# Probabilistic Security Analysis in a Declarative Framework

Alessandra Di Pierro[1], Chris Hankin[2], and Herbert Wiklicky[2]

[1] Dipartimento di Informatica, Universitá di Pisa, Italy
[2] Department of Computing, Imperial College, London, UK

## 1 The Approach

The aim of this research is to develop a methodology for the security analysis of programs which resembles the classical program analysis methods, e.g. [4]. Our investigations consider probabilistic languages and their semantics as well as a probabilistic semantics for (non)deterministic languages. In order to accommodate both Shannon's notion of "information" — basically a probabilistic/statistical concept — and Scott's notion of "information" — essentially an order-theoretic concept — we look at a probabilistic extension of a declarative programming language.

In the following we will concentrate on a particular declarative programming paradigm, Concurrent Constraint Programming (CCP) [7, 6], and a probabilistic version of this paradigm previously introduced by the authors in [2, 3], where it was called Probabilistic Concurrent Constraint Programming (PCCP). The syntax and the basic execution model of both CCP and PCCP are very similar. Both languages are based on the notion of a generic *constraint system* $\mathcal{C}$, defined as a cylindric algebraic cpo (see [7] for more details), which encodes the information ordering. Probabilities are introduced via a probabilistic choice $\Box_{i=1}^{n}\ \mathbf{ask}(c_i) \rightarrow p_i : A_i$ and a prioritised parallel construct $\|_{i=i}^{n}\ q_i : A_i$ (see [2, 3] for further details). In the following we restrict our attention to binary versions of these two operators.

## 2 Probabilistic Concurrent Constraint Programming

The syntax of a PCCP agent is given in Table 1, where $c$ and $c_i$ are *finite* constraints in $\mathcal{C}$. We will assume that the contraint system $\mathcal{C}$ is *countable*. A PCCP program $P$ is then an object of the form $D.A$, where $D$ is a set of procedure declarations of the form $p(x) : -A$ and $A$ is an agent.

The operational semantics of PCCP has a simple definition in terms of a probabilistic transition system, $(Conf, \longrightarrow_p)$, where $Conf$ is the set of configurations $\langle A, d\rangle$ representing the state of the system at a certain moment, namely the agent $A$ which has still to be executed and the common store $d$; and the transition relation $\longrightarrow_p$ is defined in Table 2. The rules given are closely related to the ones for (nondeterministic) CCP, and we refer to [1] for a detailed description.

| | | |
|---|---|---|
| $A ::=$ | **stop** | successfull termination |
| $\mid$ | $\mathbf{tell}(c)$ | adding a constraint |
| $\mid$ | $\Box_{i=1}^{2}\ \mathbf{ask}(c_i) \rightarrow p_i : A_i$ | probabilistic choice |
| $\mid$ | $\Vert_{i=i}^{2}\ q_i : A_i$ | prioritised paralellism |
| $\mid$ | $\exists_x A$ | hiding, local variables |
| $\mid$ | $p(x)$ | procedure call, recursion |

**Table 1.** The Syntax of PCCP Agents.

| | | |
|---|---|---|
| **R1** | $\langle \mathbf{tell}(c), d\rangle \longrightarrow_1 \langle \mathbf{stop}, c \sqcup d\rangle$ | |
| **R2** | $\langle \Box_{i=1}^{2}\ \mathbf{ask}(c_i) \rightarrow p_i : A_i, d\rangle \longrightarrow_{\tilde{p}_j} \langle A_j, d\rangle$ | $j \in [1,2]$ and $d \vdash c_j$ |
| **R3** | $\dfrac{\langle A_j, c\rangle \longrightarrow_p \langle A'_j, c'\rangle}{\langle \Vert_{i=1}^{2}\ p_i : A_i, c\rangle \longrightarrow_{p\cdot\tilde{p}_j} \langle \Vert_{j\neq i=1}^{2}\ p_i : A_i \parallel p_j : A'_j, c'\rangle}$ | $j \in [1,2]$ |
| **R4** | $\dfrac{\langle A, d \sqcup \exists_x c\rangle \longrightarrow_p \langle A', d'\rangle}{\langle \exists_x^d A, c\rangle \longrightarrow_p \langle \exists_x^{d'} A', c \sqcup \exists_x d'\rangle}$ | |
| **R5** | $\langle p(y), c\rangle \longrightarrow_1 \langle A, c\rangle$ | $p(x) : -A \in P$ |

