Electronic Notes in Theoretical Computer Science 48 (2001)
URL: http://www.elsevier.nl/locate/entcs/volume48.html pp. 1 – 23

Probabilistic Confinement in a Declarative Framework

Alessandra Di Pierro [1]

Dipartimento di Informatica
Universitá di Pisa
Pisa, Italy

Chris Hankin [2] Herbert Wiklicky [3]

Department of Computing
Imperial College
London, United Kingdom

Abstract

We show how to formulate and analyse some security notions in the context of declarative programming. We concentrate on a particular class of security properties, namely the so-called *confinement* properties. Our reference language is concurrent constraint programming. We use a probabilistic version of this language (PCCP) to highlight via simple program examples the difference between probabilistic and nondeterministic confinement. The different role played by variables in imperative and constraint programming hinders a direct translation of the notion of confinement into our declarative setting. Therefore, we introduce the notion of *identity confinement* which is more appropriate for constraint languages. Finally, we present an approximating probabilistic semantics which can be used as a base for the analysis of confinement properties, and show its correctness with respect to the operational semantics of PCCP.

1

The aim of this work is to investigate the security problem from a probabilistic viewpoint by formalising one aspect of the multi-form nature of this problem in a probabilistic programming language. The aspect we consider is *confidentiality*, that is the protection of some sensitive data against unauthorised

[1] Email: dipierro@di.unipi.it
[2] Email: clh@doc.ic.ac.uk
[3] Email: herbert@doc.ic.ac.uk

©2001 Published by Elsevier Science B. V. Open access under CC BY-NC-ND license.

disclosure. This aspect can be formalised in terms of the concept of *noninterference* [8] which states conditions for guaranteeing secure information flow throughout a computer system.

We propose a semantics-based formal specification of a noninterference property which ensures that a given program does not leak some private information which it is allowed to access. This property is often referred to as *confinement* after Lampson who first proposed the problem in the '70's [11].

Our specification is based on the Probabilistic Concurrent Constraint Programming language (PCCP), previously introduced in [3,4] as a probabilistic extension of concurrent constraint programming [10]. The probabilistic choice construct provided by PCCP allows us to take into account also attackers which are able to exploit some statistical information.

Previous approaches, notably the work by Volpano and Smith [19,17] and by Sands and Sabelfeld [14,13], consider imperative programming languages. Our choice of a declarative language, and in particular the PCCP language, is essentially motivated by the availability of a simple and mathematically sound semantics for this language which forms a base on which a systematic design of program analysis can be developed. In fact, the ultimate aim of this research is the development of a methodology for a probabilistic security analysis of (probabilistic) programs which resembles the classical program analysis methods [12].

As a first step in this direction, we present in this paper a control-flow analysis for PCCP programs which is able to detect whether a program is confined according to an appropriate notion of *declarative confinement*. The analysis refers to an abstract version of the derivation tree of a PCCP agent, where all possible transitions between stores are recorded together with their probabilities, and its correctness is shown with respect to the concrete operational semantics.

1.1 Confinement

An information processing system has a *confinement* property if it protects some important — so-called high level — information from non-authorised — so-called low level — access. That means the information, e.g. the state or value of a high level variable, does not "leak" out, i.e. influence low level information. In a more complex implementation of the standard security model [1] one may have several levels ("confident", "secret", "top secret", etc) but we consider here only the simple two levels case.

For example, a computer operating system might try to protect certain user information, e.g. passwords, etc. (high level information), from being accessed by some applications, e.g. spreadsheets, browsers, etc (low level applications), though it might well allow these applications (the low level) access to other data, e.g. the system time, etc. If the low level applications are unable to access this high level information, e.g. the password of a user, we can say that

it is confined.

Depending on the nature of the information flow between high and low level information, one can distinguish several types of confinement, namely *deterministic*, *nondeterministic*, and *probabilistic* confinement [19].

It is important to notice that nondeterministic confinement is somehow weaker than probabilistic confinement, as it is not able to capture those situations in which the probabilistic nature of an implementation may allow for the detection of the confidential information, e.g. by running the program a sufficient number of times [9]. In the context of imperative programming languages, confinement properties with respect to the value of high and low level variables, have been recently discussed in [17,20,18] where a type-system based security analysis is developed. Another recent contribution to this problem is the work in [13,14], where the use of probabilistic power-domains is proposed, which allows for a compositional specification of the non-interference property underlying a type-based security analysis.

1.2 An Example: War and Peace

We present a simple example illustrating the kind of security problem we are going to consider in this paper. Imagine a world crisis scenario: a new war might start. The Pentagon is preparing for war, but nobody knows whether it is already attacking or it is still negotiating. Clearly, this information is "sensitive" and the generals would like to confine it. They only allow a restricted number of people to have access to the one-bit information on whether the Pentagon is in a state of "war" or "peace". Therefore, the information is deterministically confined: the Pentagon's regulations allow no outsider to know about the actual situation. However, the access to low-level information is still possible for outsiders, e.g. via the press office. Nevertheless, the processing of this low-level information is different in times of "war" and "peace": In the first case all information has to be processed in atomic steps (e.g. in order to avoid system overload) while in the latter the GHQ puts no restriction on the way low level information has to be treated.

The question then is: If one can observe the behaviour of the Pentagon without directly accessing the sensitive information, e.g. by normal means of communication, is it still possible to get (indirect) information about whether it is in a state of "war" or "peace".

1.3 The Approach

The question above can be answered by first giving a formal specification of which information flow is to be considered secure, in terms of a programming language semantics.

We attack the above question by means of an approach based on a formal programming language semantics. We use for this purpose the Probabilistic Concurrent Constraint Programming (PCCP) language, which was introduced

Table 1
The Syntax of PCCP Agents.

A	::=	$\mathbf{0}$	successful termination
	$\mid$	(c)	adding a constraint
	$\mid$	$\sum_{i=1}^{n} (c_i) \to p_i : A_i$	probabilistic choice
	$\mid$	$\|_{i=1}^{n} q_i : A_i$	prioritised parallelism
	$\mid$	$\exists_x A$	hiding, local variables
	$\mid$	$p(x)$	procedure call, recursion

in [3,4] as a probabilistic version of the Concurrent Constraint Programming (CCP) paradigm [16,15]. The basic features of this language are recalled in Section 2 together with its semantics.

In Section 3 we then use this setting to give a formal specification of what we consider as a secure information flow. Having formalised the problem as a confinement property we can now apply static analysis methods in order to guarantee that a given program has a secure information flow.

We will adopt a control-flow analysis approach. In Section 4 an abstract specification of this analysis is defined by considering only those elements of the transitions occurring in an execution of the program which contain information essential for detecting the violation of the confinement property. The analysis is then proved to be safe with respect to the semantics of the language (cf. Section 5).

2

The syntax and the basic execution model of PCCP are very similar to CCP. Both languages are based on the notion of a generic *constraint system* $\mathcal{C}$, defined as a cylindric algebraic complete partial order (see [16,2] for more details), which encodes the information ordering. In PCCP probability is introduced via a probabilistic choice which replaces the nondeterministic choice of CCP, and a form of probabilistic parallelism, which replaces the pure nondeterminism in the interleaving semantics of CCP by introducing priorities. In the following we recall the syntax and the basic operational model of PCCP agents.

A PCCP program P is a set of procedure declarations of the form $p(x) :- A$, where A is an agent. The syntax of a PCCP agent is given in Table 1, where c and c_i are *finite* constraints in $\mathcal{C}$, and p_i and q_i are real numbers representing probabilities.

