Relational Analysis and Precision via Probabilistic Abstract Interpretation

Alessandra Di Pierro[1] Pascal Sotin[2] Herbert Wiklicky[3]

Abstract

Within the context of a quantitative generalisation of the well established framework of Abstract Interpretation – i.e. Probabilistic Abstract Interpretation – we investigate a quantitative notion of precision which allows us to compare analyses on the basis of their expected exactness for a given program. We illustrate this approach by considering various types of numerical abstractions of the values of variables for independent analysis as well as weakly and fully relational analysis. We utilise for this a linear operator semantics of a simple imperative programming language. In this setting, fully relational dependencies are realised via the tensor product. Independent analyses and weakly relational analyses are realised as abstractions of the fully relational analysis.

Keywords: Probabilistic Semantics, Linear Operators, Probabilistic Abstract Interpretation, Relational Analysis

1 Introduction

Static program analysis aims in providing safe approximations of various program properties. In Abstract Interpretation these properties form a lattice of domains that model possible (abstract) values. An analysis based on a certain abstract domain can be rather imprecise. One of the reasons for this imprecision is the possible obfuscation of the relationship between variables. Additionally, the representation of a set of (concrete) values by (abstract) elements makes it impossible to discover, for example, "forbidden sections" within an interval. In this paper we investigate a framework for measuring the imprecision of a domain (or analysis) and we concentrate in particular on the issue of how to analyse *relational* dependency in this quantitative setting. We illustrate our approach by considering (weakly) relational analyses which preserve partially the relation between variables.

Probabilistic Abstract Interpretation (PAI) [9] – a quantitative generalisation of classical Abstract Interpretation [5] – provides the context in which we can introduce a quantitative notion of precision and use it to compare any two analyses. We utilise

[1] Email: alessandra.dipierro@univr.it

[2] Email: pascal.sotin@irisa.fr

[3] Email: herbert@doc.ic.ac.uk

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

a Linear Operator Semantics (LOS) of a simple imperative programming language where relational dependencies are realised via tensor products.

In order to compare the results of various analyses we introduce a new measure for the precision of a given analysis. This is based on a quantitative version of the well-known lattice theoretic comparison of Abstract Interpretations cf [18]. The identification of each probabilistic abstraction with a corresponding orthogonal projection operator allows us to exploit the Birkhoff-von Neumann lattice structure on these operators [2] and its ordering to compare PAI's. The fact that PAI's are represented by linear operators also allows us to compare the difference between any two of them in metric terms, even when their classical counterparts are not comparable (i.e. one is not a refinement of the other).

2 Language and Semantics

We consider the probabilistic language **pWhile** presented in [7]. The usual deterministic **While** language is a proper sub-language of **pWhile** without the probabilistic choice statement. Even when the intention is to analyse only deterministic, i.e. **While**, programs, it is necessary to consider probabilities for the "abstracted" versions of such programs. One can intuitively see that a statement like **if** $even(\mathtt{x})$ **then** S_1 **else** S_2 **fi** corresponds to an abstract version where S_1 and S_2 are executed with a half probability each, i.e. to the statement **choose** $0.5 : S_1$ **or** $0.5 : S_2$ (provided we assume that the value of `x` is uniformly distributed). The situation in classical analysis is similar, although there deterministic programs are usually "abstracted" to non-deterministic ones (operating with sets of possible results).

2.1 Syntax.

A program P in **pWhile** is made up from a possibly empty list of variable declarations D followed by a statement S. Formally, $P ::= D;\ S \mid S$ with $D ::= d;\ D \mid d$. A declaration d fixes the types of the program's variables, i.e. **int** or **bool**. The syntax of statements S is as follows:

$$\begin{aligned} S \ ::= \ & \textbf{skip} \mid \textbf{stop} \mid \mathtt{x} \leftarrow e \mid S_1;\ S_2 \mid \textbf{choose}\ p_1 : S_1\ \textbf{or}\ p_2 : S_2 \\ & \mid \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi} \mid \textbf{while}\ b\ \textbf{do}\ S\ \textbf{od} \end{aligned}$$

In the **choose** statement we allow only for constant probabilities p_i and assume w.l.o.g. that they are normalised, i.e. add up to 1. Arithmetic expressions are of the usual form

$$a \ ::= n \mid \mathtt{x} \mid a_1 \odot a_2$$

with $n \in \mathbb{Z}$, `x` a program variable and '$\odot$' representing one of the usual arithmetic operations '$+$', '$-$', or '$\times$'. Boolean expressions are also defined as one would expect by

$$b \ ::= \textsc{true} \mid \textsc{false} \mid \mathtt{x} \mid \neg b \mid b_1 \vee b_2 \mid b_1 \wedge b_2 \mid a_1 \lessgtr a_2,$$

where `x` is a Boolean variable and '$\lessgtr$' denotes one of the standard comparison operators for arithmetic expressions, i.e. $<, \leq, =, \neq, \geq, >$.

R0	$\langle \mathbf{skip}, \sigma\rangle \longrightarrow_1 \langle \mathbf{stop}, \sigma\rangle$	
R1	$\langle \mathbf{stop}, \sigma\rangle \longrightarrow_1 \langle \mathbf{stop}, \sigma\rangle$	
R2	$\langle \mathtt{x} \leftarrow e, \sigma\rangle \longrightarrow_1 \langle \mathbf{stop}, \sigma[\mathtt{x} \mapsto \mathcal{E}(e)\sigma]\rangle$	
R3$_1$	$\dfrac{\langle S_1, \sigma\rangle \longrightarrow_p \langle S_1', \sigma'\rangle}{\langle S_1; S_2, \sigma\rangle \longrightarrow_p \langle S_1'; S_2, \sigma'\rangle}$	
R3$_2$	$\dfrac{\langle S_1, \sigma\rangle \longrightarrow_p \langle \mathbf{stop}, \sigma'\rangle}{\langle S_1; S_2, \sigma\rangle \longrightarrow_p \langle S_2, \sigma'\rangle}$	
R4$_1$	$\langle \mathbf{choose}\ p_1 : S_1\ \mathbf{or}\ p_2 : S_2, \sigma\rangle \longrightarrow_{p_1} \langle S_1, \sigma\rangle$	
R4$_2$	$\langle \mathbf{choose}\ p_1 : S_1\ \mathbf{or}\ p_2 : S_2, \sigma\rangle \longrightarrow_{p_2} \langle S_2, \sigma\rangle$	
R5$_1$	$\langle \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}, \sigma\rangle \longrightarrow_1 \langle S_1, \sigma\rangle$	if $\mathcal{E}(b)\sigma = \text{TRUE}$
R5$_2$	$\langle \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}, \sigma\rangle \longrightarrow_1 \langle S_2, \sigma\rangle$	if $\mathcal{E}(b)\sigma = \text{FALSE}$
R6$_1$	$\langle \mathbf{while}\ b\ \mathbf{do}\ S\ \mathbf{od}, \sigma\rangle \longrightarrow_1 \langle S;\ \mathbf{while}\ b\ \mathbf{do}\ S\ \mathbf{od}, \sigma\rangle$	if $\mathcal{E}(b)\sigma = \text{TRUE}$
R6$_2$	$\langle \mathbf{while}\ b\ \mathbf{do}\ S\ \mathbf{od}, \sigma\rangle \longrightarrow_1 \langle \mathbf{stop}, \sigma\rangle$	if $\mathcal{E}(b)\sigma = \text{FALSE}$

Table 1
A Probabilistic Transition System for **pWhile**

2.2 Linear Operator Semantics.

The Structural Operational Semantics (SOS) of **pWhile** can, as usual, be given via a transition relation which is in this case a probabilistic one where each transition has a probability associated with it. The configurations are – as in the deterministic case – pairs $\langle S, \sigma\rangle$ with S a statement and σ a state, i.e. a map which associates to each variable $\mathtt{x}$ a (unique) value in the corresponding sets $\mathbb{Z}$ and $\mathbb{B}$ of integers and Booleans. We denote by **Var** the set of all variables, by **State** = **Var** $\rightarrow \mathbb{Z} \cup \mathbb{B}$ the set of states, and by **Conf** the set of all configurations. The transition relation, $\longrightarrow_p \subseteq$ **Conf** $\times [0,1] \times$ **Conf**, for **pWhile** is defined in [8] and we recall it here in Table 1. The function $\mathcal{E}(\cdot)\cdot$ represents the usual semantics of expressions e in state σ, see e.g. [16].

While the operational semantics of programs is usually defined in terms of transition relations many other areas specify the “dynamics” of a system in terms of other mathematical objects. In the theory of stochastic processes one usually considers matrices, or linear operators, to present a particular system. For Discrete Time Markov Chains (DTMC), cf. e.g. [11], it is, for example, sufficient to specify the transition probabilities between two distinct “states” s_1 and s_2 (these correspond in our case to configurations and not to states in our sense). All these probabilities p_{ij} from s_i to s_j can be written down as the (generator) matrix $\mathbf{P} = (p_{ij})_{ij}$ of the DTMC we want to describe. This matrix $\mathbf{P}$ is always a *stochastic* matrix, i.e. a matrix where all row entries sum up to one.

