Concurrent Constraint Programming: Towards Probabilistic Abstract Interpretation

Alessandra Di Pierro
Dipartimento di Informatica
Universitá di Pisa, Italy
dipierro@di.unipi.it

Herbert Wiklicky
Department of Computing
Imperial College, London, UK
herbert@doc.ic.ac.uk

ABSTRACT

We present a method for approximating the semantics of probabilistic programs to the purpose of constructing semantics-based analyses of such programs. The method resembles the one based on Galois connection as developed in the Cousot framework for abstract interpretation. The main difference between our approach and the standard theory of abstract interpretation is the choice of linear space structures instead of order-theoretic ones as semantical (concrete and abstract) domains. We show that our method generates "best approximations" according to an appropriate notion of precision defined in terms of a norm. Moreover, if re-casted in a order-theoretic setting these approximations are correct in the sense of classical abstract interpretation theory. We use Concurrent Constraint Programming as a reference programming paradigm. The basic concepts and ideas can nevertheless be applied to any other paradigm. The results we present are intended to be the first step towards a general theory of probabilistic abstract interpretation, which re-formulates the classical theory in a setting suitable for a quantitative reasoning about programs.

Categories and Subject Descriptors

D.3.1 [**Programming Languages**]: Formal Definitions and Theory—*Semantics*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*Program analysis*; F.1.2 [**Theory of Computation**]: Modes of Computation—*Probabilistic computation*

General Terms

Static analysis, probabilistic concurrent constraint programming, probabilistic abstract interpretation

1. INTRODUCTION

The construction of semantics-based program analysis algorithms or in general the specification and validation of the analyses of programs finds a good formal support in the theory of Abstract Interpretation, which has become very popular since it was introduced in 1977 by Patrick and Radhia Cousot [9].

According to the classical framework an abstract interpretation is obtained by replacing the concrete semantics by an approximating abstract one. The notion of approximation is classically encoded by suitable partial orders on the domains objects, and the correspondence between abstract and concrete domains is expressed by means of an adjoint framework based on the notion of a Galois connection, which ensures the soundness of the approximation.

The ultimate aim of this work is to construct a similar theory whose applications include the analysis of probabilistic programs as well as a probabilistic analysis of standard, i.e. (non)deterministic programs.

Probabilistic computation extends the classical notion of computation by including quantities in the form of real numbers (i.e. probabilities). These quantitative aspects must be taken into account in any semantics for programs which perform probabilistic computation. Various approaches towards the semantics of probabilistic programs have been investigated up to now, each trying to capture the probabilistic feature in a suitable way. Early contributions in this area go back to the fundamental papers of Saheb-Djahromi [34], and Kozen [29]. More recent results are related to probabilistic power-domains [25, 26, 27], probabilistic predicate transformers [31] and stochastic process calculi [2, 15].

In [16, 17] the authors develop a probabilistic version of concurrent constraint programming [35] called Probabilistic Concurrent Constraint Programming (PCCP). The denotational semantics of PCCP is given in terms of a space of probability distributions on the underlying constraint system. These distributions can be represented as vectors in the free real vector space on the constraint system.

In the search for an appropriate domain of denotations we regard linear spaces — much in line with Kozen's work — as very well suited candidates because of the natural combination of quantitative and qualitative aspects implicit in their structure. Unfortunately, the consideration of linear spaces as semantical domains prevents the straightforward use of the classical theory of abstract interpretation for doing semantics-based analyses as it cannot be applied to structures which do not exhibit an explicit notion of order relation (like vector spaces).

In this paper we show how to construct safe abstractions of probabilistic fixpoint semantics by using the notion of an adjoint to a linear map, namely the so-called Moore-Penrose pseudo-inverse [7]. We argue that this notion is an adequate alternative to Galois connections for probabilistic domains. In fact, for a linear map representing an abstraction function between two linear spaces, we can define its Moore-Penrose pseudo-inverse as the corresponding concretisation function. We show that by doing so, we can still achieve conservativity and a form of soundness for the resulting static analyses. Moreover, the nature of linear structures allows for an additional numerical investigation of the optimality of such an analysis as well as the comparison of different abstractions based on their (average) approximation error.

In order to simplify the presentation we will make throughout the paper two basic assumptions: we will consider a synchronisation-free sub-language of PCCP, and we will assume finite constraint systems. These assumptions as well as the choice of PCCP as the reference paradigm are not essential for the presented results, which we believe are still valid in the general case and for other programming paradigms.

The plan of the paper is as follows: In Section 2 we briefly recall the basic constructs of Concurrent Constraint Programming (CCP) and its probabilistic version PCCP together with their denotational semantics. In Section 3 we compare the structural features of Galois connections and Moore-Penrose pseudo-inverse. In Section 4 we then instantiate our framework for the case of lifted abstractions in PCCP. The example in Section 5 illustrates our methodology. Finally, in Section 6 we discuss some possible further developments.

2. THE LANGUAGE

In the following we will concentrate on a particular declarative programming paradigm, that is Concurrent Constraint Programming (CCP) [35], and a probabilistic version of this paradigm previously introduced by the authors in [16, 17], where it is called Probabilistic Concurrent Constraint Programming (PCCP).

2.1 Syntax

The syntax and the basic execution model of PCCP are very similar to CCP [14, 17]. Both languages are based on the notion of a *constraint system* $\mathcal{C}$, defined as a complete algebraic lattice $(\mathcal{C}, \vdash, \sqcup, \mathit{true}, \mathit{false})$. The set of constraints $\mathcal{C}$ is ordered with respect to an entailment relation $\vdash$; $\sqcup$ is the least upper bound (lub) operation, and *true*, *false* are the least and greatest elements of $\mathcal{C}$, respectively (see [35, 14] for more details). For the time being we will consider only *finite* constraint systems; this assumption will allow us to avoid the consideration of complex topological structures for defining the domain for the denotational semantics.

It is worth mentioning that the constraint system $\mathcal{C}$ underlying the probabilistic language PCCP is the same as for classical CCP. This distinguishes our approach from those involving probabilistic or fuzzy constraint systems, e.g. [23] or [6].

A CCP/PCCP program P is an object of the form $D.A$, where D is a set of procedure declarations of the form $p(x)$: $-A$ and A is a CCP/PCCP agent. The syntax of CCP agents is given in Table 1. The PCCP syntax is exactly the same as CCP except that we allow probabilities to appear in both the choice and the parallel constructs, as shown in in Table 2.

A	$::=$	$\mathbf{stop}$	successful termination
	$\mid$	$\mathbf{tell}(c)$	adding a constraint
	$\mid$	$\square_{i=1}^{n}\ \mathbf{ask}(c_i) \rightarrow A_i$	choice
	$\mid$	$\Vert_{i=i}^{n}\ A_i$	parallelism
	$\mid$	$\exists_x A$	hiding, local variables
	$\mid$	$p(x)$	procedure call, recursion

Table 1: The Syntax of CCP Agents

$\square_{i=1}^{n}\ \mathbf{ask}(c_i) \rightarrow p_i : A_i$	probabilistic choice
$\Vert_{i=i}^{n}\ q_i : A_i$	prioritised parallelism

Table 2: PCCP Agents for Choice and Parallelism

2.2 Intuitive Semantics

The operational semantics of CCP is described by means of a transition system [14]. For PCCP we take the same transition system but we extend it by associating to each transition a probability [19, 16]. Such a probability expresses the likelihood that a transition takes place. Therefore, while for deterministic transitions the probability is 1, for the choice and the parallel constructs the transition probability has to be defined according to the distributions p_i and q_i respectively. This can be seen as restricting the original non-determinism by imposing some requirements on the frequency of choices.

The operational meaning of the probabilistic choice construct is as follows: First, check whether constraints c_i are entailed by the store. Then we have to *normalise* the probability distribution by considering only the *enabled* agents, i.e. those agents whose guard c_i is entailed. This means that we have to re-define the probability distribution so as only enabled agents have non-zero probabilities and the sum of these probabilities is one. In general, this can be done by considering for enabled agents the normalised transition probability,

$$\tilde{p}_i = \frac{p_i}{\sum_{\vdash c_j} p_j},$$

where the sum $\sum_{\vdash c_j} p_j$ is over all enabled agents. Finally, one of the enabled agents is chosen according to the new probability distribution $\tilde{p}_i$.

The intuitive semantics of the *prioritised interleaving* is very similar. Again we replace the (implicit) non-determinism of the scheduler, which has to decide in the interleaving semantics which agent has to be executed first, by a probabilistic version. This selection has to be made among all those agents A_i which are *active*, i.e. can make a transition. Then we have to normalise the priorities q_i of the active agents. This normalisation is done in the same way as described above for the probabilistic choice. Finally, the scheduler chooses one of the active agents according to the new probability distribution $\tilde{q}_i$.