Table 2
The transition system for PCCP

1	$\langle \text{tell}(c), d\rangle \longrightarrow_1 \langle \mathbf{0}, c \sqcup d\rangle$	
2	$\langle \Box_{i=1}^{n} \text{ask}(c_i) \to p_i : A_i, d\rangle \longrightarrow_{\tilde{p}_j} \langle A_j, d\rangle$	$j \in [1,n]$ and $d \vdash c_j$
3	$\dfrac{\langle A_j, c\rangle \longrightarrow_p \langle A_j', c'\rangle}{\langle \|_{i=1}^{n}\ p_i : A_i, c\rangle \longrightarrow_{p\cdot\tilde{p}_j} \langle \|_{j\neq i=1}^{n}\ p_i : A_i \parallel p_j : A_j', c'\rangle}$	$j \in [1,n]$
4	$\dfrac{\langle A, d \sqcup \exists_x c\rangle \longrightarrow_p \langle A', d'\rangle}{\langle \exists_x^d A, c\rangle \longrightarrow_p \langle \exists_x^{d'} A', c \sqcup \exists_x d'\rangle}$	
5	$\langle p(y), c\rangle \longrightarrow_1 \langle A, c\rangle$	$p(x) : -A \in P$

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

Rule **1** describes the effect of tell(c). This agent always terminates successfully with probability one, and the new store is the least upper bound of the constraint c and the current store d, i.e. $c \sqcup d$. Note that the agent $\mathbf{0}$ represents successful termination and is used to distinguish success from other forms of termination, e.g. deadlock (no guard is enabled) or in general situations in which agents get *stuck*.

Both rules **2** and **3** refer to *normalised* probabilities $\tilde{p}_i$. Normalisation can be defined for a generic set of real numbers $X = \{x_i\}$, as the process of replacing each x_i by $\tilde{x}_i$ defined as follows:

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

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

The superscript in the notation $\tilde{x}_i^X$ is used to indicate the set X with respect to which we normalise.

Rule **2** describes *probabilistic choice*. An agent A_i is called *enabled* iff

its guard c_i is entailed by the store, i.e. $d \vdash c_i$. The normalisation process described above is applied only to those p_i's which are associated to enabled agents. The $\tilde{p_i}$'s are therefore normalised with respect to the set $\{p_i \mid d \vdash c_i\}$. Then the choice is made among the enabled agents only.

Rule **3** describes a *prioritised interleaving*: Each time the scheduler has to select an agent to be executed, it will choose according to the probabilities $\tilde{p}_i$, i.e. the probabilities normalised with respect to the set $\{p_i \mid A_i \text{ is active }\}$. An agent A is called *active* if it can make a transition, i.e. there exists $\langle A', d' \rangle \in \mathrm{Conf}$ such that $\langle A, d \rangle \longrightarrow_p \langle A', d' \rangle$. Again, the normalisation and the successive choice is applied to active agents only. In this way the scheduling probability of an agent actually represents its *priority* [6]. Note that there is no rule for a transition from the $\mathbf{0}$ agent to any other agent, i.e. $\mathbf{0}$ is never active.

Rule **4** in Table 2 deals with the introduction of local variables; we use the notation $\exists_x^d A$ for the agent A with local store d containing information on x which is hidden to the external store (see [15,16,2] for further details). Obviously, the transition probability p is not changed by hiding.

In the recursion rule **5** for the procedure call $p(y)$, we assume that the link between the actual parameter y and the formal parameter x has been correctly established before $p(y)$ is replaced by the body of its definition in the program P. In [16], this link is elegantly expressed by using the hiding operation $\exists_x$ and only one fresh variable. As this is a deterministic operation the transition probability in this rule is one.

Based on these transition rules we now define an operational semantics for PCCP in terms of a notion of observables which captures the results of *finite* executions of a given agent together with their probabilities. Let us first introduce some basic definitions.

Definition 2.2 Let A be a PCCP agent. A computational path ψ for A is defined by

$$\psi = \langle A_0, c_0 \rangle \longrightarrow_{p_1} \langle A_1, c_1 \rangle \longrightarrow_{p_2} \ldots \longrightarrow_{p_n} \langle A_n, c_n \rangle,$$

where $A_0 = A$, $c_0 = \mathit{true}$, $A_n = \mathbf{0}$ and $n < \infty$.

Note that this definition only account for *successful termination*. Our observables will not include infinite computation nor those situations in which the agent in the final configuration is not the $\mathbf{0}$ agent and yet is unable to make a transition, i.e. the case of *suspended computations*.

Definition 2.3 Let ψ be a computational path for A

$$\langle A_0, c_0 \rangle \longrightarrow_{p_1} \langle A_1, c_1 \rangle \longrightarrow_{p_2} \ldots \longrightarrow_{p_n} \langle A_n, c_n \rangle.$$

We define the result of ψ as $\mathit{res}(\psi) = c_n$ and its probability as $\mathit{prob}(\psi) = \prod_{i=1}^{n} p_i$.

We denote by $\mathrm{Comp}(A)$ the set of all computational paths for A.

Given a PCCP program, the set $\mathcal{R}_P$ of the results of an agent A is the

(multi-)set of all pairs $\langle c, p\rangle$, where c is the final store corresponding to the least upper bound of the partial constraints accumulated during a computational path, and p is the probability of reaching that result.

$$\mathcal{R}_P(A) = \{\langle c, p\rangle \mid \exists \psi \in \mathrm{Comp}(A) : c = res(\psi) \text{ and } p = prob(\psi)\}.$$

Because of nondeterminism, there might be different computational paths leading to the same result. Thus, we need to 'compactify' the results so as to identify all those pairs with the same constraint as a first component. This operation is formally defined as follows.

2.4 Let $S = \{\langle c_{ij}, p_{ij}\rangle\}_{i,j}$ be a (multi-)set of results, where c_{ij} denote the jth occurrence of the constraint c_i, and let $P_{c_i} = \sum_{c_i} p_{c_i}$ be the sum of all probabilities occurring in the set which are associated with c_i. The compactification of S is defined as follows:

$$\mathcal{K}(S) = \{\langle c_i, P_{c_i}\rangle \mid P_{c_i} = \textstyle\sum_{c_i} p_{c_i} = \sum_j p_{ij}\}_i.$$

We observe that this operation may not always result in a probability distribution when infinite computations are involved. In particular, this may happen when the derivation tree has infinitely many infinite branches. This case needs a more complicated, measure-theoretical treatment which we will not develop here as it is not essential for the purposes of this paper.

We can now define the observables associated to an agent A as:

$$\mathcal{O}_P(A) = \mathcal{K}(\mathcal{R}_P(A)).$$

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 store 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.

A more general notion of computational path can be defined which corresponds to a computation starting from any store c and not necessarily the empty one. Given an agent A we will denote by $\mathrm{Comp}(A, c)$ the set of all general computational paths starting from store c. Of course, $\mathrm{Comp}(A)$ correspond exactly to $\mathrm{Comp}(A, \mathit{true})$.

An operation on computational paths — which will turn out to be useful in Section 5 — is *prefixing*. This is formally defined as follows:

2.5 Let A be a PCCP agent and $\psi \in \mathrm{Comp}(A, c)$ be the general computational path

$$\langle A, c\rangle \longrightarrow_{p_1} \langle A_1, c_1\rangle \longrightarrow_{p_2} \ldots \longrightarrow_{p_n} \langle A_n, c_n\rangle.$$

Consider the transition $s = \langle B, d\rangle \longrightarrow_p \langle A, c\rangle$. We define the composition $s \cdot \psi$ as the general computational path

$$\langle B, d\rangle \longrightarrow_p \langle A, c\rangle \longrightarrow_{p_1} \langle A_1, c_1\rangle \longrightarrow_{p_2} \ldots \longrightarrow_{p_n} \langle A_n, c_n\rangle.$$

The following proposition states an important property of the transition system given in Table 2.