The SOS transition relation for **pWhile** – as well as its restriction to the reachable configurations of a given program P, which gives a representation of the SOS semantics of a particular program – can also be encoded as a matrix or linear operator (cf. [6]), i.e. the matrix $\mathbf{T}$ (or $\mathbf{T}(P)$) indexed by configurations $c_i = \langle S_i, \sigma_i \rangle, c_j = \langle S_j, \sigma_j \rangle \in \mathbf{Conf}$ defined by $(\mathbf{T})_{c_i,c_j} = p$ if $\langle S_i, \sigma_i \rangle \longrightarrow_p \langle S_j, \sigma_j \rangle$ and 0 otherwise. This representation is based on the definition of some enumeration of all (reachable) configurations.

For programs which visit only finitely many configurations, e.g. programs which utilise only a finite subset $\underline{\mathbb{Z}}$ of $\mathbb{Z}$, this will result in a finite-dimensional matrix. In general, for infinite configuration sets, we obtain this way linear operators on the Banach space $\ell_1(\mathbf{Conf})$. We refer to the matrix representation of the semantics of a program as the program Linear Operator Semantics (LOS). It contains exactly the same information as the SOS semantics.

Note that for the rules for **pWhile** in Table 1 the operator $\mathbf{T}$ obtained this way will always be row-normalised, i.e. stochastic, as the transition probabilities for all configurations add up to one: For the deterministic rules there is simply only one possible transition with probability 1 and for the **choose** statement we have assumed that p_1 and p_2 are normalised (if this is not the case, we could normalise them before execution as long as they are constants).

If we start in a configuration $c = \langle S, \sigma \rangle$ we can represent this initial situation by a (row) vector $\vec{c} = (0, \ldots, 0, 1, 0 \ldots)$ with only one non-zero entry for the coordinate corresponding to c. If we compute the product $\vec{c}\mathbf{T}$ we get a vector where each non-zero entry $(\vec{c}\mathbf{T})_i$ corresponds to a configuration c_i reachable from c; in fact, $(\vec{c}\mathbf{T})_i$ specifies the probabilitiy p_i of making a transition $c \longrightarrow_{p_i} c_i$. One can iterate this and obtain the probability of reaching c_j in n steps from c as $(\vec{c}\mathbf{T}^n)_j$.

The current computational state $\vec{\sigma}$, i.e. the possible values of all variables, together with the remaining program S after n steps is given by one of several possible configurations, each of which is reached with a certain probability (depending on the initial configuration c and the semantics of the program P we consider). We can represent this knowledge simply by the vector $\vec{c}\mathbf{T}^n(P)$. Each of these vectors has entries which sum up to one (as $\mathbf{T}(P)$ is stochastic). In other words we obtain after a number of iterations a *distribution* on $\mathbf{Conf} = \mathbf{Stmt} \times \mathbf{State}$ (where $\mathbf{Stmt}$ is the set of all statements in **pWhile**).

It is well known from the theory of probability, e.g. [11], that distributions (and measures) on $X \times Y$ are represented by elements in the *tensor product* of distributions over X and over Y. As distributions (or measures) on X are sub-sets of the vector space $\mathcal{V}(X) = \{\langle x_s, s \rangle_{s \in X} \mid x_s \in \mathbb{R}\} = \{(x_s)_{s \in X}\}$, we have:

$$\mathbf{Dist}(\mathbf{Conf}) \subseteq \mathcal{V}(\mathbf{Conf}) = \mathcal{V}(\mathbf{Stmt} \times \mathbf{State}) = \mathcal{V}(\mathbf{Stmt}) \otimes \mathcal{V}(\mathbf{State}).$$

We also observe that a function, i.e. a state σ in $\mathbf{Var} \rightarrow \mathbf{Value}$, can be seen as an element in $\mathbf{Value}^{|\mathbf{Var}|} = \mathbf{Value} \times \ldots \times \mathbf{Value}$ (for every variable we just specify its value in $\mathbf{Value} = \mathbb{Z} \cup \mathbb{B}$). Thus, distributions over $\mathbf{State}$ can be represented too by a tensor product and $\mathbf{T}$ is a (stochastic) linear operator on

$$\mathcal{V}(\mathbf{Conf}) = \mathcal{V}(\mathbf{Stmt}) \otimes \mathcal{V}(\mathbf{Value}^{|\mathbf{Var}|}) = \mathcal{V}(\mathbf{Stmt}) \otimes \mathcal{V}(\mathbf{Value})^{\otimes|\mathbf{Var}|}.$$

It is therefore always possible to decompose $\mathbf{T}$ into a linear combination of factors $\mathbf{T} = \sum_k \mathbf{T}_k$ where each factor $\mathbf{T}_k$ is a tensor product expressing the transfer of control between two statements and of the operations on single variables, i.e.

$$\mathbf{T}_k = \mathbf{S} \otimes \mathbf{T}_1 \otimes \ldots \otimes \mathbf{T}_n,$$

$\mathbf{T}(\ell_1,\ell_2)$	$= \mathbf{E}(\ell_1,\ell_2)\otimes\mathbf{I}$	for $[\mathbf{skip}]^{\ell_1}$
$\mathbf{T}(\ell_1,\ell_2)$	$= \mathbf{E}(\ell_1,\ell_2)\otimes\mathbf{U}(\mathtt{x}\leftarrow e)$	for $[\mathtt{x}\leftarrow e]^{\ell_1}$
$\mathbf{T}(\ell,\underline{\ell_t})$	$= \mathbf{E}(\ell,\ell_t)\otimes\mathbf{P}(b=\textsc{true})$	for $[b]^{\ell}$
$\mathbf{T}(\ell,\ell_f)$	$= \mathbf{E}(\ell,\ell_f)\otimes\mathbf{P}(b=\textsc{false})$	for $[b]^{\ell}$
$\mathbf{T}(\ell,\ell_k)$	$= \mathbf{E}(\ell,\ell_k)\otimes\mathbf{I}$	for $[\mathbf{choose}]^{\ell}$
$\mathbf{T}(\ell,\ell)$	$= \mathbf{E}(\ell,\ell)\otimes\mathbf{I}$	for $[\mathbf{stop}]^{\ell}$

Table 2
Linear Operator Semantics for **pWhile**

where $\mathbf{S}$ expresses the control flow between statements and $\mathbf{T}_1,\dots,\mathbf{T}_n$ define the update of variables $\mathtt{x}_1,\dots,\mathtt{x}_n$.

In [8] we presented a syntax-directed construction of the LOS semantics of **pWhile** which avoids translating (infinite) SOS transition relations into matrices but instead constructs directly a tensor product representation of $\mathbf{T}(S)$. For this we need to consider a labelled version of the **pWhile** syntax: S ::= $[\mathbf{skip}]^\ell$ $\mid$ $[\mathbf{stop}]^\ell$ $\mid$ $[p\leftarrow e]^\ell$ $\mid$ $S_1;\ S_2$ $\mid$ $[\mathbf{choose}]^\ell\ p_1 : S_1$ **or** $p_2 : S_2$ $\mid$ **if** $[b]^\ell$ **then** S_1 **else** S_2 **fi** $\mid$ **while** $[b]^\ell$ **do** S **od**, with labels $\ell \in \mathbf{Lab}(S)$.

For a given program S we then determine (statically) its probabilistic *control flow*, $\mathcal{F}(S)$, i.e. a set of triples $\langle \ell_i, p_{ij}, \ell_j\rangle$ which describe a possible transfer of control from label ℓ_i to label ℓ_j with probability p_{ij} (where $p_{ij} = 1$ for all deterministic statements, i.e. except for a **choose** statement). For tests $[b]^\ell$ (in **if** and **while** constructs) we distinguish between the control transfer in case the test succeeds or fails by underlining the target label for success.

The Linear Operator Semantics of a **pWhile** program S is then defined as the operator $\mathbf{T} = \mathbf{T}(S)$ on $\mathcal{V}(\mathbf{Lab}(S)\times\mathbf{State})$. It is given by a linear combination of local updates $\mathbf{T}(\ell_i,\ell_j)$:

$$\mathbf{T}(S) = \sum_{\langle i,p_{ij},j\rangle\in\mathcal{F}(S)} p_{ij}\mathbf{T}(\ell_i,\ell_j).$$

The aim of $\mathbf{T}(S)$ is to *collect* for every triple in the probabilistic flow $\mathcal{F}(S)$ of S its effects, weighted according the probability associated to this triple. The operators $\mathbf{T}(\ell_i,\ell_j)$ which implement the local state updates and control transfers from ℓ_i to ℓ_j are presented in Table 2.

Each local operator $\mathbf{T}(\ell_i,\ell_j)$ is of the form $\mathbf{E}(\ell_i,\ell_j)\otimes\mathbf{N}$ where the first factor $\mathbf{E}(\ell_i,\ell_j)$ realises the transfer of control from label ℓ_i to label ℓ_j while the second factor $\mathbf{N}$ represents a state update or – in the case of tests – a filter operator.