**Table 2.** The transition system for PCCP.

Both rules **R2** and **R3** refer to normalised quantities $\tilde{p}_i$. This process can be defined for a generic set of real numbers $X = \{x_i\}$, as the process of replacing each $x_i$ by $\tilde{x}_i$ as follows:

**Definition 1.** Let $x = \sum_i x_i$, and $n$ the cardinality of $X = \{x_i\}$. Then

$$\tilde{x}_i = \begin{cases} \frac{x_i}{x} & \text{if } x \neq 0 \\ \frac{1}{n} & \text{otherwise} \end{cases}$$

## 3 The Analysis

Our analysis builds a rooted graph which approximates the stochastic matrix for the program. Each node in the graph is a pair consisting of a constraint and a real number. The root of the graph is $\langle true, 1\rangle$. An approximation of the meaning of the program can be recovered by considering the maximal acyclic paths through the graph.

Let us now define the analysis formally, see Table 3. The analysis returns a set of triples. Each triple consists of a set of constraints, a single constraint and a probability. The first component (the set of constraints) specifies the (constraint part of) nodes that must appear on every path from the root to the node corresponding to the second two components. In Table 3 we have focussed on the choice and parallelism operators. Given a finite set of constraints, this gives the true semantics. The rules for hiding and calls will need to introduce approximation. In particular, the rule for calls will require some kind of widening to account for recursion; we postpone this to the full paper, which will be available at `www.di.unipi.it/~ dipierro/WITS/pccp.ps`.

$$
\begin{array}{ll}
[\![\mathbf{stop}]\!] & = \emptyset \\
[\![\mathbf{tell}(c)]\!] & = \{\langle \emptyset, c, 1\rangle\} \\
[\![ []_{i=1}^{2} \ \mathbf{ask}(c_i) \rightarrow p_i : A_i ]\!] & = (\langle \{c_1\}, \mathit{true}, 1\rangle \oplus [\![A_1]\!]) \cup \\
& \quad (\langle \{c_2\}, \mathit{true}, 1\rangle \oplus [\![A_2]\!]) \cup \\
& \quad (\langle \{c_1, c_2\}, \mathit{true}, p_1\rangle \oplus [\![A_1]\!]) \cup \\
& \quad (\langle \{c_1, c_2\}, \mathit{true}, p_2\rangle \oplus [\![A_2]\!]) \\
[\![ (\|_{i=i}^{2} \ q_i : A_i ]\!] & = (\langle \emptyset, \mathit{true}, q_1\rangle \oplus [\![A_1]\!]) \cup \\
& \quad (\langle \emptyset, \mathit{true}, q_2\rangle \oplus [\![A_2]\!]) \cup \\
& \quad \bigcup_{\langle a,b,\tilde{q_1}\rangle \in [\![A_1]\!]} (\langle a \ \cup \ \{b\}, \mathit{true}, 1\rangle \oplus [\![A_2]\!]) \cup \\
& \quad \bigcup_{\langle a,b,\tilde{q_2}\rangle \in [\![A_2]\!]} (\langle a \ \cup \ \{b\}, \mathit{true}, 1\rangle \oplus [\![A_1]\!])
\end{array}
$$

**Table 3.** The Analysis for PCCP Agents.

The semantics in Table 3 uses an auxiliary operator $\oplus$. Its definition is as follows:

$$\langle a, b, p\rangle \oplus X = \{\langle a \ \cup \ x, b \sqcup y, pq\rangle \ \mid \ \langle x, y, q\rangle \in X\}.$$

### 3.1 Declarative Confinement

As an example of how various security notions fit naturally in this declarative framework, let us look at "probabilistic confinement" as discussed by Volpano and Smith [8].