As another simplifying assumption, we will ignore for the time being the aspects related to concurrency, whose treatment would require more complicated structures. Therefore, we will concentrate on a synchronisation free version of both CCP and PCCP, that is the sub-languages where the choice constructs are of the form: $[]_{i=1}^{n}\ \mathbf{ask}(true) \rightarrow A_i$ and $[]_{i=1}^{n}\ \mathbf{ask}(true) \rightarrow p_i : A_i$, respectively. This syntactic restriction implies that all agents in a choice are always enabled.

2.3 Denotational Semantics

In this section we will briefly recall some basic aspects of the denotational semantics for CCP and PCCP. For more details we refer to [14] for the semantics of CCP, and to [17] for the semantics of PCCP.

The basic idea of the denotational semantics is to associate to every agent a mathematical description of its computational behaviour. One aspect of a computation one is often interested in is the input/output behaviour. This can be described for CCP by a set of constraints representing the possible final stores after the execution of the agent. For PCCP the input/output observables can be represented by probability distributions on the constraint system, that is sets of pairs $\langle c, p\rangle$, where c is a result and p is the probability of computing that result. For the synchronisation-free languages, such observables can be described compositionally in a simple way.

2.3.1 Semantics of CCP

The denotational semantics of the synchronisation-free version of CCP has an elegant formulation in terms of a Smyth power-domain construction on the constraint system $\mathcal{C}$ [14, 24, 30].

The basic construction of the semantics considers *interpretations*, that is maps from agents into some special subsets of $\mathcal{C}$ describing the results of the agent computation, and a fixpoint operator $\tilde{\Phi}$ on the set of interpretations. A semantics is obtained by iteratively applying $\tilde{\Phi}$ starting from the initial interpretation $\tilde{i}_0$ which assigns to each agent the set $\{true\}$. The definition of $\tilde{\Phi}$ is given in Table 3, where the operation $\bigodot$ is defined for sets $\tilde{i}_j$ of constraints by:

$$\bigodot_j \tilde{i}_j = \left\{ \bigsqcup_j d_j \mid d_j \in \tilde{i}_j \right\}.$$

The semantics of an agent is the fixpoint of $\tilde{\Phi}$ obtained as the limit, $\lim \tilde{\Phi}$, of the sequence

$$\tilde{i}_0, \tilde{\Phi}(\tilde{i}_0), \tilde{\Phi}^2(\tilde{i}_0), \ldots$$

2.3.2 Free Vector Space on $\mathcal{C}$

The denotational semantics of the synchronisation-free sublanguage of PCCP was defined in [17]. The construction

$\tilde{\Phi}(\tilde{i})(\mathbf{stop})$	$=$	$\{true\}$	
$\tilde{\Phi}(\tilde{i})(\mathbf{tell}(c))$	$=$	$\{c\}$	
$\tilde{\Phi}(\tilde{i})([]_{i=1}^{n}\ true \rightarrow A_i)$	$=$	$\bigcup_{i=1}^{n}\ \tilde{\Phi}(\tilde{i})(A_i)$	
$\tilde{\Phi}(\tilde{i})(\Vert_{i=1}^{n}\ A_i)$	$=$	$\bigodot_{i=1}^{n} \tilde{\Phi}(\tilde{i})(A_i)$	
$\tilde{\Phi}(\tilde{i})(\exists_x A)$	$=$	$\{\exists_x c \mid c \in \tilde{\Phi}(\tilde{i})(A)\}$	
$\tilde{\Phi}(\tilde{i})(p(x))$	$=$	$\tilde{i}(A)$	$p(y) : -A \in D$

Table 3: The Definition of $\tilde{\Phi}$ for CCP.

is very similar to the classical one for CCP [14], but interpretations are now probability distributions. We will represent probability distributions as vectors in the free real vector space $\mathcal{V}(\mathcal{C})$ on the constraint system $\mathcal{C}$. This space is constructed as the set of all formal linear combinations of constraints:

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

For finite constraint systems this space is isomorphic to the standard real vector space $\mathbb{R}^n$, where n is the cardinality of $\mathcal{C}$. We can denote a vector in $\mathcal{V}(\mathcal{C})$ equivalently either as a formal sum

$$\vec{x} = \sum_{c \in \mathcal{C}} x_c c,$$

or (once we fix an enumeration of $\mathcal{C}$) as an n-tuple

$$\vec{x} = (x_c)_{c \in \mathcal{C}},$$

or as a set of pairs

$$\vec{x} = \{\langle c, x_c\rangle\}_{c \in \mathcal{C}}.$$

We can introduce a number of standard structures on $\mathcal{V}(\mathcal{C})$ which we inherit from $\mathbb{R}^n$. We can define for example for vectors $\vec{x} \in \mathcal{V}(\mathcal{C})$ a *p-norm* by:

$$\Vert\vec{x}\Vert_p = \Vert(x_c)_{c \in \mathcal{C}}\Vert_p = \left(\sum_{c \in \mathcal{C}} |x_c|^p \right)^{1/p}$$

which can be used to define a norm topology on $\mathcal{V}(\mathcal{C})$.

Another standard structure given on finite dimensional vector spaces which we will utilise on $\mathcal{V}(\mathcal{C})$ is the notion of an *inner product*:

$$\langle \vec{x}, \vec{y}\rangle = \langle (x_c)_{c \in \mathcal{C}}, (y_c)_{c \in \mathcal{C}}\rangle = \sum_{c \in \mathcal{C}} x_c y_c.$$

This notion provides an alternative characterisation of the 2-norm, namely

$$\Vert\vec{x}\Vert_2 = \sqrt{\langle \vec{x}, \vec{x}\rangle}.$$

This highlights the special role of the 2-norm among the other p-norms. It is often referred to as the standard *Euclidean* distance and we will use it later in order to define a notion of "precision" for abstract interpretations.

Every norm on a vector space defines a topology on the vector space [20]. In the finite dimensional case all p-norm topologies are equivalent, i.e. if a sequence converges with respect to one norm it also converges with respect to all the other p-norms [20]. Therefore, we can refer to *the* topology of $\mathcal{V}(\mathcal{C})$ for all the topological notions we will use in the following without specifying any particular norm.

The inner product also allows us to define the unique *adjoint* α^* of a linear map $\alpha : \mathcal{V}(\mathcal{C}) \mapsto \mathcal{V}(\mathcal{C})$ via [33]:

$$\langle \alpha(\vec{x}), \vec{y} \rangle = \langle \vec{x}, \alpha^*(\vec{y}) \rangle .$$

If α is represented in the usual way by a matrix A, the matrix corresponding to α^* is given by its transpose A^t, i.e. the matrix obtained from A by exchanging rows and columns $(a_{ij})^t = a_{ji}$.

The effective domain we need to consider is actually only a subset of $\mathcal{V}(\mathcal{C})$ corresponding to the so called *simplex*, that is the subset of all vectors whose coordinates sum up to one, namely those vectors whose 1−norm is one:

$$\mathcal{S} = \{\vec{x} \in \mathcal{V}(\mathcal{C}) \mid x_i \geq 0 \text{ and } \|\vec{x}\|_1 = 1\} .$$

Note that, because of the assumption of finite constraint systems, we can identify the simplex with the space of measures on $\mathcal{C}$ as these can be represented by distributions [4]. This assumption also guarantees that $\mathcal{S}$ is a bounded and closed (therefore compact), and convex subset of $\mathcal{V}(\mathcal{C})$.

2.3.3 Fixpoint Operator for PCCP

We will construct the semantics of PCCP by following the standard method in denotational semantics. We define the meaning of PCCP agents compositionally via a set of recursive equations over the free vector space. The intuition behind these equations is as follows.

The agent **stop** can only produce an empty store with probability 1, thus we get:

$$\vec{i}(\mathbf{stop}) = \{\langle true, 1\rangle\}.$$

Analogously, $\mathbf{tell}(c)$ always adds c to the store with probability 1, which results in the equation

$$\vec{i}(\mathbf{tell}(c)) = \{\langle c, 1\rangle\}.$$

The probability 1 expresses the fact that these basic constructs are deterministic.