The *matrix units* $\mathbf{E}(m,n)$ contain only one non-zero entry, namely $(\mathbf{E}(m,n))_{mn} = 1$, and $\mathbf{I}$ is the *identity* operator. Using these basic building blocks we can define a number of "filters" $\mathbf{P}$ as depicted in Table 3. The operator $\mathbf{P}(c)$ has only one non-zero entry: the diagonal element $\mathbf{P}_{cc} = 1$, i.e. $\mathbf{P}(c) = \mathbf{E}(c,c)$. This operator extracts the probability corresponding to the c-th coordinate of a vector, i.e. for $\vec{x} = (x_i)_i$ the multiplication with $\mathbf{P}(c)$ results in a vector $\vec{x'} = \vec{x}\mathbf{P}(c)$ with only one non-zero coordinate, namely $x'_c = x_c$. The operator $\mathbf{P}(\sigma)$ performs a

$$
\begin{array}{rclrcl}
(\mathbf{P}(c))_{ij} & = & \begin{cases} 1 \text{ if } i = c = j \\ 0 \text{ otherwise.} \end{cases} & (\mathbf{U}(c))_{ij} & = & \begin{cases} 1 \text{ if } j = c \\ 0 \text{ otherwise.} \end{cases} \\
\mathbf{P}(\sigma) & = & \bigotimes_{i=1}^{v} \mathbf{P}(\sigma(\mathtt{x}_i)) & \mathbf{U}(\mathtt{x}_k \leftarrow c) & = & \bigotimes_{i=1}^{k-1} \mathbf{I} \otimes \mathbf{U}(c) \otimes \bigotimes_{i=k+1}^{v} \mathbf{I} \\
\mathbf{P}(e = c) & = & \sum_{\mathcal{E}(e)\sigma = c} \mathbf{P}(\sigma) & \mathbf{U}(\mathtt{x}_k \leftarrow e) & = & \sum_{c} \mathbf{P}(e = c)\mathbf{U}(\mathtt{x}_k \leftarrow c)
\end{array}
$$

Table 3
Test and Update Operators for **pWhile**

similar test for a vector representing the probabilistic state of the computation. It filters the probability that the computation is in a classical state σ. This is achieved by checking whether each variable $\mathtt{x}_i$ has the value specified by σ namely $\sigma(\mathtt{x}_i)$. Finally, the operator $\mathbf{P}(e = c)$ filters those states where the values of the variables $\mathtt{x}_i$ are such that the evaluation of the expression e results in c. The number of (diagonal) non-zero entries of this operator is exactly the number of states σ for which $\mathcal{E}(e)\sigma = c$.

The update operators $\mathbf{U}$ implement the actual state changes. From an initial probabilistic state $\vec{\sigma}$ – i.e. a distribution over classical states – we get a new probabilistic state $\vec{\sigma}'$ via $\vec{\sigma}\mathbf{U}$. The simple operator $\mathbf{U}(c)$ implements the deterministic update of a variable $\mathtt{x}_i$ to the constant c: Whatever the value(s) of $\mathtt{x}_i$ are, after applying $\mathbf{U}(c)$ to the state vector describing $\mathtt{x}_i$, we get a point distribution expressing the fact that the value of $\mathtt{x}_i$ is now certainly c. The operator $\mathbf{U}(\mathtt{x}_k \leftarrow c)$ puts $\mathbf{U}(c)$ into the context of other variables: Most factors in the tensor product are identities, i.e. most variables keep their previous values, only $\mathtt{x}_k$ is deterministically updated to its new constant value c using the previously defined $\mathbf{U}(c)$ operator. The operator $\mathbf{U}(\mathtt{x}_k \leftarrow e)$ updates a variable not to a constant but to the value of an expression e. This update is realised using the filter operator $\mathbf{P}(e = c)$: For all possible values c of e we select those states where e evaluates to c and then update $\mathtt{x}_k$ to this c.

For the **skip** and **stop** no changes to the state happen, we only transfer control (deterministically) to the next statement or loop on the current (terminal) statement using matrix units $\mathbf{E}$. Also in the case of a **choose** there is no change to the state but only a transfer of control, however the probabilities p_{ij} will in general be different from 1, unlike **skip**. With assignments we have both a state update, implemented using $\mathbf{U}(p \leftarrow e)$ as well as a control flow step. For tests b we use a filter operator $\mathbf{P}(b = \text{TRUE})$ to select those states which pass the test or $\mathbf{P}(b = \text{FALSE})$ fail it to determine to which label control will pass.

Note that $\mathbf{P}(b = \text{TRUE}) + \mathbf{P}(b = \text{FALSE}) = \mathbf{I}$, i.e. at any test b every state will cause exactly one (unambiguous) control transfer. We allow in **pWhile** only for constant probabilities p_i in the **choose** construct, which sum up to 1 and as with classical **While** we have no “blocked” configurations (even the terminal **stop** statements ‘loop’).

2.3 Some Notations.

In order to simplify the presentation, we will assume in this paper that each variable can only take values in a finite set. Thus the LOS of the programs we consider – even if they are non-terminating – is presented by finite dimensional matrices. It is possible to generalise our setting to infinite dimensional vector spaces, in particular if we base the LOS on a *Hilbert space* structure[4] (for details see standard references like [4,13] etc.). Finite dimensional real vector spaces are always also Hilbert spaces.

Linear operators, i.e. linear maps on a vector space, can be represented as matrices. This provides a useful notation for expressing and combining linear operators. We use row vectors, so $\mathbf{F}(x) = \vec{x}\mathbf{F}$ and $\mathbf{F} \circ \mathbf{G} = \mathbf{GF}$, $\vec{x}$ being a row vector, $\mathbf{F}$ and $\mathbf{G}$ being both linear operators and their associated matrices. We usually simplify the notation of vectors by $\vec{x} = x$. We denote the identity matrix by $\mathbf{I}$ and, if necessary, we indicate its dimension by a sub-script index, i.e. $\mathbf{I}_n$ is the $n \times n$ diagonal matrix $\mathbf{I}_n = \mathrm{diag}(1, 1, \dots, 1)$. Any Hilbert space comes with a norm defined by the inner product: $\|x\|^2 = \langle x, x\rangle$. This norm is extended to the linear operators via the operator norm defined by $\|\mathbf{F}\| = \sup_x \|x\mathbf{F}\|$. This induced norm is said to be sub-multiplicative because by definition, we have $\|x\mathbf{F}\| \le \|x\| \cdot \|\mathbf{F}\|$ and $\|\mathbf{GF}\| \le \|\mathbf{G}\| \cdot \|\mathbf{F}\|$. We denote by $.^t$ the transposition of matrices and vectors, i.e. $((a^t)_{ij}) = (a_{ji})$. In order to save space, we will sometimes use the transposed form of matrices and vectors.

3 Relational Static Analysis

Muchnick and Jones describe in [12] as an *independent attribute method* a flow analysis where a state at each control point is the conjunction of possible attributes for each variable, and as *relational method* a flow analysis where a state at a control point is the possible conjunction of attributes for each variable.

As an example, consider the following small program which assignes the value of y to x which both can take values in the set $\{1, 2\}$:

```
var x,y:[1,2] begin x:=y; stop
```

We could obtain here two analyses: (i) $\mathtt{x} \in \{1, 2\}$ and $\mathtt{y} \in \{1, 2\}$ and (ii) $\mathtt{x} = \mathtt{y}$, which describe the state at the end of the program. We refer to analysis (i) as *non-relational*, because it forgets the relation $\mathtt{x} = \mathtt{y}$, so it allows unreachable states like $\mathtt{x} = 1, \mathtt{y} = 2$. Analysis (ii) is *relational*, because it can remember information linking variables together. For this program, the equality relation is recorded, but as values have been forgotten, the result also contains unreachable states like $\mathtt{x} = 3, \mathtt{y} = 3$.

One can also think of *weakly relational* analyses where some information about the relationship between variables is preserved. Well known examples of such weakly relational analyses are *zone* and *octagon* abstraction [14].

Note that there is no direct way to compare the analyses (i) and (ii) above in the classical framework. To overcome this we will present a framework which allows us to measure the precision of abstractions quantitatively. As in the classical case (cf e.g. [16, 4.4.2]), we will see that relationality and tensor product are closely related.

[4] A linear space is a Hilbert space if it has a scalar (or inner) product $\langle ., .\rangle$ and it is topologically complete wrt the norm generated by the scalar product.

Additionally, in our setting we can even express the "strength" of dependencies, i.e. the correlation between (abstract) properties. For our quantitative analysis we will use the PAI framework which we will recall briefly in the next section.

3.1 Probabilistic Abstract Interpretation

We abstract the Linear Operator Semantics in the framework of Probabilistic Abstract Interpretation [10,9]. The basic idea behind Abstract Interpretation is to analyse a program by looking at a simplified or abstract semantics which only registers aspects of the program that are relevant to the specific analysis. Typically, these aspects are encoded in the definition of an abstract domain which is usually structured, like the concrete domain, as a complete partial order. In the standard Abstract Interpretation theory by Cousot and Cousot the translation between the concrete and the abstract semantics is achieved via Galois connections (i.e. pairs of adjoint maps) which guarantee the correctness of the abstraction [5]. In our setting where we deal with linear operators defined on vector spaces, the relation between the concrete and the abstract semantics is formalised via the notion of a linear generalised inverse which can be seen as a linear analogon of a Galois connection [10]. This is the Moore-Penrose pseudo-inverse which is defined below. We refer in the following definitions to the general notion of Hilbert spaces as our (concrete and abstract) probabilistic domains, although as mentioned in Section 2, in this paper we restrict ourself mainly to the consideration of finite-dimensional vector spaces.

