A smooth probabilistic extension of concurrent constraint programming

Romain Beauxis
INRIA Futurs and LIX,
Ecole Polytechnique, France
beauxis@lix.polytechnique.fr

1. Introduction

CONCURRENT constraint programming (CCP [1]) is a model of computation in which the information available to the process is represented by the notion of constraint. Each process has access to a global store, with respect to which it tests and adds constraints. A domain-theoretic denotational semantics for CCP has been defined in [2]. In this semantics a process is mapped to the set of resting points that it can reach. It is possible to compute this set by a fixed point construction. This paper studies an extension of CCP with probabilistic executions. We develop a mathematical framework for defining the meaning of an infinite probabilistic execution. We introduce a denotation semantics. The CCP+P language is defined by the following grammar, and enjoys the expected properties.

Definition 1 The CCP+P language is defined by the following grammar, where the e’s and i’s represent constraints.

Agent A ::= 0 | e | A · A | 3X (c, A) ♦([A], c, p, p) | p(X)
Procedure D ::= e | p(X) : A | D ; D
Process P ::= D . A

A CCP+P configuration is a set of elements of the form C = ∪(i(P, c, p)). [1(C)] = {x(∑(P, c, p))} is its projection on the constraints lattice.

3. Example: Crowds routing

CROWDS is a probabilistic anonymity protocol. It features a pool of agents, called the forwarders. When an agent wants to send a message to another agent, it sends the message to one forwarder of the pool. Then, the forwarder decides with probability p to deliver the message to its destination, or with probability 1 − p to forward it again to another node in the pool.

4. Decomposition of valuations

The state of a probabilistic program during its execution is represented by a valuation. A valuation is a function that measures the opens of a topology. There are open valuations that can be ordered. During a computation, the valuation representing the current state of the program only increases. In order to represent the outcome of an infinite run, we need to consider infinite directed sequences of valuations. These sequences are known to converge to a valuation. However, in order to decompose the probabilities of the individual states, we need that this result is a simple valuation.

Definition 2 Valuations:

(1) µ(∅) = 0
(2) µ(∅) + µ(U) = µ(U ∪ U) + µ(U ∩ U)
(3) O ⊆ U ⇒ µ(O) ≤ µ(U)
(4) µ(X) = ° ∞

Dirac valuations:

δ_x : O → {0, 1, if x ∈ O
otherwise

Simple valuation:

v = ∑ a_δ_x

We study the conditions under which all valuations are simple valuations. That way, every computation can be interpreted as a linear mapping from simple valuations to simple valuations, even when running an infinite sequence of operations.

Definition 3 A lattice X, is totally well-ordered, written WOL, if and only if there exists a well-ordered sequence (x_i) such that x_i ≤ x_j ⇒ i ≤ j and υ∪(x_i) = X

Theorem 1 Any valuation on a WOL equipped with the Scott topology is a simple valuation.

Hence, any valuation can be represented as a vector on the vector space generated by the simple valuations. Furthermore, the denotational semantics of our programs is a linear closure operator:

Definition 4 An operator p on a partially ordered set is a closure operator if x ≤ p(x), p(p(x)) = p(x) and x ≤ y ⇒ p(x) ≤ p(y).

These operators can be identified to their image on the cone of simple valuations: Let C^*(B) = {∑_a b_aj | a_j ≥ 0, ∑_a _j a_j < ° ∞, b_j ∈ B} be the set of all possible positive linear combinations of vectors from B. Let p be a linear closure operator and x a valuation, then p(x) = min{ x ∩ C^*(p(d)) | d constraint}

Theorem 2 The mapping: ψ : ∑_a p_δ_x = d, where d is the result of an infinite fair computation of υ∪((P, c, p)), is a linear closure operator.

5. Denotational semantics

We propose a denotational semantics where a process is represented by a linear closure operator. This operator is identified to the cone of its fixed points. These fixed points are simple valuations.

• [0] = E
• [c] = C^*(δ_xc) for all possible d
• [c → p] = C^*(δ_xc | d ≥ c) ∪ C^*(p(d)) | d ≥ c
• [A]B = [A] ∩ [B]
• [∪_{(P, c, p)}] = C^*(∑_p f_p((P(x)))δ_x)

We prove the correspondence with the operational semantics: this semantics is sound and fully abstract.

Theorem 3.

[ρ] = ρ_p
[ρ] = [Q] if and only if for any context C, ρ(C) = ρ(C)

References