For the choice construct $[]_{i=1}^n \; true \; \rightarrow p_i : A_i$ we note that all the agents A_i are always enabled (because of the absence of synchronisation); therefore the meaning of a probabilistic choice agent is the linear combination of the vectors associated to each alternative A_i:

$$\vec{i}([]_{i=1}^n \; true \; \rightarrow p_i : A_i) = \sum_{i=1}^n \; p_i \cdot \vec{i}(A_i)$$

that is their sum weighted by the corresponding probabilities p_i's.

The parallel agent $\|_{i=1}^n \;\; q_i : A_i$ produces a constraint $c = \bigsqcup_i d_i$ which is the conjunction (least upper bound) of some constraints produced by each of the agents A_i; the probability is the product $p = \prod_i p_i$ of the probabilities associated to each of such constraints. Formally,

$$\vec{i}(\|_{i=1}^n \;\; q_i : A_i) = \bigotimes_{i=1}^n \; q_i \cdot \vec{i}(A_i),$$

where $\bigotimes$ is the tensor product. This is defined for vectors $\vec{x}_j \in \mathcal{V}(\mathcal{C})$ by:

$$\bigotimes_j \vec{x}_j = \left\{ \left\langle \bigsqcup_j d_j, \prod_j p_j \right\rangle \mid \langle d_j, p_j\rangle \in \vec{x}_j \right\}.$$

Hiding is modelled by means of the operator $\exists\!\!\!\exists_x$ defined for a generic interpretation $\{\langle c_i, p_i\rangle\}_i$ by:

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

Intuitively, the constraints computed by $\exists_x A$ are the constraints of A with the variable x hidden from the other agents:

$$\vec{i}(\exists_x A) = \exists\!\!\!\exists_x \vec{i}(A)$$

The last equation defining the meaning of the procedure call $p(x)$ is obvious: The semantics of a procedure call is given by the semantics of the agent in the body of its definition $p(y) : -A \in D$, i.e.

$$\vec{i}(p(x)) = \vec{i}(A)$$

We abstract here from the technicalities needed to model the parameters passing and we assume that this has been correctly done before replacing $p(x)$ by its definition (see [14] for more details).

$\vec{\Phi}(\vec{i})(\mathbf{stop})$	$=$	$\{\langle true, 1\rangle\}$
$\vec{\Phi}(\vec{i})(\mathbf{tell}(c))$	$=$	$\{\langle c, 1\rangle\}$
$\vec{\Phi}(\vec{i})([]_{i=1}^n \; true \; \rightarrow p_i : A_i)$	$=$	$\sum_{i=1}^n \; p_i \cdot \vec{\Phi}(\vec{i})(A_i)$
$\vec{\Phi}(\vec{i})(\|_{i=1}^n \;\; q_i : A_i)$	$=$	$\bigotimes_{i=1}^n \; q_i \cdot \vec{\Phi}(\vec{i})(A_i)$
$\vec{\Phi}(\vec{i})(\exists_x A)$	$=$	$\exists\!\!\!\exists_x \vec{i}(A)$
$\vec{\Phi}(\vec{i})(p(x))$	$=$	$\vec{i}(A)$

Table 4: The Definition of $\vec{\Phi}$ for PCCP.

The semantics of a PCCP agent is then described by a fixpoint operator $\vec{\Phi}$ on the set of interpretations on $\mathcal{V}(\mathcal{C})$, which encodes these equations. The definition of $\vec{\Phi}$ is given in Table 4. The operator $\vec{\Phi}$ can be shown to be linear, thus continuous. Moreover it leaves the simplex invariant, i.e. for all $\vec{i} \in \mathcal{S}$ its image $\vec{\Phi}(\vec{i}) \in \mathcal{S}$.

The construction of a fixpoint for Φ mimics the classical fixpoint construction: Starting with the initial interpretation $\vec{i}_0$ assigning to each agent A the distribution $\vec{i}_0(A) = \{\langle true, 1\rangle\}$, we iteratively apply Φ in order to construct the sequence of interpretations

$$\{\vec{i}_n\}_n = \vec{i}_0, \Phi(\vec{i}_0), \Phi^2(\vec{i}_0), \ldots, \Phi^n(\vec{i}_0), \ldots$$

This sequence converges pointwise, and the continuity of $\vec{\Phi}$ ensures that $\lim \vec{\Phi}$ is a fixpoint of $\vec{\Phi}$. Moreover we can show that this semantics captures the operational notion of probabilistic input/output observables [17].

2.3.4 *Relationship between CCP and PCCP*

Probabilistic nondeterminism is a particular case of pure nondeterminism, where the choice is not completely arbitrary but is governed by some established probability distribution. Therefore, while it is immediate to retrieve the CCP semantics from the PCCP one (by just ignoring the information about probabilities), the reverse operation necessarily involves a restriction on the nondeterministic scheduler. In order to make these considerations more formal we introduce the following operations:

Given a vector $\vec{x} = \{\langle c_i, p_i\rangle\} \in \mathcal{V}(\mathcal{C})$, we define the *support set*, $supp(\vec{x})$ of $\vec{x}$ as the set

$$supp(\vec{x}) = \{c_i \mid \langle c_i, p_i\rangle \in \vec{x} \text{ and } p_i \neq 0\}.$$

Vice versa for a set $\tilde{x} = \{c_i\} \in \mathcal{P}(\mathcal{C})$, we define a (uniform) distribution vector

$$vec(\tilde{x}) = \left\{\left\langle c_i, \frac{1}{|\tilde{x}|}\right\rangle\right\}$$

where $|\tilde{x}|$ denotes the cardinality of $\tilde{x}$.

The operation $supp$ allows us to retrieve the semantics of a CCP agent from the semantics of the corresponding PCCP agent. In particular, the following relations hold for the semantical equations of the choice and the parallel constructs in CCP and PCCP:

$$\bigcup_i supp(\vec{x}_i) = supp\left(\sum_i q_i\vec{x}_i\right)$$

and

$$\bigodot_i supp(\vec{x}_i) = supp\left(\bigotimes_i q_i\vec{x}_i\right),$$

whenever for all i the weights $q_i \neq 0$.

In general, it can be shown that for finite computations the semantics of a CCP agent (e.g. its input/output observables or equivalently its denotation in terms of the fixpoint operator $\tilde{\Phi}$), can be retrieved by taking the support set of the distribution representing the corresponding PCCP agent. This translation can be seen as an abstraction of the PCCP semantics.

As for the operation vec, it allows us to translate a CCP agent into a PCCP one by assuming a uniform distribution on the choice. In fact, the operations vec and $supp$ could be seen as a kind of abstraction/concretisation pair which one could use to define the CCP semantics as an abstract interpretation of the PCCP one (see Section 6).

3. ABSTRACT INTERPRETATION

Abstract Interpretation — as introduced by Cousot in the late 1970s [10] — is a theory which provides systematic methods to design abstract semantics that correctly approximate some given concrete semantics. It represents a unifying framework for designing and then validating static analyses of programs. This approach is essentially based on qualitative notions such as the one of partially ordered domains. Our aim is to transfer this general theory to a quantitative setting as we encounter it in the semantics of probabilistic languages, and in particular of PCCP.

3.1 Approximations

The basic idea behind abstract interpretation is to analyse a program not in terms of its standard or *concrete semantics*, but rather in terms of an appropriately simplified approximated or *abstract semantics*, which only registers aspects of the program which are of relevance with respect to a specific analysis (cf. [11, 1, 32]). Typically these aspects are encoded in the definition of an abstract domain, which is usually structured, like the concrete domain, as a complete partially ordered set.

The ordering on these domains expresses the notion of approximation, i.e. relative precision of the domain values: $x \leq y$ means that x is more precise than y. Note that in the domains of the standard denotational semantics the ordering is usually interpreted in a dual way: $x \leq y$ means here that x is an approximation of y. In the following we will denote the order on a domain $\mathcal{D}$ by $\leq_{\mathcal{D}}$. On an abstract domain, abstract operations are defined which simulate the behaviour of their concrete counterparts. Thus, given a concrete semantic operation $f : \mathcal{C} \mapsto \mathcal{A}$, an abstract interpretation is defined by specifying corresponding abstract domains $\mathcal{D}$ and $\mathcal{B}$ and an abstract function $f^{\mathcal{D}} : \mathcal{D} \mapsto \mathcal{B}$.

The translation between the concrete and the abstract semantics of a program is achieved via pairs of adjoint maps, called Galois connections (see Section 3.2). Such pairs relate the domains $\mathcal{C}$ and $\mathcal{D}$ and $\mathcal{A}$ and $\mathcal{B}$ respectively and each of them consists of an *abstraction* function α and a *concretisation* function γ. The first function associates to each concrete element in $\mathcal{C}$ (or $\mathcal{A}$) its abstract denotation in $\mathcal{D}$ (or $\mathcal{B}$), while the second assigns to abstract elements in $\mathcal{D}$ (or $\mathcal{B}$) their concrete values in $\mathcal{C}$ (or $\mathcal{A}$).