Definition 3.1 Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be two Hilbert spaces and let $\mathbf{A} : \mathcal{H}_1 \mapsto \mathcal{H}_2$ be a bounded linear map between them. A bounded linear map $\mathbf{A}^\dagger = \mathbf{G} : \mathcal{H}_2 \mapsto \mathcal{H}_1$ is the *Moore-Penrose pseudo-inverse* of $\mathbf{A}$ iff

$$\mathbf{A} \circ \mathbf{G} = \mathbf{P_A} \text{ and } \mathbf{G} \circ \mathbf{A} = \mathbf{P_G},$$

where $\mathbf{P_A}$ and $\mathbf{P_G}$ denote orthogonal projections onto the ranges of $\mathbf{A}$ and $\mathbf{G}$.

A linear operator $\mathbf{P}$ on a Hilbert space $\mathcal{H}$ is called an *orthogonal projection* (operator) if we have $\mathbf{PP} = \mathbf{P}^2 = \mathbf{P}^*$ – more about this notion in Section 4.1. We define the *image* or *range*, $Im(\mathbf{A})$, of a linear operator $\mathbf{A}$ as the set $\{\mathbf{A}x \mid x \in \mathcal{H}\}$, and its *kernel* by $Ker(\mathbf{A}) = \{x \in \mathcal{H} \mid \mathbf{A}(x) = o\}$, with o the null vector in $\mathcal{H}$, i.e. the neutral element with respect to vector addition.

If $\mathcal{C}$ an $\mathcal{D}$ are two Hilbert spaces, and $\mathbf{A} : \mathcal{C} \rightarrow \mathcal{D}$ and $\mathbf{G} : \mathcal{D} \rightarrow \mathcal{C}$ are bounded linear operators between (the concrete domain) $\mathcal{C}$ and (the abstract domain) $\mathcal{D}$, such that $\mathbf{G}$ is the Moore-Penrose pseudo-inverse of $\mathbf{A}$, then we say that $(\mathcal{C}, \mathbf{A}, \mathcal{D}, \mathbf{G})$ forms a probabilistic abstract interpretation.

The Moore-Penrose pseudo-inverse, if it exists, is always unique. A necessary and sufficient condition for the existence of the Moore-Penrose pseudo-inverse for a bounded linear operator $\mathbf{A}$ on a Hilbert space $\mathcal{H}$ is that $\mathbf{A}$ is *normally solvable*, i.e. its range $Im(\mathbf{A})$ is closed [3, Thm 4.24]. All operators on a finite dimensional Hilbert space are Moore-Penrose invertible. The properties of the Moore-Penrose pseudo-inverse (cf. e.g. [1]) guarantee a form of optimality of the abstractions constructed via PAI; in fact, they are the *closest* to the concrete semantics one can construct, where closeness is defined via the distance induced by the norm on the Hilbert space. As this is a numerical quantity, we can get an estimate of the information

x \ y	1	2	x
1	0.7	0	0.7
2	0	0.3	0.3
y	0.7	0.3	

(a)

x \ y	1	2	x
1	0.5	0.2	0.7
2	0.2	0.1	0.3
y	0.7	0.3	

(b)

Fig. 1. Probability Distributions

lost in the abstraction [10].

3.2 Relational Abstraction: Tensor Product

In the following, we consider programs with l labels, v variables, and n possible values for each variable. This means that the first factor in each $\mathbf{T}_k$ is a $l \times l$ matrix (specifying the control steps within the program) and that each $\mathbf{T}_k$ is a tensor product of $v+1$ factors (the control flow part and one factor for each variable), and finally, each of the factors corresponding to the variables is an $n \times n$ matrix. The three abstraction operators we define below are all of the form $\mathbf{A}(\mathbf{S}) = \mathbf{I}_l \otimes \mathbf{S}$, where the control flow part of the LOS is left unchanged (the first factor is the identity) and $\mathbf{S}$ is a matrix describing the individual abstraction we have in mind.

Given a concrete semantics $\mathbf{T}$ and an abstraction operator $\mathbf{A}$, we can construct the abstracted version in PAI as $\mathbf{T}^{\#} = \mathbf{A}^{\dagger}\mathbf{T}\mathbf{A}$. As our LOS is constructed using tensor products it is important that the Moore-Penrose pseudo-inverse of a tensor product can easily be computed as follows [1, 2.1,Ex 3]:

$$(\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \ldots \otimes \mathbf{A}_n)^{\dagger} = \mathbf{A}_1^{\dagger} \otimes \mathbf{A}_2^{\dagger} \otimes \ldots \otimes \mathbf{A}_n^{\dagger}.$$

The abstraction of a semantics of the form $\mathbf{T} = \sum_k \mathbf{T}_k$ is simply

$$\mathbf{T}^{\#} = \mathbf{A}^{\dagger}\mathbf{T}\mathbf{A} = \sum_k \mathbf{A}^{\dagger}\mathbf{T}_k\mathbf{A} = \sum_k \mathbf{T}_k^{\#}.$$

The tensor product factor in each $\mathbf{T}_k = \mathbf{T}_{1k} \otimes \ldots \otimes \mathbf{T}_{nk}$ can be treated separately if the abstraction is of the above form: $\mathbf{T}_k^{\#} = \mathbf{A}^{\dagger}\mathbf{T}_k\mathbf{A} = (\mathbf{A}_1 \otimes \ldots \otimes \mathbf{A}_n)^{\dagger}(\mathbf{T}_{1k} \otimes \ldots \otimes \mathbf{T}_{nk})(\mathbf{A}_1 \otimes \ldots \otimes \mathbf{A}_n) = (\mathbf{A}_1^{\dagger}\mathbf{T}_{1k}\mathbf{A}_1) \otimes \ldots \otimes (\mathbf{A}_n^{\dagger}\mathbf{T}_{nk}\mathbf{A}_n)$.

An abstraction of the form $\mathbf{S} = \bigotimes_{i=1}^{v} \mathbf{A}_i$ (with non-trivial $\mathbf{A}_i$) which abstracts the concrete state as a tensor product results in a relational analysis. The structure of the LOS semantics guarantees that dependencies and correlations between variables are preserved in the abstract semantics $\mathbf{T}^{\#}$.

3.3 Non-Relational or Independent Abstraction

Let us first describe intuitively a probabilistic abstraction which ignores the relations between variables. In the analysis (i) of the program $\mathtt{x},\mathtt{y} \in \{1,2\}$; `x:=y;` given in Section 3, we can interpret the two variables `x` and `y` as *random variables*. Then the original state represents the *joint probability distribution* for both variables, i.e. a tensor product. The non-relational abstraction consists in retaining only their *marginal probability distribution*, see e.g. [11, Chapter IX].

For example, if $\mathtt{x} \in \{1,2\}$ takes value 1 with probability 0.7, and otherwise we have $\mathtt{x} = 2$, then the final state of the program is depicted in Figure 1(a). The centre of the array represents the concrete state, *e.g.* $P(x = 1 \wedge y = 2) = 0$. The last row and column represents the abstract state. If we consider only the abstract state, unreachable states like $\mathtt{x} = 1, \mathtt{y} = 2$ are no longer forbidden, because this state also abstracts the situation in Figure 1(b).

This kind of abstractions corresponds to a combination of abstractions for each variable which ignore the values of the other variables. This can be easily achieved by utilising the *forgetful* PAI which is represented by the abstraction $\mathbf{A}_f = (1, \ldots, 1)^t$. If it is applied to any distribution it results in the single scalar 1 (i.e. $d \cdot \mathbf{A}_f = (1)$). Thus, if we abstract the values of a single variable we need an abstraction of the form

$$\mathbf{A}(x_k) = (\bigotimes_{i=1}^{k-1} \mathbf{A}_f) \otimes \mathbf{A}_k \otimes (\bigotimes_{i=k+1}^{v} \mathbf{A}_f)$$

with $\mathbf{A}_k$ the intended abstraction of $(x)_k$. We can combine these abstractions simply using the direct sum, i.e. $\mathbf{S} = \bigoplus_{k=1}^{v} \mathbf{A}(x_k)$.

3.4 Weakly Relational Abstraction

In static analysis, forgetting all relationship between variables is a strong abstraction which destroys important information. Attempts to overcome this and to retain some relational information lead to simple and rather efficient algorithms. In his thesis [14], Antoine Miné introduces weakly relational domains, which rely on limited linear constraints. The *zone* abstract domain handles invariants of the form $X - Y \leq c$ and $\pm X \leq c$. The octagon abstract domain is an extension of the latter and handles invariants of the form $\pm X \pm Y \leq c$.

In the PAI framework we can define weakly relational abstraction by either allowing *weaker* abstractions $\tilde{\mathbf{A}}$ for some of the variables or abstracting additional information via abstractions $\mathbf{B}$. More concretely, we can consider abstractions of the form

$$\mathbf{A}(x_k) = (\bigotimes_{i=1}^{k-1} \tilde{\mathbf{A}}_i) \otimes \mathbf{A}_k \otimes (\bigotimes_{i=k+1}^{v} \tilde{\mathbf{A}}_i)$$

and $\mathbf{S} = \bigoplus_{k=1}^{v} \mathbf{A}(x_k)$, or alternatively, we could add additional terms in the definition of the non-relational abstraction above, i.e. $\mathbf{S} = (\bigoplus_{k=1}^{v} \mathbf{A}(x_k)) \oplus \bigoplus_l \mathbf{B}_l$ with

$$\mathbf{A}(x_k) = (\bigotimes_{i=1}^{k-1} \mathbf{A}_f) \otimes \mathbf{A}_k \otimes (\bigotimes_{i=k+1}^{v} \mathbf{A}_f),$$

and $\mathbf{B}_l$ some abstractions, e.g. the difference between any pair of variables as in Miné's zone and octagon analysis.