Proposition 2.6 *Let $\langle A, c\rangle$ be a configuration with $A \neq \mathbf{0}$ and $c \in \mathcal{C}$. Then the following condition holds:*

$$\sum_{\langle A,c\rangle \longrightarrow_{p_i} \langle A_i, c_i\rangle} p_i = 1.$$

i.e. the transitions from $\langle A, c\rangle$ are normalised.

Proof. The proof is by induction on the structure of A.

- $A = a(d)$. According to rule **1** there is only one, normalised transition.
- $A = \sum_{i=1}^{n} a(d_i) \to p_i : A_i$. The transition probabilities $\tilde{p}_j$ in rule **2** are normalised by definition.
- $A = \|_{i=1}^{n}\ p_i : A_i$. By the induction hypothesis, the transitions $\langle A_i, c\rangle \longrightarrow_{p_{k_i}} \langle A_{k_i}, c_{k_i}\rangle$ are normalised for all A_i, i.e. $\sum_{k_i} p_{k_i} = 1$. Furthermore, the $\tilde{p}_i$ are normalised by definition. Therefore, the transitions $\langle \|_{i=1}^{n}\ p_i : A_i, c\rangle \longrightarrow_{p_{k_j}\cdot\tilde{p}_j} \langle \|_{j\neq i=1}^{n}\ p_i : A_i \parallel p_j : A_{k_j}, c_{k_j}\rangle$ are normalised, as
$$\sum_i \left(\sum_{k_i} \tilde{p}_i \cdot p_{k_i}\right) = \left(\sum_i \tilde{p}_i\right) \cdot \left(\sum_{k_i} p_{k_i}\right) = 1.$$
- $A = \exists_x^d A'$. By induction hypothesis, the transitions $\langle A', d \sqcup \exists_x c\rangle \longrightarrow_p \langle A'', d'\rangle$ are normalised, therefore the transitions $\langle \exists_x^d A', c\rangle \longrightarrow_p \langle \exists_x^{d'} A'', c \sqcup \exists_x d'\rangle$ are normalised too.
- $A = p(y)$. The procedure call, according to rule **5**, is deterministic, thus normalised.

□

3 [illegible] C [illegible]

The notion of confinement is typically formulated for imperative languages in terms of variables' values. In declarative programming the role of variables is substantially different from the one they play in imperative programming. We will therefore introduce a notion of confinement which is more appropriate in our declarative setting. It refers to the identity of some agents instead of the value of some variables: A set of agents A_i confine their identity if it is impossible to specify a context in which one can determine which of the agents is actually executed.

One could think of the context given by a certain store, or, more generally, by an agent C which — whenever run in combination with each of the A_i's — is able to reveal the identity of a certain A_i. This is the case when the results obtained by the execution of C in parallel with each A_i are different, which allow us to discriminate between the A_i's. Note that the first case of a store context is obtained by restricting to agents $C = a(c)$ for some store c.

The only context which is interesting in our setting is parallel composition, as there is no interaction between sub-agents in the other structured agents, like the choice.

3.1 A set of agents A_i is *identity confined* if for all agents C the parallel executions $A_i \parallel C$ are all observationally equivalent, i.e. if all observables $\mathcal{O}(A_i \parallel C)$ are the same.

Depending on the notion of observables one may refer to, one can obtain different confinement properties. In particular, we can distinguish between *nondeterministic* identity confinement if we base our analysis on the classical nondeterministic I/O observables $\mathcal{O}_C$, and *probabilistic* identity confinement if we look at the probabilistic I/O observables $\mathcal{O}_P$.

3.1 More on the War or Peace Example

The example described in Section 1.2 can be formulated in a (P)CCP framework as follows. We introduce an agent P, P(entagon), defined as:

$$P \equiv \quad (war) \to 1 : A \,[]\, (peace) \to 1 : B$$

which, depending if it is 'war' or 'peace' executes either of the two agents A or B defined as:

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

i.e. in times of 'war' we can only one constraint at a certain time, while in 'peace' we can several in one step. The constraints 'war' and 'peace' are considered to be "hidden" or "private" to P, so they are *deterministically confined* (no other agent can simply for them).

If we now look at *nondeterministic confinement*, we see that for an agent C, representing the world C(ommunity), the whole system $P \parallel C$ has the same I/O behaviour in times of 'war' and 'peace':

$$\mathcal{O}_C(A \parallel C) = \mathcal{O}_C(B \parallel C)$$

i.e the nondeterministic observables $A \parallel C$ and $B \parallel C$ are identical.

The world community cannot decide if it is war or peace, simply because the Pentagon still produce the same possible results. The identity of $P \equiv A$ or $P \equiv B$ is protected, no C can test for the difference. The "high" level information ('peace' or 'war') is therefore confined in a nondeterministic sense.

However, if we look at the *probabilistic confinement* we see that $A \parallel C$ and $B \parallel C$ may have a slightly different behaviour (semantics). In fact, consider the agent:

$$C \equiv \quad (c) \to \frac{2}{3} : \quad (e) \,[]\, (d) \to \frac{1}{3} : \quad (f).$$

If we combine A and B with C we get the two derivation trees (according to the operational semantics given in Table 2) depicted in Figure 1 and Figure 2.

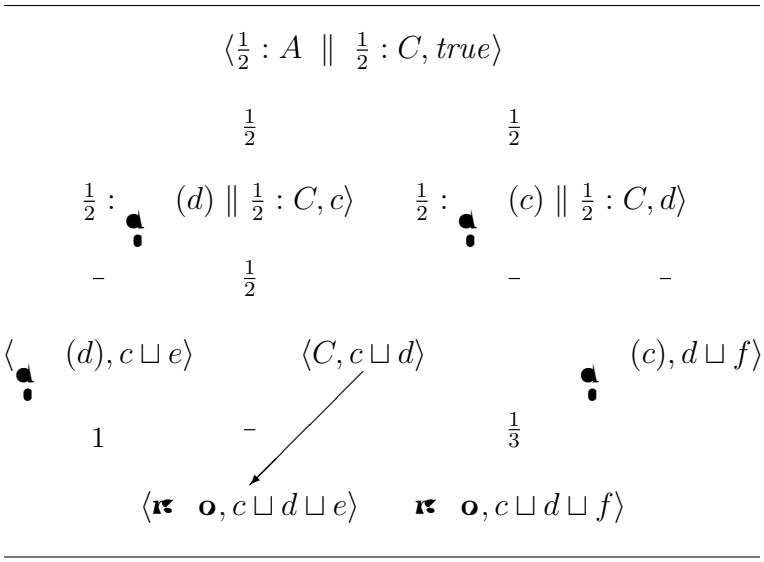


Fig. 1. The Execution of $\frac{1}{2} : A \parallel \frac{1}{2} : C$

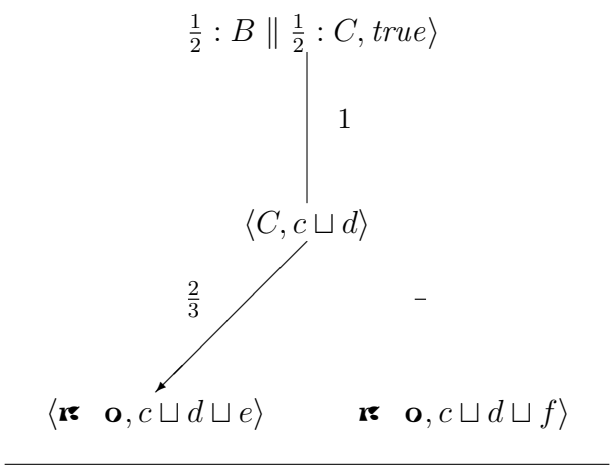


Fig. 2. The Execution of $\frac{1}{2} : B \parallel \frac{1}{2} : C$

Thus, the corresponding probabilistic observables are respectively:

