Abstract Interpretation for Worst and Average Case Analysis

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

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

Abstract. We review Wilhelm's work on WCET for hard real-time applications and also recent work on analysis of soft-real time systems using probabilistic methods. We then present Probabilistic Abstract Interpretation (PAI) as a quantitative variation of the classical approach; PAI aims to provide close approximations – this should be contrasted to the safe approximations studied in the standard setting. We discuss the relation between PAI and classical Abstract Interpretation as well as average case analysis.

1 Introduction

This paper has been written as a contribution to the Festschrift to celebrate the sixtieth birthday of Reinhard Wilhelm. As befits such an occasion, we have sought to relate some of our current interests to Reinhard's canon of work. Given such an illustrious career, there is no shortage of possible topics. We have taken Reinhard's relatively recent work on worst case execution times (WCET) as our inspiration. This combines abstract interpretation, the topic about which the middle author and Reinhard first met 21 years ago, with quantitative issues – an area which is the topic of much of our recent work.

Reinhard's approach, which is surveyed in [1], concentrates on WCET for hard real-time problems – examples include safety-critical systems such as control software in automobiles. It is not difficult to see traditional approaches to WCET, for example those based on Timing Schemata [2], as abstract interpretation. States are abstracted to upper bounds on execution times and language constructs abstracted to work with these. For example a suitable interpretation for the conditional construct might be the function $\lambda xyz.x + max(y, z)$. The complexities of modern processors mean that WCET is much less predictable than this; cache performance and pipelines can have a dramatic impact and timing anomalies [3] can have counter-intuitive effects. As a consequence, Reinhard's approach is much more sophisticated: the first phase extracts a control flow graph – for some classes of language this may already involve an abstract interpretation or program analysis; he then performs some abstract interpretation to predict cache and pipeline behaviour; the result of this is upper bounds on the execution times for basic blocks and these are combined and integer linear programming is used to produce an upper bound on WCET for the whole

T. Reps, M. Sagiv, and J. Bauer (Eds.): Wilhelm Festschrift, LNCS 4444, pp. 160–174, 2007.

program. The paper [1] analyses the sources of unpredictability in these bounds and is actually a manifesto for "Design for Predictability".

In safety-critical embedded systems it is essential to work with worst case execution times. In systems with softer real-time constraints average case execution times can be a useful design tool in analysing power/performance trade-offs. Example application areas are multi-media and mobile devices. Approaches based on stochastic modelling have become popular in recent years. Stochastic Petri Nets [4,5], Stochastic Automata Networks [6] and Stochastic State Machine Languages have all been used. Kwiatkowska [7] and co-workers have used the latter with PRISM and MAPLE to optimise average energy usage whilst bounding the average number of requests waiting to be served. PRISM is used to generate the generator stochastic matrix for systems and MAPLE is used for formulating and solving the linear optimisation problem. As an alternative to model-checking, we have developed an approach to abstract interpretation of such stochastic matrices [8]. In this paper we explore how our framework can be used to study average case behaviour.

2 Probabilistic Abstract Interpretation

Without doubt, static program analysis is a fascinating and interesting area from a purely theoretical point of view – but it is done for a practical reason: In order to protect oneself against nasty surprises when software is run, the idea is to predict in advance what will happen when a program is executed. Unfortunately, well known fundamental results, like the Halting problem, tell us that it is in principle impossible to know everything about the behaviour of every program. The solution to this obstacle of undecidabilty is to aim for partial answers to some of the questions.

However, different applications and users have different priorities and interests and therefore accept different kinds of imprecision. When it comes, for example, to systems which are critical for life and limb of humans one might be cautious and attempt to determine absolute limits on what can go wrong in the worst case – like in the case of safety critical systems in cars, air planes, etc. If, on the other hand, the possible damage is only in terms of lost money, time or other resource one might be inclined to accept an estimate in order to forecast average profits or losses – as in the context of speculative threading, power consumption of mobile devices, etc.

There is no one-size-fits-all approach to this issue: While the cliché of a German engineer will aim to protect his "Vorsprung durch Technik" by accommodating for a worst case catastrophe, the caricature of a British speculator might gamble on an average case scenario – and both approaches can be justified in certain circumstances.

Abstract Interpretation [9,10,11] provides a general methodology for constructing static analyses which is, to some extent, independent of the particular

style used to specify the program analysis. Thus, it applies to any formulation of a (data/control flow or type/effect-system) analysis.

One common theme behind all traditional approaches to program analysis (data and control flow analysis, type and effect systems, abstract interpretation) is that in order to remain computable, one can only provide *approximate answers* [11]. As a consequence an analysis does not usually give precise information; moreover, in order for this information to be useful, the analysis must be *safe*, that is the information obtained from the analysis must be proved to be correct with respect to a semantics of the programming language.

