Analysing Approximate Confinement under Uniform Attacks

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

Abstract. We are concerned to give certain guarantees about the security of a system. We identify two kinds of attack: the internally scheduled attack (exemplified by Trojan Horse attacks) and externally scheduled attacks (exemplified by timing attacks). In this paper we focus on the latter. We present a semantic framework for studying such attacks in the context of PCCP, a simple process algebra with a constraint store. We show that a measure of the efficacy of an attacker can be determined by considering its observable behaviour over the "average" store of the system (for some number of steps). We show how to construct an analysis to determine the average store using the technique of probabilistic abstract interpretation.

1 Introduction

Confidentiality is that aspect of computer security concerned with how information is allowed to flow through a computer system. Information flows are treated in a binary fashion: they are either allowed to flow or not. Models for confidentiality typically characterise the absence of information flow between objects (across interfaces or along channels) by essentially reducing non-interference to confinement. The first attempt to formalise such an absence of information flow is the notion of non-interference proposed by Goguen and Meseguer in their seminal paper [12]. In this paper we approach the problem of confidentiality by looking at models which are able to give a *quantitative* estimate of the information flowing through a system. Such models abandon the purely qualitative binary view of the information flow by characterising how much information is actually "leaking" from the system rather than the complete absence of any flow. This allows us to define notions of non-interference which are *approximate* and yet able to capture the security properties of a system in a more "realistic" way: in real systems high-level input interferes with low-level output all the time [16].

Our approach is therefore not to aim for a methodology to develop "secure" systems — e.g. a type system which guarantees the confidentiality of information [20, 22, 21] — rather our approach is to develop a framework for understanding and analysing systems as they are and not as they should be.

* Chris Hankin is partly funded by the EU FET open project SecSafe.

The security of a system may be measured by its robustness to certain kinds of attack. We differentiate two situations: internally scheduled spies in which the spy is considered part of the system (Trojan Horse attacks are of this kind) – we studied this kind of spy in [8]; externally scheduled spies in which the spy is only allowed to observe the system at fixed points and for a fixed time – this is the subject of this paper. The latter kind of attack is suitable for modelling *timing attacks* [13]; with this kind of attack a snooper can determine a private key by keeping track of *how long* a computer takes to decipher messages.

This paper introduces a framework for studying attacks involving externally scheduled spies. We use a simple probabilistic process calculus, PCCP, to illustrate the main concepts; PCCP is introduced in the next section. We formally define the notion of approximate confinement for both internally scheduled and externally scheduled spies in Section 3. We then present an analysis of approximate confinement for external attack and illustrate the technique with some examples in Section 4.

2 The Language

We illustrate our approach by referring to a probabilistic declarative language, namely Probabilistic Concurrent Constraint Programming (PCCP), which was introduced in [9] as a probabilistic version of the Concurrent Constraint Programming (CCP) paradigm [19]. This language can be seen as a kind of "process algebra" enhanced with a notion of "computational state", referred to as "store". These states are ordered by an entailment relation $\vdash$ (also denoted by $\sqsupseteq$) — they form a cpo — and all computations lead to sequences of stores which are *monotone* with respect to $\vdash$.

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 [19, 5] 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.

The syntax of a PCCP agent is given by the following grammar, where c and c_i are *finite* constraints in $\mathcal{C}$, and p_i and q_i are real numbers representing probabilities:

$$A ::= \mathbf{stop} \;\Big|\; \mathbf{tell}(c) \;\Big|\; \talloblong_{i=1}^{n} \mathbf{ask}(c_i) \rightarrow p_i : A_i \;\Big|\; \|_{i=1}^{n}\, q_i : A_i \;\Big|\; \exists_x A \;\Big|\; p(x).$$

2.1 Operational semantics

The operational semantics of PCCP is defined 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 and the transition relation $\longrightarrow_p$ is defined in Table 1. The state of the system is described by the agent

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

Table 1. The Transition System for PCCP

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 [5] for a detailed description. The rules for probabilistic choice and prioritised parallelism involve a normalisation process needed to re-distribute the probabilities among those agents A_i which can actually be chosen for execution. Such agents must be enabled (i.e. the corresponding guards $\mathbf{ask}(c_i)$ succeed) or active (i.e. able to make a transition). The probability after normalisation is denoted by $\tilde{p}_j$.

Definition 1. Let A be a PCCP agent. A computational path π for A in store d is defined by $\pi \equiv \langle A_0, c_0\rangle \longrightarrow_{p_1} \langle A_1, c_1\rangle \longrightarrow_{p_2} \dots \longrightarrow_{p_n} \langle A_n, c_n\rangle$, where $A_0 = A$, $c_0 = d$, $A_n = \mathbf{stop}$ and $n < \infty$.

Note that this definition only accounts for *successful termination*. The notion of observables we consider does not include infinite computation nor those situations in which the agent in the final configuration is not the **stop** agent and yet is unable to make a transition, i.e. the case of *suspended computations*. We denote by $\mathrm{Comp}(A, d)$ the set of all computational paths for A in store d.

Definition 2. Let $\pi \in \mathrm{Comp}(A, d)$ be a computational path for A in store d $\pi \equiv \langle A, d\rangle = \langle A_0, c_0\rangle \longrightarrow_{p_1} \langle A_1, c_1\rangle \longrightarrow_{p_2} \dots \longrightarrow_{p_n} \langle A_n, c_n\rangle$.We define the result of π as $res(\pi) = c_n$ and its probability as $prob(\pi) = \prod_{i=1}^{n} p_i$.

Given a PCCP agent A, the set $\mathcal{R}$ of the results of A is the family 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:

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

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. To this purpose we introduce the following equivalence relation on $\mathrm{Comp}(A, d)$.

Definition 3. Let $\pi, \pi' \in \mathrm{Comp}(A, d)$ be two computational paths for A in store d. We define $\pi \equiv \pi'$ iff $res(\pi) = res(\pi')$. The equivalence class of π is denoted by $[\pi]$.

The definitions of $res(\pi)$ and $prob(\pi)$ are extended to $\mathrm{Comp}(A)/_{\equiv}$ in the obvious way by $res([\pi]) = res(\pi)$ and $prob([\pi]) = \sum_{\pi' \in [\pi]} prob(\pi')$. Thus, the compactification of the results of a given agent A in store d corresponds to the set $\mathcal{K}(\mathcal{R}(A, d)) = \{\langle res([\pi]), prob([\pi])\rangle \mid [\pi] \in \mathrm{Comp}(A)/_{\equiv}\}$.