It is common in programming languages to define the concrete semantics via a fixpoint operator Φ on some particular (concrete) domain $\mathcal{C}$, e.g. as explained in Section 2.3 for CCP and PCCP. For our treatment we will concentrate on this case of fixpoint semantics. Then an abstract interpretation is specified by an abstract domain $\mathcal{D}$ and an abstract operator Ψ on $\mathcal{D}$ which reflects in some appropriate way what is happening in the concrete domain.

3.2 Galois Connections

In the standard Cousot and Cousot theory, the correctness of an abstract interpretation is guaranteed by ensuring that the pair of functions α and γ forms a *Galois connection*:

DEFINITION 1. *Let $\mathcal{C} = (\mathcal{C}, \leq_{\mathcal{C}})$ and $\mathcal{D} = (\mathcal{D}, \leq_{\mathcal{D}})$ be two partially ordered set. If there are two functions $\alpha : \mathcal{C} \mapsto \mathcal{D}$ and $\gamma : \mathcal{D} \mapsto \mathcal{C}$ such that for all $c \in \mathcal{C}$ and all $d \in \mathcal{D}$:*

$$c \leq_{\mathcal{C}} \gamma(d) \text{ iff } \alpha(c) \leq_{\mathcal{D}} d,$$

then $(\mathcal{C}, \alpha, \gamma, \mathcal{D})$ forms a Galois connection.

The intended meaning is that an abstract element d approximates a concrete one c if $c \leq_{\mathcal{C}} \gamma(d)$ or equivalently (by adjunction) if $\alpha(c) \leq_{\mathcal{D}} d$. Therefore, the concrete value corresponding to an abstract denotation d is $\gamma(d)$, while the adjunction guarantees that $\alpha(c)$ is the best possible approximation of c in $\mathcal{D}$ (because whenever d is a correct approximation of c, then $\alpha(c) \leq_{\mathcal{D}} d$).

An alternative characterisation of a Galois connection is as follows:

THEOREM 1. *Let $\mathcal{C} = (\mathcal{C}, \leq_{\mathcal{C}})$ and $\mathcal{D} = (\mathcal{D}, \leq_{\mathcal{D}})$ be two partially ordered set together with two functions $\alpha : \mathcal{C} \mapsto \mathcal{D}$ and $\gamma : \mathcal{D} \mapsto \mathcal{C}$. Then $(\mathcal{C}, \alpha, \gamma, \mathcal{D})$ form a Galois connection iff*

1. *α and γ are order-preserving,*
2. *$\alpha \circ \gamma$ is reductive (i.e. for any $d \in \mathcal{D}$, $\alpha \circ \gamma(d) \leq_{\mathcal{D}} d$),*
3. *$\gamma \circ \alpha$ is extensive (i.e. for any $c \in \mathcal{C}$, $c \leq_{\mathcal{C}} \gamma \circ \alpha(c)$).*

A further important property of Galois connections guarantees that the approximation of a concrete semantics by means of two functions α and γ related by a Galois connection is not only safe but also *conservative* in as far as repeating the abstraction or the concretisation gives the same results as by a single application of these functions. This property is expressed by the following proposition:

PROPOSITION 1. *Let $(\mathcal{C}, \alpha, \gamma, \mathcal{D})$ be a Galois connection. Then α and γ are* quasi-inverse*, i.e.*

1. *$\alpha \circ \gamma \circ \alpha = \alpha$, and*
2. *$\gamma \circ \alpha \circ \gamma = \gamma$.*

3.3 Correctness and Completeness

Like for any approximation technique, *correctness* is a basic requirement for an abstract interpretation. In the scenario described in Section 3.1, an abstract function $f^{\mathcal{D}} : \mathcal{D} \mapsto \mathcal{B}$ is a *correct* approximation of a concrete function $f : \mathcal{C} \mapsto \mathcal{A}$ if

$$\alpha \circ f \leq_{\mathcal{A}} f^{D} \circ \alpha,$$

where by abuse of notation we indicate by α both the abstraction function from $\mathcal{A}$ to $\mathcal{B}$ on the left hand side of the relation, and the abstraction function from $\mathcal{C}$ to $\mathcal{D}$ on the right hand side of the relation.

In the typical situation in which the concrete semantics is given in terms of a fixpoint operator Φ on the concrete domain $\mathcal{C}$, the correctness of an abstract interpretation defined by $(\mathcal{D}, \Psi)$ is expressed by the following condition which is often referred to as *fixpoint correctness* [22]:

$$\alpha \circ fix(\Phi) \leq_{\mathcal{D}} fix(\Psi) \circ \alpha,$$

where fix denotes a fixpoint of the semantical operator. It is well known that any abstract domain $\mathcal{D}$ induces the so-called *best correct approximation* of Φ [10] defined by

$$\Psi = \alpha \circ \Phi \circ \gamma. \tag{1}$$

Note that the choice of the best correct approximation of a function does not guarantee that the results of an abstract computation are best approximations of the corresponding concrete ones. Whenever this happens then the abstract interpretation is *complete*.

Unlike the correctness condition, completeness is not an essential requirement, but rather an ideal situation which does not occur very often in practice. In the literature it is often also referred to as exactness [8] or optimality [13].

Completeness means for an abstract interpretation that in the above relations for correctness the equality holds, so that the abstraction results in no loss of information. It is a dual notion to correctness; in particular *fixpoint completeness* is dual to fixpoint correctness and is expressed by

$$\alpha \circ fix(\Phi) = fix(\Psi) \circ \alpha$$

A basic result of the abstract interpretation theory ensures that a (fixpoint) complete abstract semantic operator can always be defined whenever the best correct approximation of the operator is (fixpoint) complete. This means that for a Ψ defined as in (1) we get a fixpoint complete abstract interpretation if and only if

$$\alpha \circ fix(\Phi) = \alpha \circ fix(\Phi) \circ \gamma \circ \alpha,$$

that is if and only if $\gamma \circ \alpha$ is the identity function.

3.4 Generalised Inverse

In a vector space there is no obvious natural order, therefore it is not possible to directly translate Galois connections into the context of a denotational semantics based on distributions. In this section we introduce the notion of a *generalised inverse* (or *pseudo-inverse*) of a linear mapping, and we show that it is an appropriate substitute for Galois connections in the probabilistic setting. The properties of the pseudo-inverse are best explained in terms of the notion of *projection*.

DEFINITION 2. *Given a finite dimensional vector space $\mathcal{V}$, a linear map $\pi : \mathcal{V} \mapsto \mathcal{V}$ is called a* projection *if $\pi^2 = \pi \circ \pi = \pi$. A projection is* orthogonal *iff $\pi = \pi^*$, where $.^*$ denotes the adjoint.*

Projections allow for a unique "decomposition" of every vector in $\mathcal{V}$ [33, Sect. 25]:

PROPOSITION 2. *Let π be a projection on $\mathcal{V}$. Then every vector $\vec{x} \in \mathcal{V}$ can be represented uniquely as $\vec{x} = \vec{x}_\perp + \vec{x}_\parallel$ with $\pi(\vec{x}_\perp) = \vec{o}$ and $\pi(\vec{x}_\parallel) = \vec{x}_\parallel$.*

There is a one-to-one correspondence between orthogonal projections and sub-spaces of a vector space $\mathcal{V}$: For every sub-space $\mathcal{W} \subseteq \mathcal{V}$ there is a (unique) orthogonal projection $\pi_{\mathcal{W}}$ whose image is $\mathcal{W}$, and vice versa.

A property of orthogonal projections, which is very important in our context, is expressed by the following proposition [7, Corr 2.1.1]:

PROPOSITION 3. *Let $\pi_{\mathcal{W}}$ be the orthogonal projection on a finite dimensional vector space $\mathcal{V}$. Then for every vector $\vec{x} \in \mathcal{V}$ its image $\pi_{\mathcal{W}}(\vec{x})$ is the unique vector in $\mathcal{W}$ such that $\|\vec{x} - \pi_{\mathcal{W}}(\vec{x})\|_2$ is minimal.*

By recalling that the 2-norm is the standard norm expressing in a finite dimensional vector space the Euclidean distance among the points of the space, Proposition 3 says that the orthogonal projection $\pi_{\mathcal{W}}(\vec{x})$ of any vector $\vec{x}$ in $\mathcal{V}$ is the unique vector in $\mathcal{W}$ which is the "closest" one to $\vec{x}$. This suggests that projections are good candidates for representing what in the classical abstract interpretation is called the *best approximation* of a concrete object c, namely the most precise among all the abstract objects that correctly approximate c.