Quantitative approaches to program analysis aim at developing techniques which provide approximate answers (in a way similar to the classical program analysis) together with some numerical estimate of the approximation introduced by the analysis.

One useful source of numerical information for a quantitative program analysis is a probabilistic semantics and in particular the use of vector space or linear algebraic structures for modelling the computational domain. By exploiting the probabilistic information resulting from a probabilistic program analysis one can quantify the precision of the analysis and obtain as a result answers which are for example "approximate up to 35%".

As a quantitative approach to program analysis we have developed Probabilistic Abstract Interpretation (PAI) [8,12] which recasts classical Abstract Interpretation in a probabilistic setting where linear spaces replace the classical order-theoretic domains, and the notion of the so-called *Moore-Penrose pseudo-inverse* of a linear operator replaces the classical notion of a Galois connection. The abstractions we get this way are *close* approximations of the concrete semantics. Thus, closeness is a quantitative replacement for classical safety which does not require any approximation ordering.

The application of operator algebraic methods instead of order theoretic ones makes the framework of probabilistic abstract interpretation essentially different from approaches which apply classical abstract interpretation to probabilistic domains [13,14]. Although classical AI techniques can also be used in a probabilistic context, e.g. to approximate distributions, as was demostrated for example in a number of papers by D.Monniaux [13,14], this will always result in safe, i.e. worst case analysis. In contrast, our PAI approach allows to construct averages and other statistical information which are more in the spirit of an average case analysis.

The definition of a probabilistic abstract interpretation is given in terms of *probabilistic domains*. We define a probabilistic domain as a suitable vector space with a inner product $\langle ., . \rangle$, namely as a Hilbert space.

Probabilistic Abstract Interpretation is defined in general for infinite dimensional Hilbert spaces. We recall here the general definition, although in this paper we will only consider the finite dimensional case. Given two probabilistic domains $\mathcal{C}$ and $\mathcal{D}$, a *probabilistic abstract interpretation* is defined by a pair of linear maps, $\mathbf{A} : \mathcal{C} \mapsto \mathcal{D}$ and $\mathbf{G} : \mathcal{D} \mapsto \mathcal{C}$, between the concrete domain $\mathcal{C}$ and the abstract domain $\mathcal{D}$, such that $\mathbf{G}$ is the *Moore-Penrose pseudo-inverse* of $\mathbf{A}$, and vice versa. Let $\mathcal{C}$ and $\mathcal{D}$ be two Hilbert spaces and $\mathbf{A} : \mathcal{C} \mapsto \mathcal{D}$ a

bounded linear map between them. A bounded linear map $\mathbf{A}^\dagger = \mathbf{G} : \mathcal{D} \mapsto \mathcal{C}$ is the Moore-Penrose pseudo-inverse of $\mathbf{A}$ iff

$$\mathbf{A} \circ \mathbf{G} = \mathbf{P}_A \quad \text{and} \quad \mathbf{G} \circ \mathbf{A} = \mathbf{P}_G$$

where $\mathbf{P}_A$ and $\mathbf{P}_G$ denote orthogonal projections (i.e. $\mathbf{P}_A^* = \mathbf{P}_A = \mathbf{P}_A^2$ and $\mathbf{P}_G^* = \mathbf{P}_G = \mathbf{P}_G^2$) onto the ranges of $\mathbf{A}$ and $\mathbf{G}$.

Alternatively, if $\mathbf{A}$ is Moore-Penrose invertible, its Moore-Penrose pseudo-inverse, $\mathbf{A}^\dagger$ satisfies the following:

$$\begin{array}{ll} \textbf{(i)}\ \mathbf{A}\mathbf{A}^\dagger\mathbf{A} = \mathbf{A}, & \textbf{(iii)}\ (\mathbf{A}\mathbf{A}^\dagger)^* = \mathbf{A}\mathbf{A}^\dagger, \\ \textbf{(ii)}\ \mathbf{A}^\dagger\mathbf{A}\mathbf{A}^\dagger = \mathbf{A}^\dagger, & \textbf{(iv)}\ (\mathbf{A}^\dagger\mathbf{A})^* = \mathbf{A}^\dagger\mathbf{A}, \end{array}$$

where $\mathbf{M}^*$ is the adjoint of $\mathbf{M}$. The adjoint $\mathbf{M}^*$ of a linear operator $\mathbf{M}$ on a Hilbert space $\mathcal{H}$ is uniquely defined via the condition $\langle \mathbf{M}(x), y\rangle = \langle x, \mathbf{M}^*(y)\rangle$, for all $x, y \in \mathcal{H}$. In matrix terms, $\mathbf{M}^*$ corresponds to the transpose complex conjugate matrix $\overline{\mathbf{M}^T}$ of the matrix $\mathbf{M}$.