We can now define the observables of an agent A with respect to store d as:

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

The observables $\mathcal{O}(A, d)$ are actually a probability distribution on the constraint system $\mathcal{C}$ and can be seen as a vector in the free vector space associated to $\mathcal{C}$, namely:

$$\mathcal{V}(\mathcal{C}) = \left\{ \sum x_c c \mid x_c \in \mathbb{R},\ c \in \mathcal{C} \right\}.$$

Each linear combination $\sum x_c c$ of constraints represents a vector distribution and its coordinates x_c correspond to the probabilities associated to each constraint.

The notion $\mathcal{O}(A, d)$ can be extended so as to consider the results of executing the agent A starting from a given distribution over constraints instead of a single constraint. The resulting notion corresponds to the vectorial sum of the observables of A in each constraint weighted by the probability associated to that store. By a slight abuse of notation we will denote this extended notion by $\mathcal{O}(A, \boldsymbol{d})$, where $\boldsymbol{d}$ is a store distribution. Formally, this is defined as:

$$\mathcal{O}(A, \boldsymbol{d}) = \sum \{p \cdot \mathcal{O}(A, c) \mid p = \boldsymbol{d}(c) \neq 0\}.$$

3 Confinement

We consider in the following the notion of *identity confinement* [6]. This notion establishes whether it is possible to identify which process is running in a given program. We will look at different types of relatively simple attacks aiming to reveal the identity of some unknown process. Our aim is to characterise the "vulnerability" of a set of agents by applying program analysis techniques. In particular, we will use the Probabilistic Abstract Interpretation technique [10, 11], which we will briefly recall in Section 4.1, to define a suitable collecting semantics for our analysis.

Example 4. In an imperative language, confinement — as formulated for example in [18] — usually refers to a standard (two-level) security model consisting of high and low level variables. One then considers the (value of the) high variable h as confined if the value of the low level variable l is not "influenced" by the value of the high variable, i.e. if the observed values of l are independent of the values of h.

The following statement illustrates the difference between non-deterministic and probabilistic confinement [17, 18]:

$$h := h \bmod 2; (l := h \ {}_{\frac{1}{2}}[]_{\frac{1}{2}} \ (l := 0 \ {}_{\frac{1}{2}}[]_{\frac{1}{2}} \ l := 1))$$

The value of l clearly depends "somehow" on h. However, if we resolve the choice non-deterministically it is impossible to say anything about the value of h by observing the possible values of l. Concretely, we get the following dependencies between h and possible values of l: For $h \bmod 2 = 0$ we have $\{l = 0, l = 1\}$ and for $h \bmod 2 = 1$ we get $\{l = 1, l = 0\}$, i.e. the possible values of l are the same independently from the fact that h is even or odd. In other words, h is non-deterministically confined.

In a probabilistic setting the observed values for l and their probabilities allow us to distinguish cases where h is even from those where h is odd. We have the following situation: For $h \bmod 2 = 0$ we get $\{\langle \frac{3}{4}, l = 0\rangle, \langle \frac{1}{4}, l = 1\rangle\}$, and for $h \bmod 2 = 1$ we have $\{\langle \frac{1}{4}, l = 0\rangle, \langle \frac{3}{4}, l = 1\rangle\}$. Therefore, the probabilities to get $l = 0$ and $l = 1$ reveal if h is even or odd, i.e. h is probabilistically *not* confined.

Example 5. A similar situation to Example 4 can be described in our declarative setting by the following agents:

$$\begin{aligned}
\texttt{highOn} &\equiv \mathbf{ask}(\mathit{true}) \to \frac{1}{2} : \mathbf{tell}(\texttt{on}) \,[]\, \mathbf{ask}(\mathit{true}) \to \frac{1}{2} : \texttt{Random} \\
\texttt{highOff} &\equiv \mathbf{ask}(\mathit{true}) \to \frac{1}{2} : \mathbf{tell}(\texttt{off}) \,[]\, \mathbf{ask}(\mathit{true}) \to \frac{1}{2} : \texttt{Random} \\
\texttt{Random} &\equiv \mathbf{ask}(\mathit{true}) \to \frac{1}{2} : \mathbf{tell}(\texttt{on}) \,[]\, \mathbf{ask}(\mathit{true}) \to \frac{1}{2} : \mathbf{tell}(\texttt{off})
\end{aligned}$$

The constraint system underlying these agents has four elements

$$\mathcal{C} = \{\mathit{true}, \texttt{on}, \texttt{off}, \mathit{false} = \texttt{on} \sqcup \texttt{off}\},$$

with the ordering $\mathit{true} \sqsubseteq \texttt{on} \sqsubseteq \mathit{false}$ and $\mathit{true} \sqsubseteq \texttt{off} \sqsubseteq \mathit{false}$. The constraints `on` and `off` represent the situations in which the low variable $l = 1$ or $l = 0$ respectively. The agent `highOn` corresponds then to the behaviour of the imperative program fragment in case that $h \bmod 2 = 1$ while `highOff` corresponds to the case where $h \bmod 2 = 0$. The auxiliary agent `Random` corresponds to the second choice in the above imperative fragment. The imperative notion of confinement now translates in our framework into a problem of identity confinement: Getting information about h in the previous setting is equivalent to discriminating between `highOn` and `highOff`, i.e. revealing their identity.

3.1 Approximate Confinement

The definition of the notion of *approximate confinement* we give is parametric with respect to a set of admissible spies $\mathcal{S}$ and can be instantiated for different types of attacks. We say that two agents A and B are *approximately confined* with respect to a set of spies $\mathcal{S}$ if there exists an $\varepsilon \geq 0$ such that for all $S \in \mathcal{S}$ the *distance* between the observables of agent A attacked by S and the observables of agent B attacked by S is smaller than ε. The precise definition of what "attacked by S" means obviously depends on the nature of the attack and will be made clear later on. In any case, as we refer to probabilistic observables (i.e. vector distributions) we will use as a measure for this distance between observables the supremum norm $\|\cdot\|_\infty$ formally defined as $\|(x_i)_{i\in I}\| = \|(x_i)_{i\in I}\|_\infty = \sup_{i\in I} |x_i|$, where $(x_i)_{i\in I}$ represents a probability distribution. This norm allows us to identify a single constraint c for which the associated probabilities are maximally different and is therefore particularly appropriate for our purposes.