4 PAI and Precision

We can restrict ourselves w.l.o.g. to abstraction operators which are surjective, i.e. $\mathbf{A}(\mathcal{C}) = \mathcal{D}$. In fact, given a PAI $(\mathcal{C}, \mathbf{A}, \mathcal{D}, \mathbf{G})$, we can always partition the abstract domain $\mathcal{D}$ by identifying those elements with the same concrete meaning. In this way we can ensure that any abstract object in $\mathcal{D}$ is the image of a concrete object in $\mathcal{C}$, i.e. we reduce the abstract domain to one which does not contain redundant objects, or equivalently, we turn the abstraction operator $\mathbf{A}$ into a surjective one. In this case the closed subspace of $\mathcal{C}$ corresponding to the projection $\mathbf{G} \circ \mathbf{A} = \mathbf{P}_G$ is isomorphic to $\mathbf{A}(\mathcal{C})$; thus we can restrict ourselves to consider only probabilistic abstract interpretations of the form $(\mathcal{C}, \mathbf{P}_G, \mathbf{P}_G(\mathcal{C}), \mathbf{I})$. This will allow us to identify orthogonal projections on a Hilbert space $\mathcal{H}$ (or equivalently its closed subspaces) with all probabilistic abstract interpretations for the given concrete domain $\mathcal{H}$.

Proposition 4.1 *Let $\mathcal{H}$ be a Hilbert space and let $P \subseteq \mathcal{H}$ be a closed subspace of $\mathcal{H}$. Then $(\mathcal{H}, \mathbf{A}^\dagger \circ \mathbf{A}, P, \mathbf{I})$ is a PAI iff $\mathbf{A}^\dagger \circ \mathbf{A}(\mathcal{H}) = P$.*

Based on this identification, we can define the lattice of probabilistic abstract interpretations on a given Hilbert space $\mathcal{H}$ by means of the lattice of orthogonal projections on $\mathcal{H}$.

4.1 The Ortholattice of Projections

As it is well-known in Quantum Mechanics, projection operators on a Hilbert space form a non-Boolean – in particular, non-distributive – lattice. This result dates back to the 1936 article by Birkhoff and von Neumann [2] where the authors' claimed objective was to "find a calculus of propositions which is formally indistinguishable from the calculus of linear subspaces of a Hilbert space with respect to *set products*, *linear sums* and *orthogonal complements*, and resembles the usual calculus of propositions with respect to *and*, *or* and *not*".

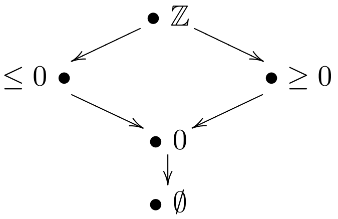
If Y is a closed subspace of a Hilbert space $\mathcal{H}$, each vector in $\mathcal{H}$ can be expressed uniquely in the form $y + z$ with $y \in Y$ and $z \in Y^\perp$, where $Y^\perp$ is a complementary subspace to Y (i.e. $Y + Y^\perp = \mathcal{H}$ and $Y \cap Y^\perp = \{o\}$). The linear operator $\mathbf{P} : \mathcal{H} \to Y$ defined by $\mathbf{P}(y + z) = y$ is called the *(orthogonal) projection* from $\mathcal{H}$ onto Y. It is easy to show that projection operators $\mathbf{P}$ are bounded (their norm is always less than or equal to 1) idempotent ($\mathbf{P}^2 = \mathbf{P}$) and Hermitian. An operator $\mathbf{A}$ is said to be *self-adjoint* or *Hermitian* if it coincides with its adjoint $\mathbf{A}^*$, that is the unique operator such that the condition $\langle \mathbf{A}^* x, y\rangle = \langle x, \mathbf{A} y\rangle$ holds for all $x, y \in \mathcal{H}$ (cf. e.g. [13, Thm 2.4.2]). In particular, projections are a special kind of self-adjoint operators, that is positive operators. An operator $\mathbf{A}$ is called *positive*, denoted by $\mathbf{A} \sqsupseteq 0$, if there exists an operator $\mathbf{B}$ such that $\mathbf{A} = \mathbf{B}^*\mathbf{B}$.

Projections can be identified with the closed subspaces of $\mathcal{H}$. As the range $Im(\mathbf{E}) = \{x\mathbf{E} \mid x \in \mathcal{H}\}$ of an orthogonal projection is a closed subspace, (cf. [4, Proposition II.3.2.b]), this correspondence is defined by associating to each projection on $\mathcal{H}$ its range $Im(\mathbf{E})$. We define the *ortho-complement* $\mathbf{E}^\perp$ of $\mathbf{E}$ as the projection with range $(Im(\mathbf{E}))^\perp$. Note that for all $\mathbf{E}$, the following holds:

$$\mathbf{E}^\perp = \mathbf{I} - \mathbf{E}.$$

The closed subspaces of $\mathcal{H}$ form a complete lattice under the operations of intersection and closed linear span. The one-to-one correspondence between this set and the collection $P(\mathcal{H})$ of all orthogonal projections on $\mathcal{H}$ allows us to transfer the lattice structure of the set of all closed subspaces of $\mathcal{H}$ to $P(\mathcal{H})$.

A partial order on projections (and in general on self-adjoint operators) can be defined directly by $\mathbf{E} \sqsubseteq \mathbf{F}$ iff $\mathbf{F} - \mathbf{E}$ is positive (e.g. [13, p105]). This is equivalent to the partial order defined via set inclusion on closed subspaces. More precisely,



$$\begin{aligned} sq(\emptyset) &= \emptyset \\ sq(0) &= 0 \\ sq(\leq 0) &= \geq 0 \\ sq(\geq 0) &= \geq 0 \\ sq(\mathbb{Z}) &= \geq 0 \end{aligned}$$

$$\mathbf{S} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Fig. 2. The lattice *Sign*, the concrete function *sq* and its matrix **S**

if $\mathbf{E}$ and $\mathbf{F}$ are projections from a Hilbert space $\mathcal{H}$ onto closed subspaces Y and Z respectively, then $\mathbf{E} \sqsubseteq \mathbf{F}$ iff $Y \subseteq Z$ (cf. [13, Proposition 2.5.2]).

The projections $\mathcal{P}(\mathcal{H})$ form a complete lattice with respect to this order, i.e. the least upper bound (lub) $\mathbf{E} \sqcup \mathbf{F}$ and the greatest lower bound (glb) $\mathbf{E} \sqcap \mathbf{F}$ always exist for any pair $\mathbf{E}$ and $\mathbf{F}$ and we have $(\mathbf{E} \sqcup \mathbf{F})^{\perp} = \mathbf{E}^{\perp} \sqcap \mathbf{F}^{\perp}$. The bottom element is given by the projection onto the null space, i.e. the operator mapping all vectors $x \in \mathcal{H}$ to the null vector, and the top element is the identity operator $\mathbf{I}$.

The problem of concretely constructing the lub $\mathbf{E} \sqcup \mathbf{F}$ and glb $\mathbf{E} \sqcap \mathbf{F}$ of two orthogonal projections $\mathbf{E}$ and $\mathbf{F}$ on $\mathcal{H}$ is in general considered as being not trivial. However, for commutative projections this can be constructed as $\mathbf{E} \sqcap \mathbf{F} = \mathbf{EF} = \mathbf{FE}$ and in the general case, using the Moore-Penrose pseudo-inverse, according to [1]:

$$\mathbf{E} \sqcap \mathbf{F} = 2\mathbf{E} : \mathbf{F} = 2\mathbf{E}(\mathbf{E} + \mathbf{F})^{\dagger}\mathbf{F}.$$

In Example 4.2 we will illustrate the use of this lattice structure on projections for comparing and constructing PAI's.

Example 4.2 We consider a small example presented in [18], where the concrete domain is the lattice *Sign* depicted in Figure 2.

Suppose that the concrete function to be analysed is $sq : Sign \rightarrow Sign$ defined as in Fig. 2. By fixing an enumeration of the elements in *Sign* we can lift the domain to a probabilistic domain. We consider the enumeration: 1. $\emptyset$, 2. 0, 3. ≤ 0, 4. ≥ 0, and 5. $\mathbb{Z}$, and define the vector space

$$\mathcal{V}(Sign) = \left\{ \sum_i a_i i \;\mid\; i \in Sign, a_i \in \mathbb{R} \right\}$$

which is isomorphic to the 5-dimensional real vector space $\mathbb{R}^5$. We can then define the matrix $\mathbf{S}$ as in Fig. 2 representing the linear operator on $\mathcal{V}(Sign)$ corresponding to the function sq.