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

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

By repeated questioning of, or communication with P we are therefore able to find out if it is 'war' or 'peace'. For example, if we execute 120 times $P \parallel C$ we will get about 70 times $c \sqcup d \sqcup e$ and 50 times $c \sqcup d \sqcup f$ as result iff $P \equiv A$ (the Pentagon is preparing war), but about 80 times $c \sqcup d \sqcup e$ and 40 times $c \sqcup d \sqcup f$ as outcome iff $P \equiv B$ (the Pentagon is keeping peace).

Table 3
The Analysis for PCCP Agents

$[\![\mathbf{o}]\!]$	$=$	$\emptyset$
$[\![\quad (c)]\!]$	$=$	$\{\langle true, c, 1\rangle\}$
$[\![\prod_{i=1}^{n} \quad (c_i) \to p_i : A_i]\!]$	$=$	$\bigoplus_i (p_i, c_i, [\![A_i]\!])$
$[\![\|\|_{i=i}^{n} q_i : A_i]\!]$	$=$	$\bigotimes_i (q_i, [\![A_i]\!])$
$[\![\exists_x A]\!]$	$=$	$\exists_x [\![A]\!]$
$[\![p(x)]\!]$	$=$	$[\![A]\!]\ p(x) :- A \in P$

4

Our aim is to develop a framework for analysing confinement properties of PCCP programs. Our analysis intends to construct the set of all possible transitions for the program in order to detect different behavioural structures of agents which will eventually allow us to reveal their identity. For this analysis we abstract the full semantics in as far as we ignore the concrete agents involved in each computational step, and only record possible transitions between stores (constraints) together with their probabilities.

These transitions — formally represented by triples $\langle c, d, p\rangle$ — can be seen as arcs of a graph which represents an abstracted version of the true derivation tree, as defined by the operational semantics in Table 2. An approximation of the meaning of the program can be recovered by considering the maximal acyclic paths through the graph. In general, the abstract graph will be larger than the true semantics.

4.1 Abstract Semantics

In order to analyse a PCCP program, we associate to each agent a set of triples $\langle c, d, p\rangle$. Each triple consists of two constraints and a probability. The interpretation of such a triple is that there is a transition from a node labelled with the first constraint to a node labelled by the second — the probability of such a transition is the third component. A formal compositional definition of such a semantics is given in Table 3. For the time being we will consider only finite constraint systems.

The operations in Table 3 requires some auxiliary constructions which we will introduce in the following.

4.1.1 Initial Sets

For a triple $\langle c, d, p\rangle$ as well as a set of possible transitions $\{\langle c_i, d_i, p_i\rangle\}_i$ we can easily extract information on specific aspects by projection to only one of the components, i.e.

$$\iota(\langle c, d, p\rangle) = c$$
$$\tau(\langle c, d, p\rangle) = d$$
$$\pi(\langle c, d, p\rangle) = p$$

and

$$\iota(\{\langle c_i, d_i, p_i\rangle\}) = \{c_i\}$$
$$\tau(\{\langle c_i, d_i, p_i\rangle\}) = \{d_i\}$$
$$\pi(\{\langle c_i, d_i, p_i\rangle\}) = \{p_i\}.$$

For a set of triples $\{\langle c_i, d_i, p_i\rangle\}$ we define the initial, or prefix transitions as those transitions whose first component never appears as a second one in any other transition. Formally:

$$I(\{\langle c_i, d_i, p_i\rangle\}_i) = \{\langle e, f, q\rangle \in \{\langle c_i, d_i, p_i\rangle\}_i \mid \exists\langle c_i, d_i, p_i\rangle : e = c_i \text{ and } \nexists\langle c_i, d_i, p_i\rangle : e = d_i\}$$

By projection we can extract the initial stores, probabilities, etc. simply as $\iota(I(\{\langle c_i, d_i, p_i\rangle\}))$, $\pi(I(\{\langle c_i, d_i, p_i\rangle\}))$, etc.

4.1.2 Auxiliary Operations

Given a triple $\langle c, d, p\rangle$ (representing a possible transition from store c into store d with probability p), we define its execution in the context of another constraint e as:

$$e \rhd \langle c, d, p\rangle = \langle e \sqcup c, e \sqcup d, p\rangle$$

We can extend this concept to a set $\{\langle c_i, d_i, p_i\rangle\}_i$ of possible transitions:

$$e \rhd \{\langle c_i, d_i, p_i\rangle\}_i = \{e \rhd \langle c_i, d_i, p_i\rangle\}_i = \{\langle e \sqcup c_i, e \sqcup d_i, p_i\rangle\}_i$$

Analogously, it is also possible to change the transition probabilities for a single triple:

$$q \cdot \langle c, d, p\rangle = \langle c, d, q \cdot p\rangle$$

or a whole set:

$$q \cdot \{\langle c_i, d_i, p_i\rangle\}_i = \{q \cdot \langle c_i, d_i, p_i\rangle\}_i = \{\langle c_i, d_i, q \cdot p_i\rangle\}_i$$

For a set of transitions we can also define a prefix multiplication, where only the initial probabilities are multiplied with q:

$$q \times \{\langle c_i, d_i, p_i\rangle\}_i = q \cdot I(\{\langle c_i, d_i, p_i\rangle\}_i) \cup \{\langle c_i, d_i, p_i\rangle\}_i \setminus I(\{\langle c_i, d_i, p_i\rangle\}_i)$$

And finally, we need similar operations for introducing "hidden" transitions:

$$\exists_x \langle c, d, p\rangle = \langle \exists_x c, \exists_x d, q\rangle$$

and again for a whole set:

$$\exists_x \{\langle c_i, d_i, p_i\rangle\}_i = \{\exists_x \langle c_i, d_i, p_i\rangle\}_i = \{\langle \exists_x c_i, \exists_x d_i, p_i\rangle\}_i.$$

4.1.3 Set Constructions

For the choice operation, the set of possible transitions depends on the current store. In particular, depending on how many guards are enabled we get different normalisations. Therefore, choice is modelled by:

$$\bigoplus_i (p_i, c_i, X_i) = \bigcup_{G \in \mathcal{P}(\{c_i\}_i)} \left(\bigcup_{G \vdash c_i} \tilde{p}_i^G \times (G \rhd X_i) \right)$$

i.e. the union over all sets representing possible guard combinations [4]. In each such combination, look at all enabled guards, normalise their probability (with respect to the other enabled guards in this combination), then 'execute' their corresponding agents in the context of the guard combination, where the initial transitions are multiplied with the normalised probability. By abuse of notation, we use G to denote both a set of constraints (or equivalently their least upper bound) and the set of probabilities associated to the enabled guards.