Admissible Spies We consider only *passive* spies, i.e. spies which are sensitive to the computational states or environments created by the agents they attack but which are unable to change the execution of the attacked agent. Moreover, our spies are *memoryless* spies, i.e. spies whose execution depends only on the single state or environment they are executed in.

Concretely we therefore consider spies of type $\mathcal{S}_n$ which are PCCP agents of the following form:

$$\mathcal{S}_n = \left\{ \square_{i=1}^{n} \mathbf{ask}(c_i) \rightarrow q_i : \mathbf{tell}(f_i) \right\},$$

where $f_i \in \mathcal{C}$ are *fresh* constraints, that is constraints which never appear in the execution of the host agents, and $c_i \in \mathcal{C}$ are some guards.

These spies may interact in various ways with the agents in question. We will consider two different models based on an *internal* and an *external* scheduling respectively.

Internal Scheduling: Trojan Horses The spy $S \in \mathcal{S}_n$ can be considered to be part of the system, i.e. it is treated like a normal PCCP agent which is executed in parallel with the attacked agent A. The behaviour we thus have to analyse is that of the agent: $q : S \parallel p : A$ with $p + q = 1$. We can then define formally the approximate confinement property for internal attacks as follows:

Definition 6. Given a set of admissible spies $\mathcal{S}$, $\varepsilon \geq 0$, and fixed scheduling priorities p and $q = 1 - p$, we call two agents A and B ε-confined iff:

$$\sup_{S\in\mathcal{S}} \|\mathcal{O}(p : A \parallel q : S) - \mathcal{O}(p : B \parallel q : S)\| = \varepsilon.$$

This definition can be generalised to a set of more than two agents. A semantics-based analysis of this property has been presented in [8, 7]. It is easy to verify that if two agents A and B are ε-confined with $\varepsilon = 0$ then they are probabilistically confined in the sense of [22, 18, 6].

The advantage of this type of attacks is that we can simply use the standard semantics of PCCP to analyse the confinement properties of A. The disadvantage is that the likelihood of S being executed is decreasing exponentially even when the store created by A does not change. This leads to a kind of "logarithmic" nature of the analysis.

Moreover, internally scheduled spies cannot be used to model attacks which are "external" to the system, such as *timing attacks* [13] and *power consumption attacks* [14]. These attacks are based on regular measurements made on the system to find out the time required to perform a given task (e.g. the modular exponentiation in the RSA crypto-system) or to find out some more physical information like the switching energy dissipated by the system when transiting from one state to the other. The system may be vulnerable against a statistical analysis of the information acquired by such measurements, (e.g. attackers may be able to break the RSA and many other crypto-systems).

External Scheduling or Uniform Attack A suitable model for the above mentioned external attacks is a spy which "observes" the agent A from the "outside" for a finite number of execution steps and is able to interfere uniformly during the given execution time interval. As this implies the definition of a scheduler essentially different from the one implicit in the semantics of PCCP, we can not simulate this behaviour using the parallel construct in PCCP. Therefore, we introduce the notation: $S \rhd_n A$, where we fix a maximal observation time n. During the first n steps of the execution of A there is each time a constant probability $\frac{1}{n}$ that S might be executed in the store currently provided by A. Thus, we call such an attack a "uniform" attack, in contrast to the "geometric" nature of the above described internal attack. The notion of approximate confinement can now be defined as follows:

Definition 7. Given a set of admissible spies $\mathcal{S}$, we call two agents A and B ε-confined for some $\varepsilon \geq 0$ and a fixed observation time n iff:

$$\sup_{S \in \mathcal{S}} \|\mathcal{O}(S \rhd_n A) - \mathcal{O}(S \rhd_n B)\| = \varepsilon.$$

Operational Semantics of Uniform Attacks The operational meaning of $S \rhd_n A$ can be formalised in terms of the SOS transition system for PCCP given in Table 1 provided we extend it with a dummy rule **R0**: $\langle \mathbf{stop}, c \rangle \longrightarrow_1 \langle \mathbf{stop}, c \rangle$.

This allows us to deal with the situation in which the observed agent terminates in a number of steps $s < n$ by extending the computation with a cycle on the last configuration. The following rule expresses the uniform attack of a spy S to an agent A:

$$\frac{\begin{array}{c}\langle S, c_t \rangle \longrightarrow_q^* \langle \mathbf{stop}, c_t \sqcup d \rangle \\ \langle A, c \rangle \longrightarrow_{p_1} \langle A_1, c_1 \rangle \longrightarrow_{p_2} \ldots \longrightarrow_{p_n} \langle A_n, c_n \rangle\end{array}}{\langle S \rhd_n A, c \rangle \longrightarrow_{\frac{1}{n} \cdot q \cdot \prod_{i=1}^n p_i} \langle A_n, c_n \sqcup d \rangle} \quad \text{for } 1 \leq t \leq n.$$

The idea is the following: Suppose that A starting in c can make n steps $\langle A, c\rangle \longrightarrow_{p_1} \langle A_1, c_1\rangle \longrightarrow_{p_2} \ldots \longrightarrow_{p_n} \langle A_n, c_n\rangle$.On the other hand, assume that S executed in some intermediate store c_t with $1 \leq t \leq n$ produces store $c_t \sqcup d$ with (accumulated) probability [1] q. The idea is then to insert the execution of $\langle S, c_t\rangle$ before the execution of $\langle A_{t+1}, c_{t+1}\rangle$. In other words the path:

$$\langle A, c\rangle \longrightarrow_{p_1} \ldots \langle A_t, c_t\rangle \longrightarrow_{p_t} \langle A_{t+1}, c_{t+1}\rangle \ldots \longrightarrow_{p_n} \langle A_n, c_n\rangle$$

becomes the path:

$$\langle S \rhd_n A, c\rangle \longrightarrow_{p_1} \ldots \langle S \rhd A_t, c_t\rangle \longrightarrow_q^* \langle A_t, c_t \sqcup d\rangle \longrightarrow_{p_t} \ldots \longrightarrow_{p_n} \langle A_n, c_n \sqcup d\rangle .$$

The probability accumulated along this path is $q \cdot \prod_{i=1}^n p_i$. Since the "intrusion" of S can happen at any of the n transitions, each of the paths has a probability $\frac{1}{n}$. Combining the probabilities and by a slight abuse of notation, we can then represent such a computation as:

$$\langle S \rhd_n A, c\rangle \longrightarrow_{\frac{1}{n} \cdot q \cdot \prod_{i=1}^n p_i} \langle A_n, c_n \sqcup d\rangle .$$

4 Analysing Approximate Confinement for Uniform Attacks

In this section we will present an analysis of the approximate confinement property with respect to external attackers. The analysis is based on an abstract semantics which associates to an agent its *average store* in a n-steps computation. This is the average of the intermediate stores computed by the agent in a computation truncated after n steps. We will show that the operational meaning of $S \rhd_n A$ is precisely characterised by the observables of S in the average store of A. Thus, the abstract semantics will allow us to calculate the ϵ quantifying the confinement of a program. The abstract semantics is defined via a collecting semantics constructed as a probabilistic abstract interpretation of the SOS semantics for PCCP in Section 2.1. We first briefly recall the basic definitions of the Probabilistic Abstract Interpretation framework in the next section.