We can now introduce the notion of pseudo-inverse which we will use as a probabilistic analogue of Galois connections. Penrose's definition of pseudo-inverse from the 1950s comes from the following theorem [7, Def. 1.1.3].

THEOREM 2. *Let $\mathcal{C}$ and $\mathcal{D}$ be two finite dimensional vector spaces. For a linear map $\alpha : \mathcal{C} \mapsto \mathcal{D}$ there exists a unique map $\gamma : \mathcal{D} \mapsto \mathcal{C}$ such that:*

1. $\alpha \circ \gamma \circ \alpha = \alpha$,
2. $\gamma \circ \alpha \circ \gamma = \gamma$,
3. $(\alpha \circ \gamma)^* = \alpha \circ \gamma$,
4. $(\gamma \circ \alpha)^* = \gamma \circ \alpha$.

The map γ is called the *Moore-Penrose pseudo-inverse* and is usually denoted by $\alpha^\dagger$. An alternative characterisation of the Moore-Penrose pseudo-inverse corresponds to Moore's original definition and dates back to the 1930s. It is given by the following theorem [7, Def. 1.1.2]:

THEOREM 3. *Let $\mathcal{C}$ and $\mathcal{D}$ be two finite dimensional vector spaces and $\alpha : \mathcal{C} \mapsto \mathcal{D}$ a linear map between them. A linear map $\gamma : \mathcal{D} \mapsto \mathcal{C}$ is the unique Moore-Penrose pseudo-inverse of A iff*

1. $\alpha \circ \gamma = \pi_\alpha$, *and*
2. $\gamma \circ \alpha = \pi_\gamma$.

where $\pi_\alpha : \mathcal{C} \mapsto \mathcal{C}$ is the orthogonal projection into the range of α and $\pi_\gamma : \mathcal{D} \mapsto \mathcal{D}$ is the orthogonal projection into the range of γ.

It can be shown that the two definitions are equivalent [7, Thm. 1.1.1]. They are often presented in terms of matrices A and $A^\dagger = \Gamma$ representing the linear maps α and $\alpha^\dagger = \gamma$; in this case A^* is simply the transpose of the matrix A. However, the "functional" formulation is more convenient in our case as it allows for an easier comparison with Galois connections.

4. CONSTRUCTION OF PROBABILISTIC ABSTRACT INTERPRETATIONS

We now show how to construct an abstract interpretation starting from a concrete (denotational) semantics in the context of constraint programming. For PCCP programs we will obtain a semantics based on vector spaces. The same construction applied to the standard semantics of CCP programs will result in a power set construction.

4.1 Lifting

4.1.1 Power-Set Lifting

Given a monotone extraction function $\alpha : \mathcal{C} \mapsto \mathcal{D}$ we can define an abstraction function between the power sets $\mathcal{P}(\mathcal{C})$ and $\mathcal{P}(\mathcal{D})$ by:

$$\tilde{\alpha}(\{c_1, c_2, \ldots\}) = \{\alpha(c_1), \alpha(c_2), \ldots\}.$$

Obviously the power sets $\mathcal{P}(\mathcal{C})$ and $\mathcal{P}(\mathcal{D})$ carry a natural order structure stemming from set inclusion. This allows us to instantiate the Galois framework for abstract interpretation as described above, i.e. starting from the lifted abstraction $\tilde{\alpha} : (\mathcal{P}(\mathcal{C}), \subseteq) \mapsto (\mathcal{P}(\mathcal{D}), \subseteq)$ we can construct the unique concretisation $\tilde{\gamma} : (\mathcal{P}(\mathcal{D}), \subseteq) \mapsto (\mathcal{P}(\mathcal{C}), \subseteq)$ such that $\tilde{\alpha}$ and $\tilde{\gamma}$ form a Galois connection. For details on this type of construction in a constraint programming framework see e.g. [21].

4.1.2 Vector Space Lifting

An obvious way to lift an extraction function $\alpha : \mathcal{C} \mapsto \mathcal{D}$ to a function from the vector space $\mathcal{V}(\mathcal{C})$ into $\mathcal{V}(\mathcal{D})$ is to define it as a linear map:

$$\vec{\alpha}(p_1 \cdot c_1 + p_2 \cdot c_2 + \ldots) = p_i \cdot \alpha(c_1) + p_2 \cdot \alpha(c_2) \ldots$$

We can now construct the unique concretisation $\vec{\gamma} : \mathcal{V}(\mathcal{D}) \mapsto \mathcal{V}(\mathcal{C})$ such that $\vec{\alpha}$ and $\vec{\gamma}$ are Moore-Penrose pseudo-inverse to each other. We will call the linear maps $\vec{\alpha}$ *probabilistic abstraction functions* and refer to an abstract interpretation defined via a pair $(\vec{\alpha}, \vec{\alpha}^\dagger)$ as a *probabilistic abstract interpretation.*

It is easy to see the relation between $\tilde{\alpha}$ and $\vec{\alpha}$ constructed from the same extraction function $\alpha : \mathcal{C} \mapsto \mathcal{D}$: For all subsets $\tilde{x}$ of $\mathcal{C}$:

$$supp(\vec{\alpha}(\text{vec}(\tilde{x}))) = \tilde{\alpha}(\tilde{x}).$$

4.2 Pseudo-Inverse and Safe Abstractions

Given a probabilistic abstraction function α, the property (1) and (2) in Theorem 2 of α and its pseudo-inverse γ correspond to the properties of Galois connections in Proposition 1 and makes the corresponding probabilistic abstract interpretation *conservative.*

Correctness can be formulated in our setting in terms of projections. In particular, as already mentioned in Section 3.4, orthogonal projections allow us to define the "best" approximation in the sense of the vector whose distance from the concrete one is minimal. In fact, for an abstract fixpoint operator defined as $\Psi = \alpha \circ \Phi \circ \gamma$ we can conclude that whenever α and γ are Moore-Penrose pseudo-inverse, the approximation defined by the abstract operator Ψ is the *closest correct approximation* of Φ in the sense that for all $\vec{d} \in \mathcal{D}$ the distance

$$\|\gamma \circ fix(\Psi)(\vec{d}) - \gamma \circ \alpha \circ fix(\Phi) \circ \gamma(\vec{d})\|_2$$

is minimal (by Theorem 3 and Proposition 3).

For a probabilistic abstraction we can also show its soundness in the classical sense. In fact, the following proposition shows that a probabilistic abstract interpretation is "safe" with respect to the inclusion order on the support sets of vectors.

PROPOSITION 4. *Let $\vec{\alpha}$ be a probabilistic abstraction function and let $\vec{\gamma}$ be its Moore-Penrose pseudo-inverse. Then $\vec{\gamma} \circ \vec{\alpha}$ is extensive with respect to the inclusion on the support sets of vectors in $\mathcal{V}(\mathcal{C})$, i.e. $\forall \vec{x} \in \mathcal{V}(\mathcal{C})$,*

$$supp(\vec{\gamma} \circ \vec{\alpha}(\vec{x})) \supseteq supp(\vec{x}).$$

Proof: Let us first look at the effect of $\vec{\gamma} \circ \vec{\alpha}$ on just the base vectors in $\mathcal{V}(\mathcal{C})$, i.e. on $\vec{c}_i$. We claim that the support of $\vec{\gamma} \circ \vec{\alpha}(\vec{c}_i)$ is larger than the support of $\vec{c}_i$. If we assumed that $\vec{c}_i$ is "perpendicular" to the projection $\vec{\gamma} \circ \vec{\alpha}$, i.e. it is mapped to the null vector $\vec{o}$, than this would imply that $\vec{\alpha}(\vec{c}_i) = \vec{\alpha} \circ \vec{\gamma} \circ \vec{\alpha}(\vec{c}_i) = \vec{\alpha}(\vec{o}) = \vec{o}$. Each base vector $\vec{c}_i$ must therefore have a non-zero "parallel" component, i.e. it can be written as $\vec{c}_i = \vec{c}_{i\perp} + \vec{c}_{i\|}$ with $\vec{c}_{i\|} \neq \vec{o}$ or equivalently $\vec{c}_{i\|} = \vec{c}_i - \vec{c}_{i\perp}$. The projection of $\vec{c}_i$ under $\vec{\gamma} \circ \vec{\alpha}$ is $\vec{\gamma} \circ \vec{\alpha}(\vec{c}_i) = \vec{\gamma} \circ \vec{\alpha}(\vec{c}_{i\perp} + \vec{c}_{i\|}) = \vec{\gamma} \circ \vec{\alpha}(\vec{c}_{i\|}) = \vec{c}_{i\|}$. As we know that $\vec{c}_{i\|}$ can be represented (uniquely) as $\vec{c}_{i\|} = \vec{c}_i + \ldots$, we know that c_i must be in the support set of $\vec{c}_{i\|}$ and thus of $\vec{\gamma} \circ \vec{\alpha}(\vec{c}_i)$. We conclude that for all base vectors the projection $\vec{\gamma} \circ \vec{\alpha}$ can only increase the support set, i.e.

