of probability lattices on B. Then we define an indexed set of probability lattices P on the domain of h by $P_s = \mathtt{Exp}(Q_s, h')$, where $h'(u) = h(s \cdot u)$.

Product. Let P_1 and P_2 be two indexed sets of probability lattices on A_1 and A_2. Define the fibered product of A_1 and A_2 as the set $\{s \cdot \langle u, v \rangle \mid s \cdot u \in A_1, s \cdot v \in A_2\}$. Define Q on this fibered product as $Q_s = P_{1_s} \times P_{2_s}$.

We will overload the symbols for quotient, expansion and product to stand for the corresponding operations on indexed sets of probability lattices also.

Combinators of Timed pcc. c, **if** c **then** A, (A, B) are inherited from pcc and their denotations are induced by their pcc definitions.

Tell. $\mathcal{D}[\![a]\!]_A \stackrel{d}{=} \{\epsilon\} \cup \{u \cdot s \in \mathbf{Obs} \mid a \in u\}$. $\mathcal{D}[\![a]\!]_C = 1$.

Ask. $\mathcal{D}[\![\mathbf{if}\ a\ \mathbf{then}\ A]\!]_A \stackrel{d}{=} \{\epsilon\} \cup \{u \cdot s \in \mathbf{Obs} \mid a \in u \Rightarrow u \cdot s \in \mathcal{D}[\![A]\!]_A\}$. For any sequence $u \cdot s$, if $a \in u$ then $\mathcal{D}[\![\mathbf{if}\ a\ \mathbf{then}\ A]\!]_C(u \cdot s) = \mathcal{D}[\![A]\!]_C(u \cdot s)$, otherwise $\mathcal{D}[\![\mathbf{if}\ a\ \mathbf{then}\ A]\!]_C(u \cdot s) = 1$. The rest of $\mathcal{D}[\![\mathbf{if}\ a\ \mathbf{then}\ A]\!]_C$ is defined by monotonicity, continuity and modularity.

Parallel Composition. $\mathcal{D}[\![A, B]\!]_A \stackrel{d}{=} \mathcal{D}[\![A]\!]_A \cap \mathcal{D}[\![B]\!]_A$. $\mathcal{D}[\![A, B]\!]_C$ is given as before by $\mathtt{Exp}(\mathcal{P}[\![A_1]\!]_C \times \mathcal{P}[\![A_2]\!]_C, \Delta)$ where $\Delta(\epsilon) = \epsilon$ and $\Delta(s \cdot u) = s \cdot \langle u, u \rangle$.

Hiding. Every observation $s \in \mathcal{D}[\![\mathbf{new}\ X\ \mathbf{in}\ A]\!]_A$ is induced by some observation $s' \in \mathcal{D}[\![A]\!]_A$, *i.e.* at every time instant t, $s(t)$ must equal the result of hiding X in the pcc process given by A at time t after history s^{t-1}.

Formally, let $\exists_X s = \exists_X s'$ denote $|s| = |s'|$, and $\forall i < |s|, \exists_X s(i) = \exists_X s'(i)$. Then

$$\mathcal{D}[\![\mathbf{new}\ X\ \mathbf{in}\ A]\!] \stackrel{d}{=} \{s \in \mathbf{Obs} \mid \exists s' \in \mathcal{D}[\![A]\!].\exists_X s = \exists_X s'\}$$

The set of probability lattices is defined as $\mathtt{Exp}(\frac{\mathcal{D}[\![A]\!]_C}{\exists_X}, \exists_X)$, where $\exists_X : |D| \to |D|$ is the existential on sequences.

Distributions. $\mathtt{Distr}(X, f)$ ensures that X follows distribution f for all time, *i.e.* after every sequence, we must get the pcc process $\mathtt{Distr}(X, f)$. Thus, we have: $\mathtt{Distr}(X, f)_A = \{s \in \textbf{Obs} \mid \forall i < \text{length}(s).\ \exists r_i.\ (X = r_i) \in s(i), f(r_i) > 0\}$, and $\mathtt{Distr}(X, f)_C = \mathtt{Exp}(F, h)$, where F is an indexed set of lattices defined by $F(s) = f$, $h(s \cdot u) = h(s) \cdot r$ if $(X = r) \in u$. As in pcc, define

$$\mathcal{D}[\![\textbf{new } (X, f) \textbf{ in } A]\!] \stackrel{d}{=} \mathcal{D}[\![\textbf{new } X \textbf{ in } (\mathtt{Distr}(X, f), A)]\!]$$