4.1 Probabilistic Abstract Interpretation

Probabilistic Abstract Interpretation (PAI) [10, 11] recasts the well-known Abstract Interpretation technique due to Cousot and Cousot [4, 1, 15]) in terms of a probabilistic semantics which replaces the standard order-theoretic one. Thus, linear spaces replace cpo's as semantical domains, so that abstraction and concretisation functions are now linear maps between the concrete and abstract domains. Moreover, the standard way to relate the abstraction and the concretisation functions via the notion of Galois connection is replaced by the use of the Moore-Penrose pseudo-inverse of a linear operator [3].

[1] There will be in general several such computations by S with all the associated probabilities summing up to 1.

We assume that the vector spaces representing the semantical domains are vector spaces with *inner product*, as is the case for finite dimensional spaces and Hilbert spaces. Under this assumption, we can define the analog of a classical Galois connection (α, γ) in a vector space setting as follows.

As in the classical case, the abstraction and concretisation functions preserve the structure of the underlying domains; this corresponds to the abstraction $\boldsymbol{\alpha}$ and the concretisation $\boldsymbol{\gamma}$ being linear maps.

Furthermore, the classical condition for Galois connections that $\alpha \circ \gamma(d) \sqsubseteq d$ can be seen in a linear setting as the condition that $\boldsymbol{\alpha} \circ \boldsymbol{\gamma}$ is the orthogonal projection in the image of $\boldsymbol{\alpha}$, that is $\boldsymbol{\alpha} \circ \boldsymbol{\gamma} = \pi_\alpha$. In the same way $\gamma \circ \alpha(c) \geq c$ turns into $\boldsymbol{\gamma} \circ \boldsymbol{\alpha} = \pi_\gamma$.

These two conditions define the so-called *Moore-Penrose pseudo-inverse* [3]. The Moore-Penrose pseudo-inverse $\mathbf{T}^\dagger$ of a linear operator $\mathbf{T}$ can be defined for real and complex finite-dimensional vector spaces as well as for infinite-dimensional Hilbert spaces [2]; it is unique and always exists.

A probabilistic abstract interpretation is now simply defined by a pair $(\boldsymbol{\alpha}, \boldsymbol{\gamma})$ of linear maps between vector spaces representing the concrete and abstract domain such that $\boldsymbol{\gamma}$ is the Moore-Penrose pseudo-inverse of $\boldsymbol{\alpha}$, i.e. $\boldsymbol{\gamma} = \boldsymbol{\alpha}^\dagger$, and vice versa.

4.2 Concrete Semantics

In Table 2 we define a relation $\longrightarrow$ which dynamically generates all the computational paths of a given program P step by step. This relation can be seen as a functional description of the transition relation $\longrightarrow_p$ in Table 1: it transforms a triple $\langle C_1, C_2, q\rangle \in\longrightarrow_p$ into a triple $\langle C_2, C_3, q'\rangle \in\longrightarrow_p$ iff $C_1 \longrightarrow_q C_2 \longrightarrow_{q''} C_3$, with $q' = q \cdot q''$, is a sub-tree of the computational tree for P.

The premises and the conclusions of each rule can be seen as distributions over sets of pairs of configurations. Therefore, Table 2 actually defines a linear operator on the vector space $\mathcal{V}(\text{Conf} \times \text{Conf})$,

$$\mathbf{F} : \mathcal{V}(\text{Conf} \times \text{Conf}) \mapsto \mathcal{V}(\text{Conf} \times \text{Conf}).$$

We will use $\mathbf{F}$ as a generator of sequences of vectors in $\mathcal{V}(\text{Conf}\times\text{Conf})$. For an agent A, a store c and a fixed length n, these sequences are constructed by iteratively applying $\mathbf{F}$ starting from an initial transition $\langle\langle \text{Init}, \mathit{true}\rangle, \langle \text{A}, c\rangle\rangle$. We represent this initial transition as a vector $\boldsymbol{t}_0$ in $\mathcal{V}(\text{Conf}\times\text{Conf})$. Thus, the concrete n-steps semantics of A in c is given by the sequence $(\boldsymbol{t}_0, \mathbf{F}(\boldsymbol{t}_0), \mathbf{F}^2(\boldsymbol{t}_0), \ldots, \mathbf{F}^n(\boldsymbol{t}_0))$ and represents the n-steps prefixes of any computational paths for A in store c.

Note that the domain of the operator $\mathbf{F}$ (i.e. the free vector space $\mathcal{V}(\text{Conf} \times \text{Conf})$) may be infinite-dimensional. However, as we consider only *finite* observations, the set of reachable configurations for each agent is always finite. Therefore, in constructing the semantics of an agent, we can effectively consider the restriction of $\mathbf{F}$ to finite-dimensional sub-spaces of $\mathcal{V}(\text{Conf} \times \text{Conf})$. This allows us to directly apply the PAI framework recalled in Section 4.1, as shown in the next section.