The ortholattice $P(\mathcal{V}(Sign))$ (or equivalently the ortholattice of all closed subspace of $\mathcal{V}(Sign)$) gives all possible PAI's on the given concrete domain as depicted in Figure 3. The matrix representation of these projections is as follows:

$$\mathbf{P}_1 = \begin{pmatrix} \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \end{pmatrix}, \mathbf{P}_2 = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix}, \mathbf{P}_3 = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}, \mathbf{P}_4 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{pmatrix}$$

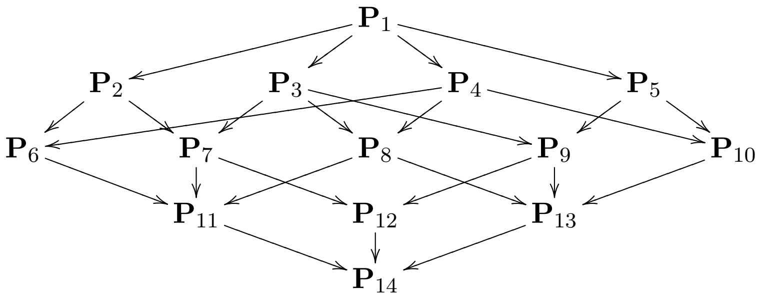


Fig. 3. The Lattice $\mathcal{P}(\mathcal{V}(\mathit{Sign}))$

$$\mathbf{P}_5 = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}, \; \mathbf{P}_6 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix}, \; \mathbf{P}_7 = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix}, \; \mathbf{P}_8 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}$$

$$\mathbf{P}_9 = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}, \; \mathbf{P}_{10} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}, \; \mathbf{P}_{11} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix}, \; \mathbf{P}_{12} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{P}_{13} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}, \; \mathbf{P}_{14} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

4.2 Precision of PAI's

Based on the Birkhoff-von Neumann order we can compare probabilistic abstract interpretations by identifying them with the corresponding closed subspaces or orthogonal projection operators. This allows us to say in some cases that one abstraction is "better" or "more precise" than another, that is to specify a notion of *relative precision*. We base our approach on the idea originally introduced in [10] that measuring the precision of an abstraction is actually measuring the *completeness* with respect to the concrete problem.

Completeness is not an essential requirement of an analysis, but rather an ideal situation which does not occur very often in practice. Intuitively, it expresses the property of an abstraction which "makes no mistakes". In a typical analysis framework, a function $f : \mathcal{A} \mapsto \mathcal{B}$ is given whose properties have to be analysed, and two different abstractions are specified on the input and the output domains respectively. In this case, a classical abstract interpretation is given by two abstraction functions, α and α', mapping the input domain $\mathcal{A}$ and the output domain $\mathcal{B}$ into an abstract input domain $\mathcal{A}^{\#}$ and an abstract output domain $\mathcal{B}^{\#}$ respectively, and by an abstract semantical functions $f^{\#} : \mathcal{A}^{\#} \mapsto \mathcal{B}^{\#}$. Then the abstract function $f^{\#} : \mathcal{A}^{\#} \mapsto \mathcal{B}^{\#}$ is said to be a *complete* approximation of a concrete function $f : \mathcal{A} \mapsto \mathcal{B}$ if $\alpha' \circ f = f^{\#} \circ \alpha$.

This notion of completeness applies essentially unchanged in the PAI setting by simply replacing classical domains and functions with probabilistic domains and linear functions. As shown e.g. in [5,18], the notion of completeness can be formulated in terms of closure operators by the equation

$$\eta \circ f = \eta \circ f \circ \rho,$$

where ρ is a closure operator on $\mathcal{A}$ expressing the input property, and η is a closure operator on $\mathcal{B}$ expressing the output property. This translates in the PAI setting to the following definition of completeness.

Definition 4.3 Given two Hilbert spaces $\mathcal{C}$ and $\mathcal{D}$ and a bounded linear map $\mathbf{F}$ between them, i.e. $\mathbf{F} : \mathcal{C} \to \mathcal{D}$, then we say that a pair of projections $\mathbf{P} : \mathcal{D} \to \mathcal{D}$ and $\mathbf{R} : \mathcal{C} \to \mathcal{C}$ is *complete* for $\mathbf{F}$ if and only if

$$\mathbf{FP} = \mathbf{RFP}.$$

Obviously, this also includes the special case of a bounded linear operator $\mathbf{F} : \mathcal{C} \mapsto \mathcal{C}$, where domain and codomain coincide (like e.g. for $\mathbf{F}$ being the fixpoint operator defining the semantics of a given program). In this case we have that a projection $\mathbf{P}$ on $\mathcal{C}$ is *complete* for $\mathbf{F}$ iff the pair $(\mathbf{P}, \mathbf{P})$ is complete for $\mathbf{F}$, i.e. $\mathbf{FP} = \mathbf{PFP}$. It is obvious that for any $\mathbf{F}$ there always exist at least two complete abstractions as for the trivial projections – i.e. identity $\mathbf{P} = \mathbf{I}$ and zero projection $\mathbf{P} = \mathbf{O}$ – the equation $\mathbf{FP} = \mathbf{PFP}$ is obviously fulfilled.

By using the equation defining the completeness of an abstraction $(\mathbf{P}, \mathbf{R})$ for a function $\mathbf{F}$ we can get an estimate of its precision by measuring the "difference" between $\mathbf{FP}$ and its optimal version $\mathbf{RFP}$. Intuitively, this difference represents the information missing in the given abstraction to get completeness or, in other words, the quantity of "false positives" in the associated analysis. To this purpose we use the Hilbert space norm and define the function $Prec_{\mathbf{F}} : P(\mathcal{H})^2 \to \mathbb{R}$ for a given semantical function $\mathbf{F}$ by

$$Prec_{\mathbf{F}}(\mathbf{P}, \mathbf{R}) = \|\mathbf{FP} - \mathbf{RFP}\| = \|(\mathbf{I} - \mathbf{R})\mathbf{FP}\| = \|\mathbf{R}^{\perp}\mathbf{FP}\|.$$

The function $Prec_{\mathbf{F}}$ introduces a total ordering on all the approximations for $\mathbf{F}$. This ordering is compatible with the Birkhoff-von Neumann ordering as shown by the following proposition.

Proposition 4.4 *Let $\mathbf{F} : \mathcal{H}_1 \mapsto \mathcal{H}_2$ be a bounded linear operator between two Hilbert spaces $\mathcal{H}_1$ and $\mathcal{H}_2$, and let $\mathbf{P}_1, \mathbf{P}_2 \in P(\mathcal{H}_2)$ and $\mathbf{R}_1, \mathbf{R}_2 \in P(\mathcal{H}_1)$. Then we have that if $\mathbf{P}_1 \sqsubseteq \mathbf{P}_2$ and $\mathbf{R}_2 \sqsubseteq \mathbf{R}_1$ then*

$$Prec_{\mathbf{F}}(\mathbf{P}_1, \mathbf{R}_1) \leq Prec_{\mathbf{F}}(\mathbf{P}_2, \mathbf{R}_2).$$

Note that, since $Prec_{\mathbf{F}}(\mathbf{P}, \mathbf{R})$ is a number expressing the loss in terms of completeness of the given abstraction, the smaller it is the more precise is the resulting analysis. We will prove Proposition 4.4 based on the following two properties.

Lemma 4.5 (Order and ortho-complement) *Let $\mathbf{P}$ and $\mathbf{R}$ be two orthogonal projections on a Hilbert space $\mathcal{H}$. Then*

$$\mathbf{P} \sqsubseteq \mathbf{R} \quad \Rightarrow \quad \mathbf{R}^{\perp} \sqsubseteq \mathbf{P}^{\perp}$$

Proof. By [13, Prop. 2.5.2] we have that $\mathbf{P} \sqsubseteq \mathbf{R}$ iff $Im(\mathbf{P}) \subseteq Im(\mathbf{R})$. Since the following property holds for the ortho-complement sub-spaces of Hilbert spaces (e.g. [15])

$$A \subseteq B \quad \Rightarrow \quad B^{\perp} \subseteq A^{\perp},$$

we have that

$$Im(\mathbf{P}) \subseteq Im(\mathbf{R}) \quad \Rightarrow \quad (Im(\mathbf{R}))^{\perp} \subseteq (Im(\mathbf{P}))^{\perp}.$$

By definition, $\mathbf{R}^{\perp}$ and $\mathbf{P}^{\perp}$ are the projections with ranges $(Im(\mathbf{R}))^{\perp}$ and $(Im(\mathbf{P}))^{\perp}$, respectively. Then apply again [13, Prop. 2.5.2]. □

Lemma 4.6 (Order and composition) *Let* $\mathbf{P}$ *and* $\mathbf{R}$ *be two orthogonal projections on a Hilbert space* $\mathcal{H}$*, then*

$$\mathbf{P} \sqsubseteq \mathbf{R} \quad \Rightarrow \quad \mathbf{P} = \mathbf{P}\mathbf{R} = \mathbf{R}\mathbf{P}$$

Proof. See [13, Prop. 2.5.2]. □

We now can prove Proposition 4.4.

Proof. By the hypothesis that $\mathbf{R}_2 \sqsubseteq \mathbf{R}_1$ and by Lemma 4.5 and Lemma 4.6 we have for all $\mathbf{P} \in P(\mathcal{H}_2)$,

$$\|\mathbf{R}_1^{\perp}\mathbf{F}\mathbf{P}\| = \|\mathbf{R}_1^{\perp}\mathbf{R}_2^{\perp}\mathbf{F}\mathbf{P}\| \leq \|\mathbf{R}_1^{\perp}\| \cdot \|\mathbf{R}_2^{\perp}\mathbf{F}\mathbf{P}\| \leq \|\mathbf{R}_2^{\perp}\mathbf{F}\mathbf{P}\|,$$

where the second inequality is simply due to the sub-multiplicativity of the norm and the last inequality is based on the fact that the norm of an orthogonal projection is always less or equal to 1.