For the parallel construct we have to look at a prefix construction which stems from an implicit sequential composition:

$$\begin{aligned}\langle c, d, p\rangle \ ; \ \langle e, f, q\rangle &= \{\langle c, d, p\rangle, d \rhd \langle e, f, q\rangle\} \\ &= \{\langle c, d, p\rangle, \langle d \sqcup e, d \sqcup f, q\rangle\}.\end{aligned}$$

Then the prefix construction for a set of triples is:

$$\begin{aligned}\langle c, d, p\rangle \ ; \ \{\langle e_i, f_i, q_i\rangle\}_i &= \{\langle c, d, p\rangle, d \rhd \langle e_i, f_i, q_i\rangle\} \\ &= \{\langle c, d, p\rangle, \langle d \sqcup e_i, d \sqcup f_i, q_i\rangle\}\end{aligned}$$

The main element of the parallel construct is then a merge operation. The merge between two sets of possible transitions $X = \{\langle c_i, d_i, p_i\rangle\}_i$ and $Y = \{\langle e_j, f_j, q_j\rangle\}_j$ can be defined recursively by:

$$\begin{aligned}X \otimes Y = &\left(\bigcup_{\langle c,d,p\rangle \in I(X)} \langle c, d, p\rangle \ ; \ (X \setminus \{\langle c, d, p\rangle\} \otimes Y) \right) \\ &\cup \left(\bigcup_{\langle e,f,p\rangle \in I(Y)} \langle e, f, q\rangle \ ; \ (X \otimes Y \setminus \{\langle e, f, q\rangle\}) \right),\end{aligned}$$

where we have:

$$X \otimes \emptyset = X = \emptyset \otimes X$$

as the base case.

[4] In fact, only consistent combinations are to be considered, i.e. if a guard c entails another guard d a set G which contains c must also contain d.

It is easy to see how to generalise this binary operation to an n-ary operation:

$$\bigotimes_i X_i = \bigcup_i \left(\bigcup_{t \,\in\, I(X_i)} t \;;\; \left(\left(\bigotimes_{j \neq i} X_j \right) \otimes (X_i \setminus \{t\}) \right) \right).$$

As we consider only finite structures this recursive definition of the merge is well defined.

In order to formulate the semantics for the parallel construct we need to modify this general merge construction so as to allow for a weighted version:

$$\bigotimes_i (q_i, X_i) = \bigcup_i \left(\bigcup_{t \,\in\, I(X_i)} (\tilde{q}_i \cdot t) \;;\; \left(\left(\bigotimes_{j \neq i} (q_j, X_j) \right) \otimes (q_i, X_i \setminus \{t\}) \right) \right),$$

where we understand that the normalisation of q_i is with respect to all q_j's whose corresponding $X_j \neq \emptyset$. The pair $(q, \emptyset)$, for any q, gives us the base case for the weighted merge operation. In the case of the merge between only two elements this corresponds to the requirement:

$$(q_1, X) \otimes (q_2, \emptyset) = (1, X) = (q_1, \emptyset) \otimes (q_2, X) = X.$$

4.2 Semantical Approximation

The analysis specified in Table 3 may look at first glance nearly as an instrumented (full) semantics of PCCP. However, a closer inspection reveals immediately that this is not the case. The semantical approximation we introduce consists basically in removing information on the agents involved in a transition, and recording only the possibility of a transition between two stores together with the associated probability.

Very often it is possible to reconstruct the full semantics from this approximated semantics by considering *maximal paths*. Intuitively, a maximal path is a path which starting from the initial transition in the store *true* goes as deep in the graph as possible. Formally this notion can be defined as follows.

4.1 Given a set of tuples $S = \{\langle c_j, d_j, p_j \rangle\}_j$, a *trace* of S is any sequence $\{\langle c_i, d_i, p_i \rangle\}_{i \in [0,n]} \subseteq S$, such that $\langle c_0, d_0, p_0 \rangle \in I(S)$ and for all $1 \leq i \leq n$, $c_i = d_{i-1}$.

We now define a prefix ordering, $\leq$, on traces as follows.

4.2 Let $tr_1 = \{\langle c_i, d_i, p_i \rangle\}_{i \in [0,n_1]}$ and $tr_2 = \{\langle e_i, f_i, q_i \rangle\}_{i \in [0,n_2]}$ be two traces in S. We define the prefix order $\leq$ by:

$$tr_1 \leq tr_2 \text{ iff } 0 \leq n_1 \leq n_2 \text{ and}$$
$$\langle c_i, d_i, p_i \rangle = \langle e_i, f_i, q_i \rangle \text{ for all } i \leq n_1.$$

For a trace $tr \subseteq S$ we denote by $\downarrow(tr)$ the set $\{tr' \subseteq S \mid tr' \leq tr\}$, of all its prefixes.

4.3 A trace $tr \subseteq S$ is *maximal* iff $tr = lub_{\leq} \downarrow(tr)$.

We indicate by $\mathcal{T}(S)$ the set of all maximal traces of S.

4.4 We define the result of a maximal trace tr as

$$res(tr) = \bigsqcup \tau(tr),$$

and its probability as

$$prob(tr) = \prod_{p \in \pi(tr)} p.$$

The results of the maximal traces of A are defined as:

$$\mathcal{R}(\llbracket A \rrbracket) = \{\langle res(tr), prob(tr)\rangle \mid tr \in \mathcal{T}(\llbracket A \rrbracket)\}.$$

4.3 Computations and their Approximation

There are cases where our approximation is indeed less precise than the full semantics, that is the maximal traces are more than the actual computational paths. For example, consider the execution of the agent:

$$D \equiv \; \mathbf{ask}(true) \to \frac{1}{4} : (\frac{1}{2} : (d) \parallel \frac{1}{2} : (\mathbf{ask}(d) \to 1 : (e))) \; \square$$
$$\mathbf{ask}(true) \to \frac{3}{4} : (\frac{1}{2} : (d) \parallel \frac{1}{2} : (\mathbf{ask}(d) \to 1 : (f))).$$

The semantics of this agent D, executed in store *true*, is given by the transitions depicted in Figure 3. The set of possible transitions we obtain for this agent is:

$$\begin{aligned}\llbracket D \rrbracket = \{\; & \langle true, d, 1/4\rangle, \langle true, d, 3/4\rangle, \\ & \langle d, d \sqcup e, 1\rangle, \langle d, d \sqcup f, 1\rangle, \\ & \langle d \sqcup e, d \sqcup e, 1\rangle, \langle d \sqcup f, d \sqcup f, 1\rangle \;\}\end{aligned}$$

from which it is impossible to reconstruct the actual semantics, (cf. Figure 4). We observe that the derivation tree representing the true semantics is (isomorphic to) a sub-tree of the reconstructed tree corresponding to the approximating semantics. This obviously implies that the observables given by the true semantics is a sub-set of the observables constructed via maximal paths in the approximating semantics. This reflects the fact that, as we will show in Section 5, the approximating semantics is *correct*.

4.4 Analysing Declarative Confinement

As an example of the use of this transition semantics for analysing security notions, let us look at "probabilistic confinement".