$$\mathbf{R1}\ \{\ldots,\langle\langle A',c'\rangle,\langle\mathbf{tell}(c),d\rangle,p'\rangle,\ldots\}$$
$$\longrightarrow\{\ldots,\langle\langle\mathbf{tell}(c),d\rangle,\langle\mathbf{stop},c\sqcup d\rangle,1\cdot p'\rangle,\ldots\}$$

$$\mathbf{R2}\ \{\ldots,\langle\langle A',c'\rangle,\langle\|_{i=1}^{n}\ \mathbf{ask}(c_i)\rightarrow p_i:A_i,d\rangle,p'\rangle,\ldots\}$$
$$\longrightarrow\{\ldots,\langle\langle\|_{i=1}^{n}\ \mathbf{ask}(c_i)\rightarrow p_i:A_i,d\rangle\,\langle A_j,d\rangle,\tilde{p}_j\cdot p'\rangle,\ldots\}$$
$$j\in[1,n]\text{ and }d\vdash c_j$$

$$\mathbf{R3}\ \{\ldots,\langle\langle A',c'\rangle,\langle\|_{i=1}^{n}\ p_i:A_i,d\rangle,p'\rangle,\ldots\}$$
$$\longrightarrow\{\ldots,\langle\langle\|_{i=1}^{n}\ p_i:A_i,d\rangle,\langle A_j',d'\rangle,\tilde{p}_j\cdot p'\rangle,\ldots\}$$
$$j\in[1,n]\text{ and }\langle A_j,d\rangle\longrightarrow_{p_j}\langle A_j',d'\rangle$$

$$\mathbf{R4}\ \{\ldots,\langle\langle A',c'\rangle,\langle\exists_x A,d\rangle,p'\rangle,\ldots\}$$
$$\longrightarrow\{\ldots,\langle\langle\exists_x A,d\rangle,\langle\exists_x A',d\sqcup\exists_x d'\rangle,p\cdot p'\rangle,\ldots\}$$
$$\langle A,\exists_x d\rangle\longrightarrow_p\langle A',d'\rangle$$

$$\mathbf{R5}\ \{\ldots,\langle\langle A',c'\rangle,\langle p(y),d\rangle,p'\rangle,\ldots\}$$
$$\longrightarrow\{\ldots,\langle\langle p(y),d\rangle,\langle A,d\rangle,1\cdot p'\rangle,\ldots\}$$
$$p(x):-A\in P$$

Table 2. A Concrete Collecting Semantics for PCCP

Example 8. For the agents `highOn` and `highOff` in Example 5 the only transitions involved are enumerated as follows:

	`highOn`	`highOff`
1	$\langle\langle\texttt{Init},\mathit{true}\rangle,\langle\texttt{highOn},\mathit{true}\rangle\rangle$	$\langle\langle\texttt{Init},\mathit{true}\rangle,\langle\texttt{highOff},\mathit{true}\rangle\rangle$
2	$\langle\langle\texttt{highOn},\mathit{true}\rangle,\langle\mathbf{stop},\texttt{on}\rangle\rangle$	$\langle\langle\texttt{highOff},\mathit{true}\rangle,\langle\mathbf{stop},\texttt{off}\rangle\rangle$
3	$\langle\langle\texttt{highOn},\mathit{true}\rangle,\langle\texttt{Random},\mathit{true}\rangle\rangle$	$\langle\langle\texttt{highOff},\mathit{true}\rangle,\langle\texttt{Random},\mathit{true}\rangle\rangle$
4	$\langle\langle\texttt{Random},\mathit{true}\rangle,\langle\mathbf{stop},\texttt{on}\rangle\rangle$	$\langle\langle\texttt{Random},\mathit{true}\rangle,\langle\mathbf{stop},\texttt{on}\rangle\rangle$
5	$\langle\langle\texttt{Random},\mathit{true}\rangle,\langle\mathbf{stop},\texttt{off}\rangle\rangle$	$\langle\langle\texttt{Random},\mathit{true}\rangle,\langle\mathbf{stop},\texttt{off}\rangle\rangle$
6	$\langle\langle\mathbf{stop},\texttt{on}\rangle,\langle\mathbf{stop},\texttt{on}\rangle\rangle$	$\langle\langle\mathbf{stop},\texttt{on}\rangle,\langle\mathbf{stop},\texttt{on}\rangle\rangle$
7	$\langle\langle\mathbf{stop},\texttt{off}\rangle,\langle\mathbf{stop},\texttt{off}\rangle\rangle$	$\langle\langle\mathbf{stop},\texttt{off}\rangle,\langle\mathbf{stop},\texttt{off}\rangle\rangle$

Therefore, the concrete semantics for these agents can be constructed by using the restrictions of the operator $\mathbf{F}$ represented by the following matrices:

$$\mathbf{F}_{\texttt{highOn}}=\begin{pmatrix}0&\frac{1}{2}&\frac{1}{2}&0&0&0&0\\0&0&0&0&0&1&0\\0&0&0&\frac{1}{2}&\frac{1}{2}&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{pmatrix}\quad\mathbf{F}_{\texttt{highOff}}=\begin{pmatrix}0&\frac{1}{2}&\frac{1}{2}&0&0&0&0\\0&0&0&0&0&0&1\\0&0&0&\frac{1}{2}&\frac{1}{2}&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{pmatrix}$$

4.3 Abstract Semantics

In order to define our analysis we will consider a more abstract semantics which we will construct as a probabilistic abstract interpretation of the operator $\mathbf{F}$ introduced in Section 4.2.

We define such a collecting semantics for PCCP via the abstract linear operator $\mathbf{G}$ induced by the probabilistic abstraction $\boldsymbol{\alpha} : \mathcal{V}(\mathrm{Conf} \times \mathrm{Conf}) \mapsto \mathcal{V}(\mathrm{Conf})$ defined as:

$$\boldsymbol{\alpha}(\{\ldots, \langle\langle A_1, c_1\rangle, \langle A_2, c_2\rangle, p\rangle, \ldots\}) = (\{\ldots, \langle\langle A_2, c_2\rangle, p\rangle, \ldots\}).$$

This abstraction only records the last configuration of a transition by transforming a distribution on transitions into a distribution on single configurations. Intuitively, this means that we only record the probability of reaching a given configuration, while the information on the configurations from which it can be reached is lost.

The operator $\mathbf{G} : \mathcal{V}(\mathrm{Conf}) \mapsto \mathcal{V}(\mathrm{Conf})$ is defined as

$$\mathbf{G} = \boldsymbol{\alpha} \cdot \mathbf{F} \cdot \boldsymbol{\gamma},$$

where $\boldsymbol{\gamma}$ is the Moore-Penrose pseudo-inverse of the abstraction $\boldsymbol{\alpha}$ above.