The following simple example illustrates how this notion is captured by our probabilistic language. Consider the PCCP agents:

$$A \equiv \frac{1}{2} : \mathbf{tell}(c) \parallel \frac{1}{2} : \mathbf{tell}(d)) \text{ and } B \equiv \mathbf{tell}(c \sqcup d)$$

as well as:

$$C \equiv \mathbf{ask}(c) \rightarrow \frac{2}{3} : \mathbf{tell}(e) \ [] \ \mathbf{ask}(d) \rightarrow \frac{1}{3} : \mathbf{tell}(f).$$

If we combine $A$ and $B$ with $C$ the corresponding probabilistic input/output observables are respectively:

$$\mathcal{O}_P \left( \frac{1}{2} : A \parallel \frac{1}{2} : C \right) = \{\langle c \sqcup d \sqcup e, \frac{7}{12}\rangle, \langle c \sqcup d \sqcup f, \frac{5}{12}\rangle\}$$

$$\mathcal{O}_P\left(\frac{1}{2} : B \parallel \frac{1}{2} : C\right) = \{\langle c \sqcup d \sqcup e, \frac{2}{3}\rangle, \langle c \sqcup d \sqcup f, \frac{1}{3}\rangle\}$$

Therefore, the context $C$ allows us to distinguish $A$ and $B$. This information can be precisely reconstructed using the analysis of Table 3.

Note that in the classical case of CCP (where all probability information is removed) the non-deterministic observables are the same for both $A \parallel C$ and $B \parallel C$:

$$\mathcal{O}_C(A \parallel C) = \mathcal{O}_C(B \parallel C) = \{c \sqcup d \sqcup e, c \sqcup d \sqcup f\}$$

One can see this as a basic example illustrating the difference between non-deterministic and probabilistic confinement in the sense of Volpano and Smith translated into a declarative framework.

## 4 Related and Further Work

Our work, in particular the example on probabilistic confinement presented above, is closely related not only to the work of Volpano and Smith, but also to recent results from Sands and Sabelfeld [5]. The main difference is that we concentrate on a declarative programming paradigm instead of considering imperative programming languages.

## References


1. Frank S. de Boer, M. Gabbrielli, E. Marchiori, and Catuscia Palamidessi. Proving Concurrent Constraint Programs Correct. *ACM Transactions on Programming Languages and Systems*, 1997.
2. Alessandra Di Pierro and Herbert Wiklicky. An operational semantics for Probabilistic Concurrent Constraint Programming. In P. Iyer, Y. Choo, and D. Schmidt, editors, *ICCL'98 – International Conference on Computer Languages*, pages 174–183. IEEE Computer Society Press, 1998.
3. Alessandra Di Pierro and Herbert Wiklicky. Probabilistic Concurrent Constraint Programming: Towards a fully abstract model. In L. Brim, J. Gruska, and J. Zlatuska, editors, *MFCS'98 – Mathematical Foundations of Computer Science*, volume 1450 of *Lecture Notes in Computer Science*, pages 446–455, Berlin – New York, August 1998. Springer Verlag.
4. Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Aanalysis*. Springer Verlag, Berlin – Heidelberg, 1999.
5. Andrei Sabelfeld and David Sands. Probabilistic noninterference for multi-threaded programs. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, page to appear, 2000.
6. V.A. Saraswat and M. Rinard. Concurrent constraint programming. In *Symposium on Principles of Programming Languages (POPL)*, pages 232–245. ACM, 1990.
7. V.A. Saraswat, M. Rinard, and P. Panangaden. Semantics foundations of concurrent constraint programming. In *Symposium on Principles of Programming Languages (POPL)*, pages 333–353. ACM, 1991.
8. Dennis Volpano and Geoffrey Smith. Confinement properties for programming languages. *SIGACT News*, 29(3):33–42, September 1998.