Hence. The definition for **hence** is as expected — observations have to "satisfy" A everywhere after the first instant; the probability lattice codes the fact that at time t there are $t - 1$ copies of B running in parallel.

$$\mathcal{D}[\![\textbf{hence } B]\!]_A \stackrel{d}{=} \{u \cdot s \in \textbf{Obs} \mid (\forall s_1, s_2) s = s_1 \cdot s_2 \Rightarrow\ s_2 \in \mathcal{D}[\![B]\!]_A\}$$

$\mathcal{D}[\![\textbf{hence } B]\!]_C(u \cdot s) = 1$ if $s = \epsilon$, otherwise $\mathcal{D}[\![\textbf{hence } B]\!]_C(u \cdot s) = \Pi\{\mathcal{D}[\![B]\!]_C(s_2) \mid s = s_1 \cdot s_2\}$.

Definable combinators. Since all the basic combinators of Default tcc[SJG] are available here, (probabilistic approximations to) all the defined combinators of Default tcc are definable in Timed pcc demonstrating that Timed pcc is an expressive synchronous language.

next $_\epsilon A$	start A at the next instant with probability $1 - \epsilon$
first a **then** $_\epsilon A$	whenever a becomes true start A with probability $1 - \epsilon$
do A **watching** $_\epsilon c$	do A with probability $1 - \epsilon$ at each instant, until c becomes true
time $_\epsilon A$ **on** c	do A with probability $1 - \epsilon$ during the instants c holds

Operational semantics. The operational semantics for Timed pcc is built on the operational semantics for pcc — in this section, we focus on the aspects of the transition system that involve time. Following the synchronous paradigm, values of variables are not carried across time — in particular, a fresh "coin toss" is performed for random variables at each time instant.

A configuration consists of a pair(Γ, Δ) — the agents currently active and the "continuation" — the program to be executed at subsequent times. The rules for asks, hiding and parallel composition and new random variables are instantaneous and remain as before, in each case the Δ is unchanged after the transition. The rule for **hence** is also instantaneous and is given by

$$((\Gamma, \textbf{hence } A), \Delta) \longrightarrow (\Gamma, (A, \textbf{hence } A, \Delta))$$

The instantaneous outputs of Γ and the associated probability lattice C is derived as in pcc.

For a given possible instantaneous output u, let $\Delta_1 \ldots \Delta_n$ be such that $(\Gamma, \emptyset) \longrightarrow^* (\Gamma'_i, \Delta_i) \not\longrightarrow$, $\mathbf{Y}_i$ are the new variables introduced in derivation, $\exists_{\mathbf{Y}_i} \sigma(\Gamma'_i) = u$. Also, let $f = \frac{C}{h}$, where $h((\Gamma_i, \Delta_i)) = i$. For output u, the timestep transition relation $\leadsto$ is defined as

$$\Gamma \rightsquigarrow \textbf{new}\ (X, f)\ \textbf{in}\ (\ \textbf{if}\ X = 1\ \textbf{then new}\ \mathbf{Y}_1\ \textbf{in}\ \Delta_1,\\ \ldots\\ \textbf{if}\ X = n\ \textbf{then new}\ \mathbf{Y}_n\ \textbf{in}\ \Delta_n)$$

In the above, for all random variables in $\mathbf{Y}_i$, we write **new** (Y, g) **in** Δ_i, where g was the distribution in the prior time instant.

Correspondence theorems. As in pcc, the operational and denotational semantics are equivalent. The proof is essentially just a "lifting" of the proof for pcc, and is omitted lack of space.

Acknowledgements. We would like to thank Lise Getoor for bringing up the problem of probability modeling, and Daphne Koller, Moses Charikar and Fernando Pereira, for several discussions. This work was supported by grants from NSF, ARPA and ONR.

References

[Abr93] S. Abramsky. Interaction categories. Available by anonymous ftp from papers/Abramsky:theory.doc.ic.ac.uk, 1993.

[Abr94] Samson Abramsky. Interaction categories and communicating sequential processes. In A. W. Roscoe, editor, *A Classical Mind: Essays in honour of C. A. R. Hoare*, pages 1–16. Prentice Hall International, 1994.

[BB91] A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. In *Special issue on Another Look at Real-time Systems*, Proceedings of the IEEE, September 1991.

[BLFG95] Albert Benveniste, Bernard C. Levy, Eric Fabre, and Paul Le Guernic. A calculus of stochastic systems for the specification, simulation, and hidden state estimation of mixed stochastic/nonstochastic systems. *Theoretical Computer Science*, 152(2):171–217, Dec 1995.