$$supp(\vec{\gamma} \circ \vec{\alpha}(\vec{c}_i)) \supseteq supp(\vec{c}_i)$$

which by linearity can be extended to all vectors:

$$supp(\vec{\gamma} \circ \vec{\alpha}(\vec{x})) = supp(\vec{\gamma} \circ \vec{\alpha}(\sum_i x_i \cdot c_i)) \supseteq supp(\vec{x}) \quad \Box$$

Analogously, we can show that $\vec{\alpha} \circ \vec{\gamma}$ is reductive. Therefore, $(\vec{\alpha}, \vec{\gamma})$ form a Galois connection with respect to the support sets of the vectors in $\mathcal{V}(\mathcal{C})$ and $\mathcal{V}(\mathcal{D})$, ordered by inclusion.

4.3 Comparison of Abstractions

The Moore-Penrose pseudo-inverse method gives us the possibility to *measure* the quality of a probabilistic abstract interpretation. The key idea is to look at the difference between the two distributions: $\gamma \circ fix(\Psi)(\vec{d})$ and $\gamma \circ \alpha \circ fix(\Phi) \circ \gamma(\vec{d})$. This difference is obviously determined by the "size" of the map (or matrix) $\gamma \circ \alpha$ describing the approximation error. We are actually only interested in normalised vectors which represent distributions, i.e. vectors whose 1-norm is exactly one. Thus, we can use the corresponding *operator norm* to obtain a measure for the (worst) approximation error. This norm is defined for an operator α as:

$$\|\alpha\|_1 = \sup_{\|x\|_1 = 1} \|\alpha(x)\|_1,$$

i.e. as the length of the "longest" vector a unit vector may be mapped to. Note that when $\gamma \circ \alpha$ is the identity matrix (i.e. the abstract interpretation is complete) its norm is 1. This can be interpreted as the fact that in this case we get 100% precision. We refer to [18] for a more detailed treatment of this issue.

5. AN EXAMPLE

We give here an example of an abstract interpretation for a simple probabilistic program which generates natural numbers. The approximation is aimed at representing the parity of the numbers generated by the program. We will first present a detailed description of the construction of the abstract interpretation, and then we give some results of numerical calculations on finite matrices representing the involved linear mappings in a truncated version of the program semantics.

5.1 Counting in CCP and PCCP

We illustrate our methodology by means of a simple example of analysis of a probabilistic program. The program consists of a simple procedure which outputs natural numbers with some probability calculated during the execution of a probabilistic choice. We will compare the probabilistic analysis of such a program with the classical analysis of a nondeterministic version of the program. The analysis is based on an abstraction which ignores the actual value of the generated numbers and only considers their even/odd property.

Note that the constraint system is not finite in this example. We can nevertheless guarantee that all semantical constructions for the finite case still hold for this particular case.

A purely nondeterministic version of a number-generator program, can be realised by means of the following CCP procedure:

$$\begin{array}{rcl} nat(x) & :- & true \rightarrow \mathbf{tell}(x=0) \\ & [] & true \rightarrow \exists_y(\mathbf{tell}(x=s(y)) \\ & & \quad \| \; nat(y)) \end{array}$$

A probabilistic version of this program can be written in PCCP as follows:

$$\begin{array}{rcl} pnat(x) & :- & true \rightarrow \frac{1}{2} : \mathbf{tell}(x=0) \\ & [] & true \rightarrow \frac{1}{2} : \exists_y(\frac{1}{2} : \mathbf{tell}(x=s(y)) \\ & & \quad \| \; \frac{1}{2} : pnat(y)) \end{array}$$

5.1.1 Concrete Denotational Semantics

The denotational semantics of these two programs can be constructed as described in Section 2.3 by means of the fixpoint operator $\tilde{\Phi}$, for the nondeterministic version, and $\vec{\Phi}$, for the probabilistic version. We will use the following shorthand notation for the constraints involved.

$$\begin{array}{rcl} x=0 & \equiv & 0 \\ x=s(0) & \equiv & 1 \\ x=s(s(0)) & \equiv & 2 \\ & \ldots & \\ \exists_y x = s(y) & \equiv & * \end{array}$$

It is then straightforward to see that we get the following approximation sequence for the denotational semantics of nat, starting with $\tilde{i}_0 = \{true\}$:

$$\begin{array}{rcl} \tilde{\Phi}^1(\{true\})(nat) & = & \{0,*\} \\ \tilde{\Phi}^2(\{true\})(nat) & = & \{0,1,*\} \\ \tilde{\Phi}^3(\{true\})(nat) & = & \{0,1,,2,*\} \\ \tilde{\Phi}^4(\{true\})(nat) & = & \{0,1,,2,3,*\} \\ \ldots & \ldots & \ldots \end{array}$$

and therefore the fixpoint of Φ as:

$$\lim \tilde{\Phi} = \{0, 1, 2, \ldots, \omega\}.$$

Likewise, for the probabilistic version of the counting program we get — starting with $\vec{i}_0 = \{\langle true, 1\rangle\}$ — the following sequence:

$$\begin{array}{rcl}
\vec{\Phi}^1(\{\langle true, 1\rangle\})(pnat) & = & \{\langle 0, \frac{1}{2}\rangle, \langle *, \frac{1}{2}\rangle\} \\
\vec{\Phi}^2(\{\langle true, 1\rangle\})(pnat) & = & \{\langle 0, \frac{1}{2}\rangle, \langle 1, \frac{1}{4}\rangle, \langle *, \frac{1}{4}\rangle\} \\
\vec{\Phi}^3(\{\langle true, 1\rangle\})(pnat) & = & \{\langle 0, \frac{1}{2}\rangle, \langle 1, \frac{1}{4}\rangle, \langle 2, \frac{1}{8}\rangle, \langle *, \frac{1}{8}\rangle\} \\
\vec{\Phi}^4(\{\langle true, 1\rangle\})(pnat) & = & \{\langle 0, \frac{1}{2}\rangle, \langle 1, \frac{1}{4}\rangle, \langle 2, \frac{1}{8}\rangle, \langle 3, \frac{1}{16}\rangle, \\
 & & \langle *, \frac{1}{16}\rangle\} \\
\ldots & \ldots & \ldots
\end{array}$$

and thus the fixpoint:

$$\lim \vec{\Phi} = \{\langle 0, 1/2\rangle, \langle 1, 1/4\rangle, \langle 2, 1/8\rangle, \ldots\}$$

5.1.2 An Abstraction

The following mapping describes an abstraction which ignores which number is indeed computed and is only interested in whether the result of a computation is even or odd.

$$\begin{array}{rcl}
\alpha(true) & = & t \\
\alpha(2k) & = & e \\
\alpha(2k+1) & = & o \\
\alpha(*) & = & \star \\
\alpha(false) & = & f
\end{array}$$

The mapping α is defined between the concrete constraint system $\mathcal{C}$, into an abstract, descriptional constraint system $\mathcal{D}$. Where $\mathcal{C}$ contains all numbers and the non-ground term $*$ as well as *true* and *false* as bottom and top element; $\mathcal{D}$ contains only 5 elements: t and f as bottom and top and e, even, and o, odd, as well as $\star$ to abstract from $*$.