Proposition 9. *The operator* $\mathbf{G} : \mathcal{V}(\mathrm{Conf}) \mapsto \mathcal{V}(\mathrm{Conf})$ *defined as* $\boldsymbol{\alpha} \cdot \mathbf{F} \cdot \boldsymbol{\gamma}$ *transforms distributions over configurations into distributions over configurations according to the rules in Table 3.*

Note that because of nondeterminism the rules in Table 3 actually transform distributions over multi-sets of configurations. Instead the linear operator $\mathbf{G}$ operates on the free vector space over Conf and therefore on distributions over sets of configurations. Therefore the correspondence in Proposition 9 holds up to an appropriate reduction of distributions on multi-sets to distributions on sets. One such reduction would transform a distribution of the form $\{\langle a, 1/2\rangle, \langle b, 1/4\rangle, \langle a, 1/4\rangle\}$ into the distribution $\{\langle a, 3/4\rangle, \langle b, 1/4\rangle\}$.

Analogously to the concrete semantics, the collecting semantics of an agent is given for a fixed length n, by the sequence $(\mathbf{G}^i(\boldsymbol{\alpha t}_0))_{i=0}^n$.

The semantics $\mathbf{G}$ allows us to construct for a program P an abstract representation of its computational tree given by the sequence of the sets of nodes at each level of the tree. The length n of the sequence represents the maximal depth of the tree we need to construct for our observation. Clearly this abstract representation does not allow us to re-construct exactly all the computational paths of P.

Example 10. Consider again Example 8. To obtain the abstract semantics for the agents `highOn` and `highOff`, we first enumerate all those configurations in Conf which are reachable by `highOn` and `highOff` respectively:

$$\langle \mathtt{highOn}, \mathit{true}\rangle, \langle \mathtt{Random}, \mathit{true}\rangle, \langle \mathbf{stop}, \mathtt{on}\rangle, \langle \mathbf{stop}, \mathtt{off}\rangle$$
$$\langle \mathtt{highOff}, \mathit{true}\rangle, \langle \mathtt{Random}, \mathit{true}\rangle, \langle \mathbf{stop}, \mathtt{on}\rangle, \langle \mathbf{stop}, \mathtt{off}\rangle.$$

$$\mathbf{R1}\ \{\ldots, \langle\langle \mathbf{tell}(c), d\rangle, p\rangle, \ldots\} \longrightarrow \{\ldots, \langle\langle \mathbf{stop}, c \sqcup d\rangle, p\rangle, \ldots\}$$

$$\mathbf{R2}\ \{\ldots, \langle\langle \Box_{i=1}^{n} \mathbf{ask}(c_i) \rightarrow p_i : A_i, d\rangle, q\rangle, \ldots\} \longrightarrow \{\ldots, \langle\langle A_j, d\rangle, q \cdot \tilde{p}_j\rangle, \ldots\} \quad j \in [1, n] \text{ and } d \vdash c_j$$

$$\mathbf{R3}\ \{\ldots, \langle\langle \|_{i=1}^{n}\ p_i : A_i, d\rangle, q\rangle, \ldots\} \longrightarrow \{\ldots, \langle\langle A'_j, d'\rangle, q \cdot \tilde{p}_j\rangle, \ldots\} \quad j \in [1, n] \text{ and } \langle A_j, d\rangle \longrightarrow_{p_j} \langle A'_j, d'\rangle$$

$$\mathbf{R4}\ \{\ldots, \langle\langle \exists_x A, d\rangle, q\rangle, \ldots\} \longrightarrow \{\ldots, \langle\langle \exists_x A', d \sqcup \exists_x d'\rangle, q \cdot p\rangle, \ldots\} \quad \langle A, \exists_x d\rangle \longrightarrow_p \langle A', d'\rangle$$

$$\mathbf{R5}\ \{\ldots, \langle\langle p(y), d\rangle, q\rangle, \ldots\} \longrightarrow \{\ldots, \langle\langle A, d\rangle, q\rangle, \ldots\} \quad p(x) : -A \in P$$

Table 3. An Abstract Collecting Semantics for PCCP

Next we calculate the abstractions $\boldsymbol{\alpha}_{\texttt{highOn}}$ and $\boldsymbol{\alpha}_{\texttt{highOff}}$, which map transitions into their final configurations — e.g. $\langle\langle \texttt{highOn}, \mathit{true}\rangle, \langle \texttt{Random}, \mathit{true}\rangle\rangle$ is mapped into $\langle \texttt{Random}, \mathit{true}\rangle$ — and their corresponding Moore-Penrose Pseudo-Inverses $\gamma_{\texttt{highOn}}$ and $\gamma_{\texttt{highOff}}$. These maps are represented by the matrices in Table 4. With this we can construct the induced semantics $\mathbf{G}_{\texttt{highOn}} = \boldsymbol{\alpha}_{\texttt{highOn}}\mathbf{F}_{\texttt{highOn}}\gamma_{\texttt{highOn}}$ and $\mathbf{G}_{\texttt{highOff}} = \boldsymbol{\alpha}_{\texttt{highOff}}\mathbf{F}_{\texttt{highOff}}\gamma_{\texttt{highOff}}$:

$$\mathbf{G}_{\texttt{highOn}} \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \mathbf{G}_{\texttt{highOff}} = \begin{pmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

4.4 Analysis: The Average Store

In order to find out whether two agents A and B are approximately confined with respect to an external attack we have to calculate the observables of $S \rhd_s A$ and $S \rhd_s B$ and evaluate their distance. The semantics for $S \rhd_s A$ as defined in Section 3.1 can be equivalently described by means of the collecting semantics introduced in the previous section. In particular, by a further abstraction we can construct the average store out of $\mathbf{G}(A)$ and $\mathbf{G}(B)$ as follows. We define the linear operator $\beta : \mathcal{V}(\text{Conf}) \mapsto \mathcal{V}(\mathcal{C})$,which transforms distributions over configurations into distributions over stores. Such an operator simply extracts the constraint from a configuration. We then calculate the *average store* of A (B) in n steps by summing up the distributions representing the stores reached by A (B) after t steps, for all $t \in [1, n]$ weighted by $\frac{1}{n}$. Formally, the average store semantics of an agent A is parametric in n and is defined as:

$$[\![A]\!]_n = \frac{1}{n}\sum_{i=1}^{n} \beta(\mathbf{G}^i \{\langle A, \mathit{true}\rangle\}).$$