Consider the PCCP agents from Section 3.1:

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

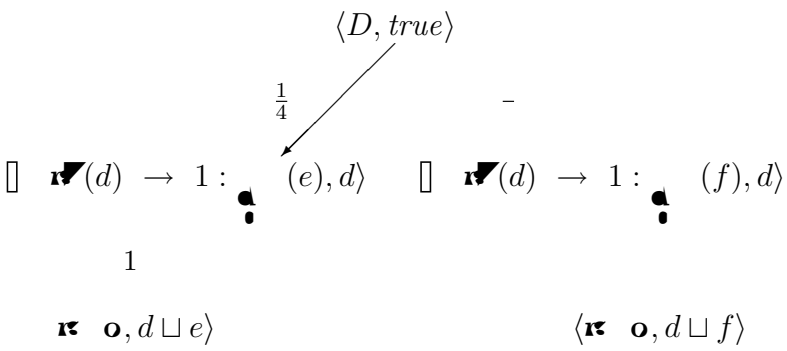


Fig. 3. The Execution of D

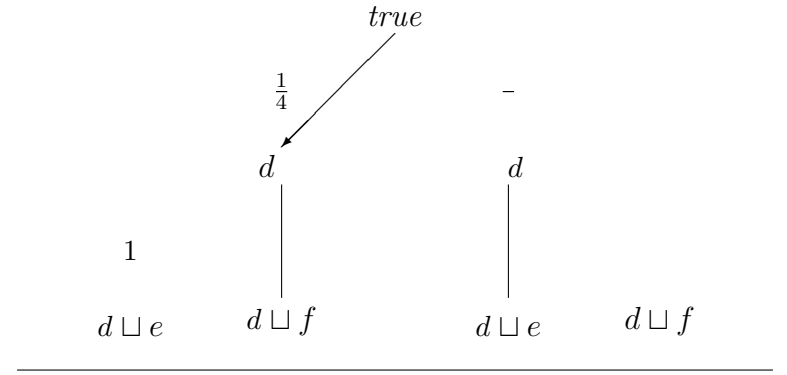


Fig. 4. The Failed Reconstruction of D

as well as:

$$C \equiv \ (c) \to \frac{2}{3} : \ (e) \ \ (d) \to \frac{1}{3} : \ (f).$$

The true semantics of the combination of respectively A and B with C results in the probabilistic input/output observables:

$$\mathcal{O}_P\left(\frac{1}{2} : A \parallel \frac{1}{2} : C\right) = \{\langle c \sqcup d \sqcup e, \frac{7}{12}\rangle, \langle c \sqcup d \sqcup f, \frac{5}{12}\rangle\}$$
$$\mathcal{O}_P\left(\frac{1}{2} : B \parallel \frac{1}{2} : C\right) = \{\langle c \sqcup d \sqcup e, \frac{2}{3}\rangle, \langle c \sqcup d \sqcup f, \frac{1}{3}\rangle\}.$$

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

$$[\![A]\!] = \{\ \langle true, c, 1/2\rangle, \langle true, d, 1/2\rangle, \langle c, c \sqcup d, 1\rangle, \langle d, c \sqcup d, 1\rangle\ \},$$

while for B we get:

$$[\![B]\!] = \{\ \langle true, c\ \sqcup\ d, 1\rangle\ \}$$

and finally for C:

$$[\![C]\!] = \{\ \langle c, c \sqcup e, 1\rangle, \langle d, d \sqcup f, 1\rangle, \\ \langle c \sqcup d, c \sqcup d \sqcup e, 2/3\rangle, \\ \langle c \sqcup d, c \sqcup d \sqcup f, 1/3\rangle\ \}.$$

By combining these semantics with the parallel rule we obtain:

$$[\![\frac{1}{2} : A \parallel \frac{1}{2} : C]\!] = \{\ \langle \mathit{true}, c, 1/2\rangle, \langle \mathit{true}, d, 1/2\rangle, \\ \langle c, c \sqcup d, 1/2\rangle, \langle d, c \sqcup d, 1/2\rangle, \\ \langle c, c \sqcup e, 1/2\rangle, \langle d, d \sqcup f, 1/2\rangle, \\ \langle c \sqcup d, c \sqcup d \sqcup e, 2/3\rangle, \langle c \sqcup d, c \sqcup d \sqcup f, 1/3\rangle, \\ \langle c \sqcup e, c \sqcup d \sqcup e, 1\rangle, \langle d \sqcup f, c \sqcup d \sqcup f, 1\rangle\ \}$$

and

$$[\![\frac{1}{2} : B \parallel \frac{1}{2} : C]\!] = \{\ \langle \mathit{true}, c \sqcup d, 1\rangle, \\ \langle c \sqcup d, c \sqcup d \sqcup e, 2/3\rangle, \\ \langle c \sqcup d, c \sqcup d \sqcup f, 1/3\rangle\ \}.$$

The maximal paths through these graphs give the same results with the same probabilities as the true semantics. As we found at least one context, namely C, in which the two agents behave differently — as it is also revealed by the analysis — we can assert that A and B are *not* identity confined.

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

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

5 C [illegible] A [illegible]

In order to show that the abstract semantics introduced in Section 4 provides a safe base for a correct analysis of PCCP programs, we study the relation between the concrete semantics given in terms of the transition system in Table 2 and the semantics $[\![\cdot]\!]$ defined in Table 3. In the correctness argument we will focus on the choice and parallel constructs and abstract for the time being from hiding and recursion.

As already mentioned, the semantics $[\![P]\!]$ of a program P consists in collecting all the possible transitions that P can make starting from store *true*. Each transition is described by a tuple which record initial and final stores and the probability of the transition, while the intermediate agents are abstracted away. Intuitively, this makes the semantics "bigger" than the concrete one: Some maximal traces can be re-constructed from the set of tuples, which do not correspond to any of the computational paths in the concrete computational tree for P.

[illegible] **5.1** We define the maximal trace associated to a computational

path

$$\psi = \langle A_0, c_0\rangle \longrightarrow_{p_0} \langle A_1, c_1\rangle \longrightarrow_{p_1} \ldots \longrightarrow_{p_n} \langle A_n, c_n\rangle,$$

as $\overline{\psi} = \{\langle c_0, c_1, p_1\rangle, \ldots, \langle c_{n-1}, c_n, p_n\rangle\}$.

5.2 Given a set of computational paths C, we define the set of maximal traces associated to C as

$$\overline{C} = \{tr \mid \exists \psi \in C \text{ with } tr = \overline{\psi}\}.$$

We now show that given an agent A, any computational path for A corresponds to a maximal trace of $[\![A]\!]$.

5.3 *For all agents A, if $\psi \in \mathrm{Comp}(A)$ then there exists a maximal trace $tr \in \mathcal{T}([\![A]\!])$ such that $tr = \overline{\psi}$, i.e.*

$$\mathcal{T}(\overline{\mathrm{Comp}(A)}) \subseteq \mathcal{T}([\![A]\!]).$$

. By induction on the structure of A.

$A =$: $\mathrm{Comp}(A) = \emptyset = [\![A]\!]$.

$A =$ (c): $\mathrm{Comp}(A) = \{\psi\}$, where ψ is the computational path

$$\langle (c), true\rangle \longrightarrow_1 \langle , c\rangle \not\longrightarrow .$$

On the other hand, $[\![(c)]\!] = \mathcal{T}([\![(c)]\!]) = \{\langle true, c, 1\rangle\}$.

It is immediate to see that $\{\langle true, c, 1\rangle\} = \overline{\psi}$.

$A = \Box_{i=1}^{n} (c_i) \to p_i : A_i$: Every computation for A is of either of the two forms:

( ) $\psi_\emptyset = \langle A, true\rangle \not\longrightarrow$.

In this case we have $\mathrm{Comp}(A) = \emptyset \subseteq \mathcal{T}([\![A]\!])$.

( ) $\psi_j = \langle A, true\rangle \longrightarrow_{\tilde{p_j}} \langle A_j, true\rangle \cdot \psi'_j$, where "$\cdot$" is the composition operator defined in Definition 2.5.