5.1.3 Abstraction of the Concrete Semantics

We now lift the abstraction α to the denotational domains. The abstract sequence of denotations are as follows. For the purely qualitative case:

$$\begin{array}{rcl}
\tilde{\alpha}\tilde{\Phi}^0\,\{true\}\,(nat) & = & \{t\} \\
\tilde{\alpha}\tilde{\Phi}^1\,\{true\}\,(nat) & = & \{e, \star\} \\
\tilde{\alpha}\tilde{\Phi}^2\,\{true\}\,(nat) & = & \{e, o, \star\} \\
\tilde{\alpha}\tilde{\Phi}^3\,\{true\}\,(nat) & = & \{e, o, \star\} \\
\tilde{\alpha}\tilde{\Phi}^4\,\{true\}\,(nat) & = & \{e, o, \star\} \\
\ldots & \ldots & \ldots
\end{array}$$

and for the quantitative case:

$$\begin{array}{rcl}
\vec{\alpha}\vec{\Phi}^0\,\{\langle true, 1\rangle\}\,(pnat) & = & \{\langle t, 1\rangle\} \\
\vec{\alpha}\vec{\Phi}^1\,\{\langle true, 1\rangle\}\,(pnat) & = & \{\langle e, \frac{1}{2}\rangle, \langle \star, \frac{1}{2}\rangle\} \\
\vec{\alpha}\vec{\Phi}^2\,\{\langle true, 1\rangle\}\,(pnat) & = & \{\langle e, \frac{1}{2}\rangle, \langle o, \frac{1}{4}\rangle, \langle \star, \frac{1}{4}\rangle\} \\
\vec{\alpha}\vec{\Phi}^3\,\{\langle true, 1\rangle\}\,(pnat) & = & \{\langle e, \frac{5}{8}\rangle, \langle o, \frac{1}{4}\rangle, \langle \star, \frac{1}{8}\rangle\} \\
\vec{\alpha}\vec{\Phi}^4\,\{\langle true, 1\rangle\}\,(pnat) & = & \{\langle e, \frac{5}{8}\rangle, \langle o, \frac{5}{16}\rangle, \langle \star, \frac{1}{16}\rangle\} \\
\ldots & \ldots & \ldots
\end{array}$$

Therefore, the corresponding fixpoints in the abstract semantics are for the operator $\tilde{\alpha}\tilde{\Phi}$:

$$\tilde{\alpha}(\lim \tilde{\Phi}) = \{e, o, \star\},$$

which basically tells us that this program computes even and odd numbers or eventually nothing at all (because it does not terminate).

For $\vec{\alpha}\vec{\Phi}$ the situation is quite different as we get as the limit distribution on $\mathcal{D}$:

$$\vec{\alpha}(\lim \vec{\Phi}) = \left\{\left\langle e, \frac{2}{3}\right\rangle, \left\langle o, \frac{1}{3}\right\rangle\right\}$$

which tells us not only that the probability of non-termination vanishes (in the long run) but also that even and odd numbers are by no means generated equally likely (basically because the even zero is the result of already half the runs).

The probabilities $\frac{2}{3}$ for e and $\frac{1}{3}$ for o are the infinite sums

$$\frac{1}{2} + \frac{1}{8} + \ldots = \sum_{k=0}^{\infty} \frac{1}{2^{2k+1}}, \text{ and}$$

$$\frac{1}{4} + \frac{1}{16} + \ldots = \sum_{k=1}^{\infty} \frac{1}{2^{2k}},$$

respectively. One can easily see that these sums are two geometric series with common ratio between 0 and 1 and therefore convergent. In particular, we have for e

$$\sum_{k=0}^{\infty} \frac{1}{2^{2k+1}} = \sum_{k=0}^{\infty} \frac{1}{2} \cdot \frac{1}{4^k},$$

that is the geometric series with constant $\frac{1}{2}$ and ratio $\frac{1}{4}$, while for o we have

$$\sum_{k=1}^{\infty} \frac{1}{2^{2k}} = (\sum_{k=0}^{\infty} \frac{1}{4^k}) - 1,$$

that is the geometric series with constant 1 and ratio $\frac{1}{4}$.

5.2 Numerical Experiments

In order to study the relation between concrete and abstract semantics in greater detail we simplify the above example by looking at only the first six iteration steps, i.e. we look at a finite $\mathcal{C}$. This will allow us to represent all the involved linear maps by means of finite matrices, thus making the calculations more feasible.

Let us assume that the elements in the reduced constraint system $\mathcal{C}$ are enumerated in the following way:

$$true, *, 0, 1, 2, 3, 4, 5, false$$

and that the elements in $\mathcal{D}$ occur in the sequence:

$$t, \star, e, o, f.$$

5.2.1 The Abstraction

We will consider the same extraction function α as in the above infinite example. Based on these base vectors for the finite dimensional vector spaces $\mathcal{V}(\mathcal{C})$ and $\mathcal{V}(\mathcal{D})$ we can represent the lifted mapping $\vec{\alpha}$ explicitely by a matrix A of the following form:

$$A = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

The Moore-Penrose pseudo-inverse $\Gamma = A^\dagger$ of A is then given by the following matrix:

$$A^\dagger = \Gamma = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

With these two matrices representing the two linear transformations $\vec{\alpha}$ and $\vec{\gamma}$ we can easily compute the abstraction or concretisation of vectors or distributions in $\mathcal{V}(\mathcal{C})$ and $\mathcal{V}(\mathcal{D})$, respectively — simply by multiplication. This way we get for example:

$$\vec{\alpha}(0,0,1,0,0,0,0,0,0) = (0,0,1,0,0)$$

reflecting the fact that concrete constraint 0 is mapped into the abstract constraint e. Similarly,

$$\vec{\alpha}(0,0,\frac{1}{2},0,\frac{1}{2},0,0,0,0) = (0,0,1,0,0)$$

reflecting that the set $\{0,2\}$, or to be precise the uniform distribution on the elements 0 and 2 in $\mathcal{C}$ is also mapped into e, and finally

$$\vec{\alpha}(0,0,0,1,1,0,0,0,0) = (0,0,\frac{1}{2},\frac{1}{2},0)$$

i.e. the set $\{1,2\}$ is mapped into a half e and half o. If we look at the pseudo-inverse, we get the following interesting transformations:

$$\vec{\gamma}(0,0,1,0,0) = (0,0,\frac{1}{3},0,\frac{1}{3},0,\frac{1}{3},0,0)$$

and

$$\vec{\gamma}(0,0,0,1,0) = (0,0,0,\frac{1}{3},0,\frac{1}{3},0,\frac{1}{3},0)$$

which is reflecting the simple fact that once we abstracted the concrete number to even and odd it is impossible to reverse this. The best we can do is to distribute evenly among all even, respectively odd, numbers.

5.2.2 *Concrete Semantics*

The matrix representation of a concrete fixpoint operator $\Phi : \mathcal{V}(\mathcal{C}) \mapsto \mathcal{V}(\mathcal{C})$ for the finite counting semantics is as follows:

$$\Phi = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

As expected, we get the following sequence of vectors starting with the initial distribution $\vec{i} = (1,0,0,0,0,0,0,0,0)$:

$$\begin{array}{rcl} \Phi^1(\vec{i}) & = & (0,\frac{1}{2},\frac{1}{2},0,0,0,0,0,0) \\ \Phi^2(\vec{i}) & = & (0,\frac{1}{4},\frac{1}{2},\frac{1}{4},0,0,0,0,0) \\ \Phi^3(\vec{i}) & = & (0,\frac{1}{8},\frac{1}{2},\frac{1}{4},\frac{1}{8},0,0,0,0) \\ \Phi^4(\vec{i}) & = & (0,\frac{1}{16},\frac{1}{2},\frac{1}{4},\frac{1}{8},\frac{1}{16},0,0,0) \\ \ldots & \ldots & \ldots \end{array}$$

5.2.3 *Abstract Semantics*

Using these matrix representations it is then easy to compute the (matrix representation of the) abstract fixpoint operator $\Psi : \mathcal{V}(\mathcal{D}) \mapsto \mathcal{V}(\mathcal{D})$ as:

$$\Psi = A \cdot \Phi \cdot \Gamma = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & \frac{5}{6} & \frac{1}{6} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

If this Ψ is applied to an initial abstract distribution $\vec{d} = (1,0,0,0,0)$ we get:

$$\begin{array}{rcl} \Psi^1(\vec{d}) & = & (0,\frac{1}{2},\frac{1}{2},0,0) \\ \Psi^2(\vec{d}) & = & (0,\frac{1}{4},\frac{1}{2},\frac{1}{4},0) \\ \Psi^3(\vec{d}) & = & (0,\frac{1}{8},\frac{5}{8},\frac{1}{4},0) \\ \Psi^4(\vec{d}) & = & (0,\frac{1}{16},\frac{5}{8},\frac{5}{16},0) \\ \ldots & \ldots & \ldots \end{array}$$

In the limit these two sequences (computing the concrete and the abstract semantics) get very close. More precisely we get:

$$\lim_{n\to\infty} \Phi^n(\vec{i}) = (0,0,\frac{1}{2},\frac{1}{4},\frac{1}{8},\frac{1}{16},\frac{1}{32},\frac{1}{32},0)$$

and therefore

$$\lim_{n\to\infty} \vec{\alpha}(\Phi^n(\vec{i})) = (0,0,\frac{21}{32},\frac{11}{32},0)$$

as compared to

$$\lim_{n\to\infty} \Psi^n(\vec{d}) = (0,0,\frac{20}{32},\frac{12}{32},0).$$

Note that the matrix $\Gamma{\cdot}A$ is the identity matrix. This implies that the probabilistic abstract interpretation is complete.