$$\boldsymbol{\alpha}_{\texttt{highOn}} = \begin{pmatrix} 1&0&0&0\\ 0&0&1&0\\ 0&1&0&0\\ 0&0&1&0\\ 0&0&0&1\\ 0&0&1&0\\ 0&0&0&1 \end{pmatrix} \quad \boldsymbol{\gamma}_{\texttt{highOn}} = \begin{pmatrix} 1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0\\ 0&\frac{1}{3}&0&\frac{1}{3}&0&\frac{1}{3}&0\\ 0&0&0&0&\frac{1}{2}&0&\frac{1}{2} \end{pmatrix}$$

$$\boldsymbol{\beta} = \begin{pmatrix} 1&0&0&0\\ 1&0&0&0\\ 0&1&0&0\\ 0&0&1&0 \end{pmatrix}$$

$$\boldsymbol{\alpha}_{\texttt{highOff}} = \begin{pmatrix} 1&0&0&0\\ 0&0&0&1\\ 0&1&0&0\\ 0&0&1&0\\ 0&0&0&1\\ 0&0&1&0\\ 0&0&0&1 \end{pmatrix} \quad \boldsymbol{\gamma}_{\texttt{highOff}} = \begin{pmatrix} 1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0\\ 0&0&0&\frac{1}{2}&0&\frac{1}{2}&0\\ 0&\frac{1}{3}&0&0&\frac{1}{3}&0&\frac{1}{3} \end{pmatrix}$$

Table 4. Matrix Representation of Operators.

Proposition 11. *For an agent A and a spy S we have:*

$$\mathcal{O}(S \rhd_n A) = \mathcal{O}(S, [\![A]\!]_n).$$

That is: The observables of an n steps attack on A by S can be obtained by simply executing S in the (n-step) average store of A. Therefore, the average store of an agent determines their ε-confinement.

Example 12. The abstraction $\boldsymbol{\beta}$ for the agents `highOn` and `highOff` is represented by the matrix in Table 4.

Using the generator $\mathbf{G}$ from Example 10 for the abstract semantics we can now (re)construct the average stores, using the following operators:

$$\frac{1}{n}\left(\sum_{i=0}^{n-1} \boldsymbol{\beta} \cdot \mathbf{G}^i_{\texttt{highOn}}\right) \text{ and } \frac{1}{n}\left(\sum_{i=0}^{n-1} \boldsymbol{\beta} \cdot \mathbf{G}^i_{\texttt{highOff}}\right).$$

By applying these two operators to the initial configurations: $\boldsymbol{t}_{\texttt{highOn}} = \{\langle\langle \texttt{highOn}, \mathit{true}\rangle, 1\rangle\} = (1,0,0,0,0)$ and $\boldsymbol{t}_{\texttt{highOff}} = \{\langle\langle \texttt{highOff}, \mathit{true}\rangle, 1\rangle\} = (0,1,0,0,0)$, we obtain the average stores

$$[\![\texttt{highOn}]\!]_3 = \left\{\left\langle \mathit{true}, \frac{1}{2}\right\rangle, \left\langle \texttt{on}, \frac{5}{12}\right\rangle, \left\langle \texttt{off}, \frac{1}{12}\right\rangle\right\}$$
$$[\![\texttt{highOff}]\!]_3 = \left\{\left\langle \mathit{true}, \frac{1}{2}\right\rangle, \left\langle \texttt{on}, \frac{1}{12}\right\rangle, \left\langle \texttt{off}, \frac{5}{12}\right\rangle\right\}$$

This shows that `highOn` and `highOff` can be distinguished by an external spy observing both agents for three steps. One such external spy is given by:

$$S \equiv \textbf{ask}(\texttt{on}) \to \frac{1}{3} : \textbf{tell}(e) \,[\!]\, \textbf{ask}(\texttt{off}) \to \frac{2}{3} : \textbf{tell}(f).$$

For a three steps external attack of S on `highOn` and `highOff` we get the following observables:

$$\mathcal{O}(S \rhd_3 \texttt{highOn}) = \left\{ \left\langle true, \frac{1}{2} \right\rangle, \left\langle \texttt{on} \sqcup e, \frac{5}{12} \right\rangle, \left\langle \texttt{off} \sqcup f, \frac{1}{12} \right\rangle \right\}$$
$$\mathcal{O}(S \rhd_3 \texttt{highOff}) = \left\{ \left\langle true, \frac{1}{2} \right\rangle, \left\langle \texttt{on} \sqcup e, \frac{1}{12} \right\rangle, \left\langle \texttt{off} \sqcup f, \frac{5}{12} \right\rangle \right\}$$

i.e. with respect to three steps external attacks by this particular spy S the two agents `highOn` and `highOff` are $\frac{1}{3}$-confined.

Example 13. In this example we show how timing attacks [13] can be modelled as external attacks. For any number $m \in \mathbb{N}$, consider the following PCCP agents:

$$\begin{aligned} \texttt{Countdown}(m) :- \; & \mathbf{ask}(m = 0) \to 1 : \mathbf{tell}(\texttt{done}) \\ [] \; & \mathbf{ask}(m \neq 0) \to 1 : \texttt{Countdown}(m-1). \end{aligned}$$

As long as such an agent $\texttt{Countdown}(m)$ is just "counting down", i.e. $m \neq 0$, the store will be unchanged. Only when the count-down is finished a marker `done` is placed into the store. For all $m \in \mathbb{N}$ the agent $\texttt{Countdown}(m)$ therefore has the same input/output observables:

$$\mathcal{O}(\texttt{Countdown}(m), true) = \{\langle \texttt{done}, 1 \rangle\}.$$

Consider two agents $\texttt{Countdown}(m_1)$ and $\texttt{Countdown}(m_2)$, with $m_1 < m_2$. As long as both agents are counting down, no spy will observe any difference, i.e. an attack $S \rhd_n \texttt{Countdown}(m)$ is pointless for $n \leq m_1$. After the first agent terminates the difference between the two agent may be observed by a spy like:

$$S \equiv \mathbf{ask}(true) \to \frac{1}{2} : \mathbf{tell}(a) \; [] \; \mathbf{ask}(\texttt{done}) \to \frac{1}{2} : \mathbf{tell}(b),$$

where a and b are fresh constraints, once it is scheduled for a time slot $n > m_1$. After also the second agent terminates, the spy will again see no difference between the two agents.