On the other hand, since $\mathbf{P}_1 \sqsubseteq \mathbf{P}_2$ we have by Lemma 4.6,

$$\|\mathbf{R}_2^{\perp}\mathbf{F}\mathbf{P}_1\| = \|\mathbf{R}_2^{\perp}\mathbf{F}\mathbf{P}_2\mathbf{P}_1\|.$$

Therefore we have

$$\|\mathbf{R}_1^{\perp}\mathbf{F}\mathbf{P}_1\| \leq \|\mathbf{R}_2^{\perp}\mathbf{F}\mathbf{P}_1\| = \|\mathbf{R}_2^{\perp}\mathbf{F}\mathbf{P}_2\mathbf{P}_1\| \leq \|\mathbf{R}_2^{\perp}\mathbf{F}\mathbf{P}_2\|,$$

where the last inequality is again due to the sub-multiplicativity of the norm and the fact that the norm of an orthogonal projection is always less or equal to 1. □

Proposition 4.4 establishes in particular that the (relative) precision $Prec_{\mathbf{F}}(\mathbf{P}, \mathbf{R})$ for any semantical function $\mathbf{F}$ is monotone in $\mathbf{P}$ and anti-monotone in $\mathbf{R}$ with respect to the Birkhoff-von Neumann order, i.e. $\mathbf{P}_1 \sqsubseteq \mathbf{P}_2$ implies $Prec_{\mathbf{F}}(\mathbf{P}_1, \mathbf{R}) \leq Prec_{\mathbf{F}}(\mathbf{P}_2, \mathbf{R})$ for any fixed $\mathbf{R}$ and $\mathbf{R}_1 \sqsupseteq \mathbf{R}_2$ implies $Prec_{\mathbf{F}}(\mathbf{P}, \mathbf{R}_1) \leq Prec_{\mathbf{F}}(\mathbf{P}, \mathbf{R}_2)$ for any fixed $\mathbf{P}$, respectively.

Example 4.7 Consider again Example 4.2. The following table presents the precisions $Prec_{\mathbf{S}}(\mathbf{P}_i, \mathbf{P}_j)$ of the abstractions for the square function $\mathbf{S}$ defined by each pair of projections $(\mathbf{P}_i, \mathbf{P}_j)$, $i, j = 1, 2, \ldots, 14$ in the lattice of PAI's on $\mathcal{V}(Sign)$.

	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$	$\mathbf{P}_5$	$\mathbf{P}_6$	$\mathbf{P}_7$	$\mathbf{P}_8$	$\mathbf{P}_9$	$\mathbf{P}_{10}$	$\mathbf{P}_{11}$	$\mathbf{P}_{12}$	$\mathbf{P}_{13}$	$\mathbf{P}_{14}$
$\mathbf{P}_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\mathbf{P}_2$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\mathbf{P}_3$	1	0.75	0	0.79	0.75	0.65	0	0	0	0.65	0	0	0	0
$\mathbf{P}_4$	1	0.91	0.79	0	0.91	0	0.79	0	0.79	0	0	0.79	0	0
$\mathbf{P}_5$	1	0.75	0	0.79	0.75	0.65	0	0	0	0.65	0	0	0	0
$\mathbf{P}_6$	1.10	1	0.87	0	1	0	0.87	0	0.87	0	0	0.87	0	0
$\mathbf{P}_7$	1.34	1	0	1.06	1	0.87	0	0	0	0.87	0	0	0	0
$\mathbf{P}_8$	1	1	1	1	1	0.82	1	0	1	0.82	0	1	0	0
$\mathbf{P}_9$	1.10	0.82	0	0.87	0.82	0.71	0	0	0	0.71	0	0	0	0
$\mathbf{P}_{10}$	1.07	0.91	0.87	0.87	0.91	0.71	0.87	0	0.87	0.71	0	0.87	0	0
$\mathbf{P}_{11}$	1.34	1	1	1.22	1	1	1	0	1	1	0	1	0	0
$\mathbf{P}_{12}$	1.34	1	0	1.06	1	0.87	0	0	0	0.87	0	0	0	0
$\mathbf{P}_{13}$	1.10	1	1	1.06	1	0.87	1	0	1	0.87	0	1	0	0
$\mathbf{P}_{14}$	1.34	1	1	1.22	1	1	1	0	1	1	0	1	0	0

We see from this table that with respect to the input property $\mathbf{P}_3$, the abstractions $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, \mathbf{P}_5, \mathbf{P}_7, \mathbf{P}_9$ and $\mathbf{P}_{12}$ are complete probabilistic abstract interpretations for the square function as the corresponding entries in the table are zero. This reflects the classical situation in the original example of [18]. Moreover, in our setting the non-zero entries indicate the degree by which a PAI fails to be complete with respect to the square function.

5 Examples

The aim of the following examples is to illustrate in very simple cases how the relationality improves the precision of abstractions and how to measure the quality of abstractions. The abstractions $\mathbf{A}_k$ we consider here are variations of k-cast, i.e. arithmetics modulo k, in our case $k = 2$ (simple parity analysis) and $k = 4$. Besides this we also need to consider the forgetful abstraction $\mathbf{A}_1 = \mathbf{A}_f$ which "ignores" the value of a given variable. For a single variable the three abstractions are represented by matrices with entries: $(\mathbf{A}_k)_{ij} = 1$ if $i \bmod k = j - 1$, and zero otherwise.

We then construct the abstractions of v variables $x_1, x_2, \ldots, x_v$ as a linear map from the v-fold tensor product into some abstract domain. This domain is typically a direct sum of "small" vector spaces which record the value of the $\mathtt{x}_i$'s modulo 2 or 4. These abstractions are all of the form

$$\mathbf{S} = \bigoplus_{i=1}^{v} \mathbf{S}(x_i) \text{ with } \mathbf{S}(x_i) = (\bigotimes_{k=1}^{i-1} \mathbf{S}_{\neg i}) \otimes \mathbf{S}_i \otimes (\bigotimes_{k=i+1}^{v} \mathbf{S}_{\neg i})$$

where $\mathbf{S}_i$ is the abstraction applied to the variable $\mathtt{x}_i$ while $\mathbf{S}_{\neg i}$ represent the abstractions of the other variables. We compare three variations of $\mathbf{S}$: (i) $\mathbf{S}_r$ with $\mathbf{S}_i = \mathbf{S}_{\neg i} = \mathbf{A}_4$, (ii) $\mathbf{S}_w$ with $\mathbf{S}_i = \mathbf{S}_4, \mathbf{S}_{\neg i} = \mathbf{A}_2$, and (iii) $\mathbf{S}_n$ with $\mathbf{S}_i = \mathbf{S}_4, \mathbf{S}_{\neg i} = \mathbf{A}_1$, more concretely:

$$\mathbf{S}_r = \bigoplus_{i=1}^{v} \mathbf{S}(x_i) \quad \text{with} \quad \mathbf{S}(x_i) = (\bigotimes_{k=1}^{i-1} \mathbf{A}_4) \otimes \mathbf{A}_4 \otimes (\bigotimes_{k=i+1}^{v} \mathbf{A}_4)$$

$$\mathbf{S}_w = \bigoplus_{i=1}^{v} \mathbf{S}(x_i) \quad \text{with} \quad \mathbf{S}(x_i) = (\bigotimes_{k=1}^{i-1} \mathbf{A}_2) \otimes \mathbf{A}_4 \otimes (\bigotimes_{k=i+1}^{v} \mathbf{A}_2)$$

$$\mathbf{S}_n = \bigoplus_{i=1}^{v} \mathbf{S}(x_i) \quad \text{with} \quad \mathbf{S}(x_i) = (\bigotimes_{k=1}^{i-1} \mathbf{A}_1) \otimes \mathbf{A}_4 \otimes (\bigotimes_{k=i+1}^{v} \mathbf{A}_1)$$

The abstraction $\mathbf{S}_r$ is fully relational; in fact it is just v copies of the same fully relational abstraction $\bigotimes_{k=1}^{v} \mathbf{A}_4$. We thus can restrict ourselves to just one of the components in the direct sum. We have $\mathbf{S}_r : \mathcal{V}(n)^{\otimes v} \rightarrow \mathcal{V}(4)^{\otimes v}$, i.e. the abstract domain is also a tensor product, but a reduced one; for three variables the abstract domain has $4^3 = 64$ dimensions. This abstraction preserves the dependency between the "4ness" of all variables. For example, we record not only (the probability) that x_i is a multiple of 4 (i.e. $x_i \bmod 4 = 0$) but also (the probability) that x_i and x_j are multiples of 4 at the same time.

The abstraction $\mathbf{S}_n$ is non-relational or independent. In this abstraction we abstract only (the probabilities) that x_i has a certain remainder when divided by 4, but the relationship between the "4ness" of variables is lost. We extract, for example, no information whether x_i is multiple of 4 whenever x_j is. This is achieved by abstracting in each term in the direct sum $\bigoplus_{i=1}^{v} \mathbf{S}(x_i)$ only the "4ness" of one variable, namely x_i, while ignoring the value of all the others. The abstraction is of the type $\mathbf{S}_n : \mathcal{V}(n)^{\otimes v} \rightarrow \mathcal{V}(4)^{\oplus v} = \mathcal{V}(4)^v$; for three variables the abstract domain is a $3 * 4 = 12$ dimensional abstract domain.