This case occurs when at least one agent A_j is enabled. Then the computational paths for A are exactly the computational paths for each enabled agent A_j.

By inductive hypothesis, for all j there exists $tr'_j \in \mathcal{T}([\![A_j]\!])$ such that $tr'_j = \overline{\psi'_j}$.

Then consider $tr_j = \tilde{p_j} \times tr'_j$. Since $\tilde{p_j}$ is a normalised probability with respect to the set G of enabled guards and $tr'_j \in [\![A_j]\!]$, we have that $tr_j \in \bigcup_{G \vdash c_i} \tilde{p}_i^G \times (G \rhd [\![A_i]\!])$. Therefore, $tr_j \in [\![A]\!]$. Moreover, by construction and the inductive hypothesis $tr_j = \overline{\psi_j}$ holds.

$A = \|_{i=1}^{n}\ p_i : A_i$: For the sake of simplicity, consider the case $A = p_0 : A_0 \parallel p_1 : A_1$. We show that for any computational path $\psi \in \mathrm{Comp}(A)$,

$$\overline{\psi} \in \mathcal{T}((p_0, \overline{\psi_0}) \otimes (p_1, \overline{\psi_1}))$$

holds, where $\psi_0 \in \mathrm{Comp}(A_0)$ and $\psi_1 \in \mathrm{Comp}(A_1)$.

Since $\mathcal{T}((p_0, \overline{\psi_0}) \otimes (p_1, \overline{\psi_1})) \subseteq \mathcal{T}([\![A]\!])$, this will show the assertion of Proposition 5.3.

The proof is by induction on the length m of ψ. Note that if n_0 and n_1 are the lengths of ψ_0 and ψ_1 respectively, then $m \leq n_0$ and $m \leq n_1$.

$m = 0$ This is the case in which both A_0 and A_1 are not active. Then $\mathrm{Comp}(A) = \emptyset \subseteq \mathcal{T}((p_0, \overline{\psi_0}) \otimes (p_1, \overline{\psi_1}))$.

$m \geq 1$ Suppose that for $i = 0, 1$

$$\psi_i = \langle A_i, true\rangle \longrightarrow_{q_i} \langle A_i', c_i\rangle \cdot \psi_i',$$

where $\psi_i' \in \mathrm{Comp}(A_i', c_i)$, and $\langle A_i, true\rangle \longrightarrow_{q_i} \langle A_i', c_i\rangle$ is the first transition step of ψ_i (with possibly $A' = \mathbf{0}$).

Then any computational path for A is of the form

$$\psi(i) = \langle A, true\rangle \longrightarrow_{\tilde{p}_i \cdot q_i} \langle A_i', c_i\rangle \cdot \psi',$$

where $\psi' \in \mathrm{Comp}(p_j : A_j \parallel p_i : A_i', c_i)$ and $j = [i+1]_2$ (i.e. $i+1$ the sum modulo 2).

Without loss of generality we can assume $i = 1$.

Since the length of ψ' is $m - 1$, we have by inductive hypothesis that

$$\overline{\psi'} \in \mathcal{T}((p_0, \overline{\psi_0}) \otimes (p_1, \overline{\psi_1'})). \qquad (*)$$

Then define $\overline{\psi} = \{\langle true, c_1, \tilde{p}_1 \cdot q_1\rangle\} \cup \overline{\psi'}$.

By definition of the merge operator $\otimes$,

$$(p_0, \overline{\psi_0}) \otimes (p_1, \overline{\psi_1}) = \bigcup_{t \,\in\, I(\overline{\psi_0})} (\tilde{p}_0 \cdot t) \;;\; \left((p_1, \overline{\psi_1}) \otimes (p_0, \overline{\psi_0} \setminus \{t\})\right) \cup \bigcup_{t \,\in\, I(\overline{\psi_1})} (\tilde{p}_1 \cdot t) \;;\; \left((p_0, \overline{\psi_0}) \otimes (p_1, \overline{\psi_1} \setminus \{t\})\right).$$

Now observe that

$$\overline{\psi} = \tilde{p}_1 \cdot I(\overline{\psi_1}) \;;\; \overline{\psi'},$$

and that

$$\overline{\psi_1'} = \overline{\psi_1} \setminus I(\overline{\psi_1}).$$

Then conclude by the inductive hypothesis $(*)$ that $\overline{\psi} \in \mathcal{T}((p_0, \overline{\psi_0}) \otimes (p_1, \overline{\psi_1}))$.

□

In order to relate the analysis of PCCP agents to the problem of their identity confinement we introduce the notion of *re-constructibility*.

5.4 An agent A is re-constructible iff

$$\overline{\mathrm{Comp}(A)} = \mathcal{T}([\![A]\!]).$$

5.5 *An agent A is re-constructible iff*

$$\mathcal{R}_P(A) = \mathcal{R}([\![A]\!])$$

. Obvious. □

5.6 *Let $\{A_i\}_{i=1}^n$ be a set of re-constructible agents. If*

$$[\![A_i]\!] = [\![A_j]\!]$$

for all $i, j \in \{1, \ldots, n\}$ then

$$\mathcal{O}_P(A_i) = \mathcal{O}_P(A_j)$$

for all $i, j \in \{1, \ldots, n\}$.

. **p** . From $[\![A_i]\!] = [\![A_j]\!]$ it follows that $\mathcal{T}([\![A_i]\!]) = \mathcal{T}([\![A_j]\!])$.

Since A_i and A_j are re-constructible by Lemma 5.5 we can conclude that $\mathcal{R}_P(A_i) = \mathcal{R}_P(A_j)$ and thus $\mathcal{O}_P(A_i) = \mathcal{O}_P(A_j)$. □

A useful criterion to decide if an agent A is re-constructible which is based only on its analysis $[\![A_i]\!]$ is stated the following.

. **p o r** **5.7** *An agent A is re-constructible if for all $d \in \iota([\![A]\!])$ the following condition holds:*

$$\sum_{\langle d, c_i, p_i \rangle \in [\![A]\!]} p_i = 1.$$

. **p** . Suppose by contradiction that A is not re-constructible. Then we have that $\overline{\mathrm{Comp}(A)} \subset \mathcal{T}([\![A]\!])$, i.e. there exists at least one trace tr in $\mathcal{T}([\![A]\!])$ which does not correspond to any computational path of A. Therefore, there exists at least one triple $\langle d, d_0, p_0 \rangle \in tr$ which does not belong to any trace in $\overline{\mathrm{Comp}(A)}$.

Now two things can happen:

- There exists a configuration $\langle A', d \rangle$ in a computational path in $\mathrm{Comp}(A)$. As all transitions and triples which correspond to actual computational paths are already normalised we have
$$\sum_{\langle d, c_i, p_i \rangle \in [\![A]\!]} p_i \geq \sum_{\langle d, c_i, p_i \rangle \in \overline{\mathrm{Comp}(A)}} p_i + p_0 = 1 + p_0 > 1.$$
- There is no configuration $\langle A', d \rangle$ in any computational path in $\mathrm{Comp}(A)$. The initial part of tr might correspond to the initial part of a computational path. However since tr does not correspond to a computational path, there must be a configuration $\langle A'', e \rangle$ from where it started to differ. The computational transitions from this configuration are already normalised. Then consider the triple$\langle e, e_0, q_0 \rangle \in tr$, which corresponds to the first difference of tr from the computational path. We have that:
$$\sum_{\langle e, e_i, q_i \rangle \in [\![A]\!]} q_i \geq \sum_{\langle e, e_i, q_i \rangle \in \overline{\mathrm{Comp}(A)}} q_i + q_0 = 1 + q_0 > 1.$$

□

6 C [illegible] r [illegible] p [illegible] p

We introduced the notion of identity confinement which characterises noninterference and allows us to distinguish between nondeterministic and probabilistic confinement in a declarative setting. The important point we have stressed is that agents which appear to be nondeterministically indistinguishable — because they have the same *possible* observables — may well violate a probabilistic notion of confinement — as one can distinguish them by analysing

the *probabilities* corresponding to possible outcomes. This result is closely related to the work on confinement in [19,17,13,14] where the setting is the one of imperative languages and a type-system based approach to the analysis.