6. CONCLUSIONS

We presented a first outline of an abstract interpretation methodology which refers to a probabilistic semantics instead of a more standard order-theoretic one. Such methodology is thus immediately viable for analysing probabilistic programs. It can also be used for a probabilistic analysis of classical programs. The key element of the work presented is the definition of a "connection" between linear spaces: To express the relationship between the concrete and the abstract domains we use two linear maps, representing the abstraction function α, and the concretisation function γ, respectively. The latter is constructed as the Moore-Penrose pseudo-inverse of α. While the structural similarities between the two notions of Moore-Penrose pseudo-inverse and Galois connection ensures conservativity and safety features of a resulting probabilistic static analysis, the nature of linear structures additionally allows for a quantitative investigation of different abstractions with respect to their approximation error.

To compute a pseudo-inverse can be an expensive task. Fortunately, there exist several algorithms to effectively compute the pseudo-inverse of a matrix iteratively, e.g. by Grenville or Ben-Israel [7]. It might be also worth mentioning that computing the Moore-Penrose pseudo-inverse is at the heart of learning in neural networks [28]. For actually constructing pseudo-inverses, an alternative could be to develop methods for constructing the abstract semantics in a more direct way. This would mean to transfer some of the systematic design techniques used in the classical framework — like sequential composition, tensor product, etc. [32, 11] — into the vector space setting.

Our presentation was based on the (concurrent) constraint programming paradigm which allows for a nice and clean separation between qualitative and quantitative aspects of computation. Also we considered here finite structures stemming from the assumption that the constraint systems underlying the language were finite. It seems nevertheless possible to extend this approach to other programming paradigms as well as to infinite structures. In particular, the notion of a pseudo-inverse can be transferred to a setting involving infinite-dimensional Banach and Hilbert spaces [3].

Within the constraint programming framework it would be interesting to compare our results with the work on abstractions in the context of fuzzy or soft constraint systems [5]. Another direction of future research will be devoted to the development of efficient and/or incremental ways to compute the Moore-Penrose pseudo-inverse and methods of constructing an abstract probabilistic semantics directly, without explicitly computing the Moore-Penrose pseudo-inverse.

Finally, we plan to investigate a kind of hybrid approach in which the abstraction and concretisation functions relate domains which can be either cpo's or vector spaces. This would allow us for example to formulate the relationship between the semantics of CCP and PCCP — described in Section 2.3.4 — in terms of abstract interpretation, in line with the work on comparative semantics [12, 8].

7. REFERENCES

[1] S. Abramsky and C. Hankin, editors. *Abstract Interpretation of Declarative Languages.* Ellis-Horwood, Chichester, England, 1987.

[2] C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. Technical Report CSR-96-12, School of Computer Science, University of Birmingham, June 1996.

[3] F. J. Beutler. The operator theory of the pseudo-inverse. *Journal of Mathematical Analysis and Applications*, 10:451–470,471–493, 1965.

[4] P. Billingsley. *Probability and Measure.* Wiley & Sons, New York, 2nd edition, 1986.

[5] S. Bistarelli, P. Codognet, Y. Georget, and F. Rossi. Abstracting soft constraint. In K. Apt, T. Kakas, E. Monfroy, and F. Rossi, editors, *Proceedings of the ERCIM/COMPULOG Workshop on Constraints*, Pahos, Cyprus, 1999. University of Cyprus.

[6] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, 1997.

[7] S. L. Campbell and D. Meyer. *Generalized Inverse of Linear Transformations.* Constable and Company, London, 1979.

[8] P. Cousot. Constructive design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation. In S. Brooks and M. Mislove, editors, *13th International Symposium on Mathematical Foundations of Programming Semantics (MFPS97)*, volume 6 of *Electronic Notes in Theoretical Computer Science*, Amsterdam, 1997. Elsevier.

[9] P. Cousot and R. Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Symposium on Principles of Programming Languages (POPL)*, pages 238–252, Los Angeles, 1977.

[10] P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Symposium on Principles of Programming Languages (POPL)*, pages 269–282, San Antonio, Texas, 1979.

[11] P. Cousot and R. Cousot. Abstract Interpretation and Applications to Logic Programs. *Journal of Logic Programming*, 13(2-3):103–180, July 1992.

[12] P. Cousot and R. Cousot. Inductive Definitions, Semantics and Abstract Interpretation. In *19th ACM Symposium on Principles of Programming Languages (POPL'92)*, pages 83–94, 1992.

[13] D. Dams, R. Gerth, and O. Grumberg. Abstract Interpretations of Reactive Systems. *ACM Trans. Program. Lang. Syst.*, 19(2):253–291, 1997.

[14] 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.

[15] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Approximating continuous markov processes. In *LICS2000*, 2000.

[16] 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.

[17] A. Di Pierro 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*, volume 1450 of *Lecture Notes in Computer Science*, pages 446–455, Berlin – New York, August 1998. Springer Verlag.

[18] A. Di Pierro and H. Wiklicky. On the Precision of Abstract Interpretations. In *Tenth International Workshop on Logic-based Program Synthesis and Transformation (LOPSTR 2000)*, pages 36–44, London, UK, 2000.

[19] A. Di Pierro and H. Wiklicky. Quantitative observables and averages in Probabilistic Constraint Programming. In K. Apt, A. Kakas, E. Monfroy, and F. Rossi, editors, *New Trends in Constraints*, volume 1865 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 2000.

[20] R. Engelking. *General Topology*, volume 6 of *Sigma Series in Pure Mathematics*. Heldermann Verlag, Berlin, 1989.

[21] M. Falaschi, M. Gabrielli, K. Marriott, and C. Palamidessi. Compositional analysis for concurrent constraint programming. In *IEEE Symposium on Logic in Computer Science, (LICS)*, pages 210–221, 1993.

[22] R. Giacobazzi, F. Ranzato, and F. Scozzari. Making Abstract Interpretations Complete. *Journal of the ACM*, 47(2):361–416, 2000.

[23] V. Gupta, R. Jagadeesan, and P. Panangaden. Stochastic programs as concurrent constraint programs. In *Symposium on Principles of Programming Languages (POPL)*. ACM, 1999.

[24] R. Jagadeesan, V. Saraswat, and V. Shanbhogue. Angelic non-determinism in concurrent constraint programming. Technical report, Xerox Park, 1991.

[25] C. Jones. *Probabilistic Non-Determinism*. PhD thesis, University of Edinburgh, Edingburgh, 1993.

[26] C. Jones and G. Plotkin. A probabilistic powerdomain of evaluations. In *Symposium on Logic in Computer Science (LICS)*, pages 186–195. IEEE Computer Society Press, 1989.

[27] A. Jung and R. Tix. The troublesome probabilistic powerdomain. In A. Edalat, A. Jung, K. Keimel, and M. Kwiatkowska, editors, *Third Workshop on Computation and Approximation*, volume 13 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers B.V., 1998. 23 pages.

[28] T. Kohonen. *Self-Organization and Associative Memory*. Springer-Verlag, Berlin, 3 edition, 1989.

[29] D. Kozen. Semantics for probabilistic programs. *Journal of Computer and System Sciences*, 22:328–350, 1981.

[30] M. Kwiatkowska. Infinite Behaviour and Fairness in Concurrent Constraint Programming. In J. W. de Bakker, W. P. Roever, and G. Rozenberg, editors, *Semantics: Foundations and Applications*, volume 666 of *Lecture Notes in Computer Science*, pages 348–383, Beekbergen The Nederland, June 1992. REX Workshop, Springer Verlag.

[31] C. Morgan, A. McIver, K. Seidel, and J. Sanders. Probabilistic predicate transformers. Technical Report PRG-TR-4-95, Programming Research Group, Oxford University Computing Laboratory, 1995.

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

[33] V. V. Prasolov. *Problems and Theorems in Linear Algebra*, volume 134 of *Translation of Mathematical Monographs*. American Mathematical Society, Providence, Rhode Island, 1994.

[34] N. Saheb-Djahromi. CPO's of measures for nondeterminism. *Theoretical Computer Science*, 12:19–37, 1980.

[35] V. 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.