[CSZ92] R. Cleaveland, S. A. Smolka, and A. Zwarico. Testing preorders for probabilistic processes. *Lecture Notes in Computer Science*, 623, 1992.

[dKW89] Johan de Kleer and Brian C. Williams. Diagnosis with behavioral modes. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1324–1330, August 1989.

[FBB+94] M. Fromherz, D. Bell, D. Bobrow, et al. RAPPER: The Copier Modeling Project. In *Working Papers of the Eighth International Workshop on Qualitative Reasoning About Physical Systems*, pages 1–12, June 1994.

[FJM90] R. Fagin, J.Y.Halpern, and N. Megiddo. A logic for reasoning about probabilities. *Information and Computation*, 87:78–128, 1990.

[FS95] Markus P.J. Fromherz and Vijay A. Saraswat. Model-based computing: Using concurrent constraint programming for modeling and model compilation. In U. Montanari and F. Rossi, editors, *Principles and Practice of Constraint Programming - CP'95*, pages 629–635. Springer-Verlag, LNCS 976, Sept. 1995.

[GJS] Vineet Gupta, Radha Jagadeesan, and Vijay Saraswat. Computing with continuous change. *Science of Computer Programming*. To appear.

[GSS95] Vineet Gupta, Vijay Saraswat, and Peter Struss. A model of a photocopier paper path. In *Proceedings of the 2nd IJCAI Workshop on Engineering Problems for Qualitative Reasoning*, August 1995.

[Hal93] N. Halbwachs. *Synchronous programming of reactive systems*. The Kluwer international series in Engineering and Computer Science. Kluwer Academic publishers, 1993.

[Har87] D. Harel. Statecharts: A visual approach to complex systems. *Science of Computer Programming*, 8:231 – 274, 1987.

[HJ90] H. Hansson and B. Jonsson. A calculus for communicating systems with time and probabilities. In *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pages 278–287. IEEE Computer Society Press, 1990.

[HS86] S. Hart and M. Sharir. Probabilistic propositional temporal logics. *Information and Control*, 70:97–155, 1986.

[HSD92] Pascal Van Hentenryck, Vijay A. Saraswat, and Yves Deville. Constraint processing in cc(fd). Technical report, Computer Science Department, Brown University, 1992.

[Joh82] P. T. Johnstone. *Stone Spaces*, volume 3 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1982.

[JP89] C. Jones and G. D. Plotkin. A probabilistic powerdomain of evaluations. In *Proceedings, Fourth Annual Symposium on Logic in Computer Science*, pages 186–195, Asilomar Conference Center, Pacific Grove, California, 1989.

[JY95] Bengt Jonsson and Wang Yi. Compositional testing preorders for probabilistic processes. In *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 431–441, San Diego, California, 1995.

[LS91] Kim G. Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, September 1991.

[Mar89] M. Ajmone Marsan. Stochastic petri nets: an elementary introduction. In *Advances in Petri Nets 1989*, pages 1–29. Springer, June 1989.

[MMS96] Carroll Morgan, Annabelle McIver, and Karen Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, May 1996.

[Nil86] N.J. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28:71–87, 1986.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan-Kaufmann Publishers, 1988.

[Sar93] Vijay A. Saraswat. *Concurrent constraint programming*. Doctoral Dissertation Award and Logic Programming Series. MIT Press, 1993.

[SJG] V. A. Saraswat, R. Jagadeesan, and V. Gupta. Timed Default Concurrent Constraint Programming. *Journal of Symbolic Computation*. To appear. Extended abstract appeared in the *Proceedings of the 22nd ACM Symposium on Principles of Programming Languages*, San Francisco, January 1995.

[SJG94] V. A. Saraswat, R. Jagadeesan, and V. Gupta. Foundations of Timed Concurrent Constraint Programming. In Samson Abramsky, editor, *Proceedings of the Ninth Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Press, July 1994.

[SRP91] V. A. Saraswat, M. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *Proceedings of Eighteenth ACM Symposium on Principles of Programming Languages, Orlando*, January 1991.

[vGSST90] Rob van Glabbeek, S. A. Smolka, B. Steffen, and C. M. N. Tofts. Reactive, generative, and stratified models of probabilistic processes. In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 130–141, Philadelphia, Pennsylvania, 1990.

[VN92] N. Viswanadham and Y. Narahari. *Performance Modeling of Automated Manufacturing Systems*. Prentice-Hall Inc., 1992.