It is important to notice that the notion of *probabilistic confinement* as discussed in this paper is — even when formulated in terms of probabilistic observables — in itself still a "classical" notion: A set of agents is probabilistically confined if *for all* contexts they have *exactly the same* probabilistic observables. Our aim for future work is to investigate "truly probabilistic" notions of confinement. One such notion can be obtained by weakening the notion of identity confinement so as to require that observables are *similar* rather than *identical* in all contexts, according to an appropriate notion of similarity. For example, in the probabilistic semantics of PCCP, where observables are normalised vectors (i.e. distributions), we can express the concept of similarity in two closely related ways:

- via the *norm* difference, i.e. by looking at
$$\|\mathcal{O}(A \parallel C, d) - \mathcal{O}(B \parallel C, d)\|,$$
- via the *inner product*, i.e considering
$$\langle \mathcal{O}(A \parallel C, d), \mathcal{O}(B \parallel C, d) \rangle .$$

Clearly both of these *similarity measures* describe for normalised vectors about the same situation: If the norm of the difference of two vectors is small then so is the angle between them, that is their inner product.

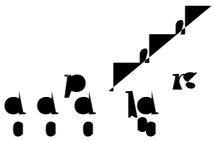
The relevance of such a notion comes from the fact that the information on the similarity of two agents can be exploited to estimate, e.g. by using statistical methods, the number of test runs which are needed to distinguish them. Typically, the more *similar* (in either of ways mentioned above) the observables of two agents are, the more difficult it becomes to distinguish them by an experiment. In the War and Peace example discussed in this paper, we argued that about 120 repetitions would allow us to distinguish $A \parallel C$ from $B \parallel C$, as we would get a distribution on the observables of about 70 to 50 as opposed to 80 to 40. Since the difference between the observables is about $\frac{1}{12}$, an experimental result obtained by only, let's say, 12 test runs would distinguish $A \parallel C$ and $B \parallel C$ with a low probability.

Other important issues remain to be investigated. Among them, there is the issue of extending our analysis which allows for detecting whether a confinement property is violated, so as to be able to eventually construct an agent — a "spy" — which can discriminate among several agents and thus reveal their identity. In particular, we are interested in a simple, single step test agent able to achieve this aim.

A possible way to attack this problem we are currently considering is to look at the set of possible traces and their probability. The test agent could be executed at every step (chosen by the scheduler) resulting eventually in different outcomes depending on the current intermediate store. The analysis of the average outcome of the test agent over all traces utilising the information

on their probability should give us the possibility to observe the different behaviour of the "spy" agent. Thus, if two agents have different traces and/or different probability distributions on the set of traces, it should be possible to determine their identity.

Finally, further work will be devoted to develop our analysis so as to be able to deal with recursion. A possible direction is the use of an abstract interpretation approach and, in particular, of the probabilistic abstract interpretation framework introduced in [5,7].



[1] Center for High Assurance Computing Systems, *The Navy Handbook for the computer security certification of trusted systems* (1998), draft version on http://chacs.nrl.navy.mil/main.html.

[2] de Boer, F. S., A. Di Pierro and C. Palamidessi, *Nondeterminism and Infinite Computations in Constraint Programming*, Theoretical Computer Science **151** (1995), pp. 37–78.

[3] Di Pierro, A. and H. 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* (1998), pp. 174–183.

[4] Di Pierro, A. and H. 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*, Lecture Notes in Computer Science **1450** (1998), pp. 446–455.

[5] Di Pierro, A. and H. Wiklicky, *Concurrent Constraint Programming: Towards Probabilistic Abstract Interpretation*, in: M. Gabbrielli and F. Pfenning, editors, *Proceedings of PPDP'00 – Principles and Practice of Declarative Programming*, ACM SIGPLAN (2000), pp. 127–138.

[6] Di Pierro, A. and H. Wiklicky, *Quantitative observables and averages in Probabilistic Concurrent Constraint Programming*, in: K. Apt, T. Kakas, E. Monfroy and F. Rossi, editors, *New Trends in Constraints — Selected Papers of the ERCIM/Compulog Workshop on Constraints, October 1999, Paphos, Cyprus*, number 1865 in Lecture Notes in Computer Science (2000), pp. 212–236.

[7] Di Pierro, A. and H. Wiklicky, *Measuring the precision of abstract interpretations*, in: *Proceedings of LOPSTR'00 – 10th International Workshop on Logic-Based Program Synthesis and Transformation, London, UK*, Lecture Notes in Computer Science (2001), pp. 1–18.

[8] Goguen, J. and J. Meseguer, *Security Policies and Security Models*, in: *Proceedings of the IEEE Symposium on Security and Privacy* (1982), pp. 11–20.

[9] Gray, III, J. W., *Probabilistic interference*, in: *Proceedings of the IEEE Symposium on Security and Privacy* (1990), pp. 170–179.

[10] Gupta, V., R. Jagadeesan and V. A. Saraswat, *Probabilistic concurrent constraint programming*, in: *Proceedings of CONCUR 97: Concurrency Theory*, number 1576 in Lecture Notes in Computer Science (1997), pp. 243–257.

[11] Lampson, B. W., *A Note on the Confinement Problem*, Communications of the ACM **16** (1973), pp. 613–615.

[12] Nielson, F., H. R. Nielson and C. Hankin, "Principles of Program Aanalysis," Springer Verlag, Berlin – Heidelberg, 1999.

[13] Sabelfeld, A. and D. Sands, *A per model of secure information flow in sequential programs*, in: *ESOP'99*, number 1576 in Lecture Notes in Computer Science (1999), pp. 40–58.

[14] Sabelfeld, A. and D. Sands, *Probabilistic noninterference for multi-threaded programs*, in: *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, 2000, pp. 200–214.

[15] Saraswat, V. A. and M. Rinard, *Concurrent constraint programming*, in: *Symposium on Principles of Programming Languages (POPL'90)* (1990), pp. 232–245.

[16] Saraswat, V. A., M. Rinard and P. Panangaden, *Semantics foundations of concurrent constraint programming*, in: *Symposium on Principles of Programming Languages (POPL'91)* (1991), pp. 333–353.

[17] Smith, G. and D. Volpano, *Secure information flow in a multi-threaded imperative language*, in: *Symposium on Principles of Programming Languages (POPL'98)* (1998), pp. 355–364.

[18] Smith, G. and D. Volpano, *Verifying secrets and relative secrecy*, in: *Symposium on Principles of Programming Languages (POPL'00)* (2000), pp. 368–276.

[19] Volpano, D. and G. Smith, *Confinement properties for programming languages*, SIGACT News **29** (1998), pp. 33–42.

[20] Volpano, D. and G. Smith, *Probabilistic noninterference in a concurrent language*, in: *Proceedings of the 11th IEEE Computer Security Foundations Workshop (CSFW '98)* (1998), pp. 34–43.