The abstraction $\mathbf{S}_w$ is weakly relational. We get in this case some information about the correlation between the values of two variables, but not, like with $\mathbf{S}_r$ modulo 4 but only about the parity ($x_i \bmod 2$). The dimensionality of this abstraction thus is a compromise between $\mathbf{S}_r$ and $\mathbf{S}_n$, i.e. we have $\mathbf{S}_w : \mathcal{V}(n)^{\otimes v} \rightarrow (\mathcal{V}(4) \otimes \bigotimes_{i=1}^{v-1} \mathcal{V}(2))^{\oplus v}$. In the case of three variables this means that the abstract domain has $3 \times 8 = 24$ dimensions.

Regarding the precision of these three abstractions when used to analyse various extremely short programs our experiments resulted in the following. For a program like `var x:[0..10]; begin x:=k; stop` (with `k` $=$ 1 or `k` $=$ 4) and `var x:[0..10]; y:[0..10]; begin x:=y; stop` we obtain the following relative precisions $Prec_{\mathbf{T}}(\mathbf{P}, \mathbf{R})$.

$\mathbf{R}$ / $\mathbf{P}$	$\emptyset$	$\mathbf{S}_n$	$\mathbf{S}_w$	$\mathbf{S}_r$	*id*
$\emptyset$	0	0	0	0	0
$\mathbf{S}_n$	1.58	0	0	0	0
$\mathbf{S}_w$	1.58	0	0	0	0
$\mathbf{S}_r$	1.58	0	0	0	0
id	2.55	1	1	1	0

$\mathbf{R}$ / $\mathbf{P}$	$\emptyset$	$\mathbf{S}_n$	$\mathbf{S}_w$	$\mathbf{S}_r$	*id*
$\emptyset$	0	0	0	0	0
$\mathbf{S}_n$	1.73	0	0	0	0
$\mathbf{S}_w$	2.24	1	0	0	0
$\mathbf{S}_r$	2.24	1	1	0	0
id	3.61	3.61	3.61	3.61	0

For another set of programs `var x:[0..10]; y:[0..3]; begin x:=k*y; stop` we get for `k` $= 2$ and `k` $= 3$:

$\mathbf{R}$ / $\mathbf{P}$	$\emptyset$	$\mathbf{S}_n$	$\mathbf{S}_w$	$\mathbf{S}_r$	*id*
$\emptyset$	0	0	0	0	0
$\mathbf{S}_n$	1.88	0.89	0.89	0.89	0
$\mathbf{S}_w$	2.14	1.52	1.29	1.29	0
$\mathbf{S}_r$	2.24	1.64	1.50	1.41	0
id	3.61	3.60	3.59	3.58	0

$\mathbf{R}$ / $\mathbf{P}$	$\emptyset$	$\mathbf{S}_n$	$\mathbf{S}_w$	$\mathbf{S}_r$	*id*
$\emptyset$	0	0	0	0	0
$\mathbf{S}_n$	1.77	0.89	0.89	0.89	0
$\mathbf{S}_w$	2.24	1.52	1.29	1.29	0
$\mathbf{S}_r$	2.24	1.64	1.50	1.41	0
id	3.61	3.60	3.59	3.58	0

All these examples, despite their trivial nature, exhibit quite clearly how relationality can improve the precision of an analysis. When we go from left to right, i.e. from the trivial abstraction to $\mathbf{S}_n$, to $\mathbf{S}_w$, to $\mathbf{S}_r$, and finally to the concrete semantics $Prec_{\mathbf{T}}(\mathbf{P}, \mathbf{R})$ decreases. In particular the last row (comparison with the concrete semantics) describes a kind of absolute precision measure. Similarily, the first row describes the defect of the trivial abstraction with respect to what an other abstraction – increasing until we compare it with the concrete semantics. We also see that the precision measure gives different results for different types of programs. Sometimes the weakly relational analysis improves the analysis, sometimes it does not – in some cases the improvement could be considered to be relevant in other it is only marginal. In principle one could use this information to decide whether it is worth to apply $\mathbf{S}_n$, $\mathbf{S}_w$, or $\mathbf{S}_r$ by trading precision against the (sometimes substantial) additional overhead, i.e. dimensionality, of an analysis.

6 Conclusion

In this paper we have presented a framework for the relational analysis of a simple imperative language which is based on a linear operator semantics and probabilistic abstract interpretation. The main advantages of this are:

(i) the exploitation of the tensor product operation for obtaining fully relational analyses: as both the concrete and the abstract semantics can be factorised, different properties can be analysed simultaneously;

(ii) the use of probabilistic domains (in the form of finite dimensional inner product linear spaces) and the Moore-Penrose pseudo-inverse for a more speculative interpretation of the analysis results and in particular for obtaining a quantitative estimate of their precision.

The latter point could be achieved via the identification of each abstraction $\mathbf{A}$ with its associated orthogonal projection $\mathbf{A}\mathbf{A}^{\dagger}$ and the use of the Birkhoff - von Neumann lattice of projections. Based on the lattice of PAI's we have introduced a notion of relative precision expressing the degree of 'incompleteness' of a given abstraction via a number calculated as the norm of an appropriate linear operator. We have applied this to a small example comparing the precision of various numerical analyses.

Other works have faced the problem of quantifying the precision of a static analysis. In [17], Rountev, Kagan and Gibas present an approach to evaluating the imprecision of a static analysis via the lower and upper bound of the set of false positive. This is a quite standard method in classical static analysis, but the authors suggest that for each fact a human experimenter should find a proof that a given result is not a false positive. This approach leads to know the exact percentage of false positive for one program and analysis. However, as it must be done by hand, there is a serious limitation to the size of the set to be tested and it could not be used inside a tool, for example to select the best analysis for a given program. To this aim our approach seems to be more promising: Given that we have a structured lattice of abstract interpretations, and a way to measure precision, we can think of a more systematic approach in the selection of the abstract domain. In particular, this can be achieved practically via a statistical interpretation of the number expressing the relative precision.

An important point in further investigations is that the representation of the semantics of a program via a tensor product establishes a connection between the notion of relational dependencies in program analysis and that of correlation and independence in statistics [11].

References

[1] A. Ben-Israel and T.N.E. Greville. *Generalised Inverses.* Springer Verlag, 2nd edition, 2003.

[2] G. Birkhoff and J. von Neumann. The logic of quantum mechanics. *Annals of Mathematics*, 37:823–843, 1936.

[3] A. Böttcher and B. Silbermann. *Introduction to Large Truncated Toeplitz Matrices.* Springer Verlag, 1999.

[4] John Conway. *A Course in Functional Analysis*, volume 96 of *Graduate Texts in Mathematics.* Springer Verlag, second edition, 1990.

[5] P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Proceedings of POPL'79*, pages 269–282, 1979.

[6] A. Di Pierro, C. Hankin, and H. Wiklicky. Measuring the confinement of probabilistic systems. *Theoretical Computer Science*, 340(1):3–56, 2005.

[7] A. Di Pierro, C. Hankin, and H. Wiklicky. On probabilistic techniques for data flow analysis. In *Proceedings of QAPL'07*, volume 190 of *ENTCS*, pages 59–77, 2007.

[8] A. Di Pierro, C. Hankin, and H. Wiklicky. A systematic approach to probabilistic pointer analysis. Proceedings of APLAS'07, to appear.

[9] A. Di Pierro and H. Wiklicky. Concurrent Constraint Programming: Towards Probabilistic Abstract Interpretation. In *Proceedings of PPDP'00*, pages 127–138, 2000.

[10] A. Di Pierro and H. Wiklicky. Measuring the precision of abstract interpretations. In *Proc. of LOPSTR'00*, volume 2042 of *LNCS*, pages 147–164. Springer Verlag, 2001.

[11] W. Feller. *An Introduction to Probability Theory and Its Applications, Vol. 1.* Wiley & Sons, New York, 3rd edition, 1970.

[12] N.D. Jones and S.S. Muchnick. Complexity of flow analysis, inductive assertion synthesis and a language due to Dijkstra. In S.S. Muchnick and N.D. Jones, editors, *Program Flow Analysis: Theory and Applications*, pages 380–393. Prentice-Hall, 1981.

[13] R.V. Kadison and J.R. Ringrose. *Fundamentals of the Theory of Operator Algebras I*, volume 15 of *Graduate Studies in Mathematics.* AMS, 1997.

[14] A. Miné. *Weakly Relational Numerical Abstract Domains.* PhD thesis, École Normale Supérieure, Paris, 2004.

[15] J.M. Monier. *Analyse MP.* Dunod, 2004.

[16] F. Nielson, H. Riis Nielson, and C. Hankin. *Principles of Program Analysis.* Springer Verlag, 1999.

[17] A. Rountev, S. Kagan, and M. Gibas. Evaluating the imprecision of static analysis. In *Workshop on Program Analysis for Software Tools and Engineering*, 2004.

[18] Francesca Scozzari. *Domain theory in abstract interpretation: equations, completeness and logic.* PhD thesis, University of Siena, 1999.