The most effective spy thus has to observe two agents $\texttt{Countdown}(m_1)$ and $\texttt{Countdown}(m_2)$ for exactly m_2 steps. In this case the average stores

$$\begin{aligned} [\![\texttt{Countdown}(m_1)]\!]_{m_2} &= \{\langle true, \frac{m_1}{m_2}\rangle, \langle \texttt{done}, \frac{m_2 - m_1}{m_2}\rangle\} \\ [\![\texttt{Countdown}(m_2)]\!]_{m_2} &= \{\langle true, 1\rangle\} \end{aligned}$$

are maximally different. To calculate this difference we first construct the observables for S spying for m_2 steps on $\texttt{Countdown}(m_1)$ and $\texttt{Countdown}(m_2)$:

$$\begin{aligned} \mathcal{O}(S \rhd_{m_1} \texttt{Countdown}(m_1)) &= \mathcal{O}(S, [\![\texttt{Countdown}(m_1)]\!]_{m_2}) \\ &= \mathcal{O}(S, \{\langle true, \frac{m_1}{m_2}\rangle, \langle \texttt{done}, \frac{m_2 - m_1}{m_2}\rangle\}) \end{aligned}$$

$$
\begin{aligned}
&= \{\langle a, \frac{m_1}{m_2} + \frac{1}{2}(\frac{m_2 - m_1}{m_2})\rangle, \langle b, \frac{1}{2}(\frac{m_2 - m_1}{m_2})\rangle\} \\
\mathcal{O}(S \rhd_{m_2} \texttt{Countdown}(m_2)) &= \mathcal{O}(S, [\![\texttt{Countdown}(m_1)]\!]_{m_2}) \\
&= \mathcal{O}(S, \{\langle \mathit{true}, 1\rangle\}) \\
&= \{\langle a, 1\rangle\}.
\end{aligned}
$$

Thus we have:

$$
\|\mathcal{O}(S \rhd_{m_1} \texttt{Countdown}(m_1)) - \mathcal{O}(S \rhd_{m_2} \texttt{Countdown}(m_2))\| = \frac{m_2 - m_1}{2m_2}.
$$

5 Conclusions

We have defined a notion of approximate confinement and shown how it can be used to analyse the robustness of systems in the presence of externally scheduled spies. This analysis is based on a state-based collecting semantics constructed as a probabilistic abstract interpretation of the concrete SOS semantics. For large choices of n or for the analysis of languages that do not share the nice monotonicity properties of PCCP, it would be necessary to construct a static analysis. This might be constructed as probabilistic abstract interpretation of the state-based collecting semantics or in a more ad hoc way — this remains a topic for future investigation.

References

1. S. Abramsky and C. Hankin, editors. *Abstract Interpretation of Declarative Languages.* Ellis-Horwood, Chichester, England, 1987.
2. F.J. Beutler. The operator theory of the pseudo-inverse. *Journal of Mathematical Analysis and Applications*, 10:451–470,471–493, 1965.
3. S.L. Campbell and D. Meyer. *Generalized Inverse of Linear Transformations.* Constable and Company, London, 1979.
4. P. Cousot and R. Cousot. Abstract Interpretation and Applications to Logic Programs. *Journal of Logic Programming*, 13(2-3):103–180, July 1992.
5. F.S. de Boer, A. Di Pierro, and C. Palamidessi. Nondeterminism and Infinite Computations in Constraint Programming. *Theoretical Computer Science*, 151(1):37–78, 1995.
6. A. Di Pierro, C. Hankin, and H. Wiklicky. Probabilistic confinement in a declarative framework. In *Declarative Programming – Selected Papers from AGP 2000 – La Havana, Cuba*, volume 48 of *Electronic Notes in Theoretical Computer Science*, pages 1–23. Elsevier, 2001.
7. A. Di Pierro, C. Hankin, and H. Wiklicky. Approximate non-interference. In Iliano Cervesato, editor, *CSFW'02 — 15th IEEE Computer Security Foundation Workshop*, pages 3–17, Cape Breton, Canada, 24–26 June 2002. IEEE Computer Society Press.
8. A. Di Pierro, C. Hankin, and H. Wiklicky. On approximate non-interference. In P. Syverson and J. Guttman, editors, *Proceedings of WITS'02 – Workshop on Issues in the Theory of Security, 14–15 January, Portland*, January 2002. `http://www.dsi.unive.it/IFIPWG1_7/WITS2002`.

9. A. Di Pierro 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*, pages 174–183. IEEE Computer Society Press, 1998.
10. A. Di Pierro 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*, pages 127–138, Montréal, Canada, September 2000. ACM SIGPLAN, Association of Computing Machinery.
11. A. Di Pierro 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*, volume 2042 of *Lecture Notes in Computer Science*, pages 147–164, Berlin – New York, 2001. Springer Verlag.
12. J. Goguen and J. Meseguer. Security Policies and Security Models. In *IEEE Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society Press, 1982.
13. P.C. Kocher. Cryptanalysis of Diffie-Hellman, RSA, DSS, and other crypto-systems using timing attacks. In D. Coppersmith, editor, *Advances in Cryptology, CRYPTO '95: 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27–31, 1995*, volume 963 of *Lecture Notes in Computer Science*, pages 171–183, Berlin — Heidelberg — London, 1995. Springer-Verlag.
14. P.C. Kocher, J.M. Jaffe, and B Jun. Differential power analysis. In *Proc. 19th International Advances in Cryptology Conference – CRYPTO '99*, pages 388–397, 1999.
15. F. Nielson, H. Riis Nielson, and C. Hankin. *Principles of Program Analysis.* Springer Verlag, Berlin – Heidelberg, 1999.
16. P.Y.A. Ryan, J. McLean, J. Millen, and V. Gilgor. Non-interference, who needs it? In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, pages 237–238, Cape Breton, Nova Scotia, Canada, June 2001. IEEE.
17. A. Sabelfeld and D. Sands. A per model of secure information flow in sequential programs. In *ESOP'99*, number 1576 in Lecture Notes in Computer Science, pages 40–58. Springer Verlag, 1999.
18. A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 200–214, 2000.
19. 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.
20. G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. In *Symposium on Principles of Programming Languages (POPL'98)*, pages 355–364, San Diego, California, 1998. ACM.
21. G. Smith and D. Volpano. Verifying secrets and relative secrecy. In *Symposium on Principles of Programming Languages (POPL'00)*, pages 368–276, Boston, Massachusetts, 2000. ACM.
22. D. Volpano and G. Smith. Probabilistic noninterference in a concurrent language. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop (CSFW '98)*, pages 34–43, Washington - Brussels - Tokyo, June 1998. IEEE.