It is instructive to compare these equations with the classical setting. For example, if (α, γ) is a Galois connection we similarly have $\alpha \circ \gamma \circ \alpha = \alpha$ and $\gamma \circ \alpha \circ \gamma = \gamma$.

Please note that we identify linear maps and operators with their matrix representation. This implies that for two linear maps or operators represented by matrices $\mathbf{M}$ and $\mathbf{N}$ their composition $\mathbf{M} \circ \mathbf{N}$ (if it is well-defined) corresponds to the matrix product $\mathbf{N} \cdot \mathbf{M} = \mathbf{N}\mathbf{M}$, i.e. in the reverse order. Similarily, the application of $\mathbf{M}$ to a (row) vector, i.e. $\mathbf{M}(x)$, corresponds to a vector/matrix multiplication $x \cdot \mathbf{M} = x\mathbf{M}$. This notation is consistent with the one for the `octave` tool [15] which was used by us to compute the examples in this paper.

As in the classical framework, given a concrete semantics we can always construct a *best correct approximation* for this semantics, although the notions of correctness and optimality assume a different connotation in our linear setting as explained in the following.

If Φ is a linear operator on some vector space $\mathcal{V}$ expressing the probabilistic semantics of a concrete system, and $\mathbf{A} : \mathcal{V} \mapsto \mathcal{W}$ is a linear abstraction function from the concrete domain into an abstract domain $\mathcal{W}$, we can compute the (unique) Moore-Penrose pseudo-inverse $\mathbf{G} = \mathbf{A}^\dagger$ of $\mathbf{A}$. An abstract semantics can then be defined as the linear operator on the abstract domain $\mathcal{W}$:

$$\Psi = \mathbf{A} \circ \Phi \circ \mathbf{G} = \mathbf{G}\Phi\mathbf{A}.$$

In the case of classical abstract interpretation the abstract semantics constructed in this way (called the *induced semantics in* [16]) is guaranteed to be the best correct approximation of the concrete semantics, meaning that it is the most precise among all correct approximation (the relative precision being left unquantified). In the linear space based setting of PAI where the order of the classical domains is replaced by some notion of metric distance, the induced abstract semantics is the *closest* one to the concrete semantics. This "closeness" property expresses both the "safety" of the approximation and its optimality, which comes from the following properties of the Moore-Penrose pseudo-inverse.

The theory of the least-square approximation [17,18] tells us that if $\mathcal{C}$ and $\mathcal{D}$ are two finite dimensional vector spaces, $\mathbf{A} : \mathcal{C} \mapsto \mathcal{D}$ a linear map between them, and $\mathbf{A}^\dagger = \mathbf{G} : \mathcal{D} \mapsto \mathcal{C}$ its Moore-Penrose pseudo-inverse, then the vector $x_0 = y\mathbf{G}$ is the one minimising the distance between $x\mathbf{A}$, for any vector x in $\mathcal{C}$, and y, i.e.

$$\inf_{x \in \mathcal{C}} \|x\mathbf{A} - y\| = \|x_0\mathbf{A} - y\|,$$

where $\|.\|$ denotes the usual *Euclidean* or *2-norm*.

In other words, if we consider the equation $x\mathbf{A} = y$ we can identify a (exact) solution x_* as a vector for which $\|x_*\mathbf{A} - y\| = 0$. In particular, in the case that no such solution vector x_* exists we can generalise the concept of a exact solution to that of a "pseudo-solution", i.e. we can look for a x_0 such that $x_0\mathbf{A}$ is the *closest* vector to y we can construct. This closest approximation to the exact solution is now constructed using the Moore-Penrose pseudo-inverse, i.e. take $x_0 = y\mathbf{A}^\dagger$.

Returning to our program analysis setting, suppose that we have an operator Φ and a vector x. We can apply Φ to x and abstract the result giving $x\Phi\mathbf{A}$ or we can apply the abstract operator to an abstract vector giving $x\mathbf{A}\mathbf{A}^\dagger\Phi\mathbf{A}$. Ideally, we would like these to be equal. If $\mathbf{A}$ is invertible then its Moore-Penrose pseudo-inverse is identical to the inverse and we are done. As in program analysis abstract domains are usually smaller (i.e. in our setting vector spaces of smaller dimension) than the concrete ones, $\mathbf{A}$ is never a square matrix and thus $\mathbf{A}\mathbf{A}^\dagger$ in $x\mathbf{A}\mathbf{A}^\dagger\Phi\mathbf{A}$ will lead to some loss of precision. The Moore-Penrose pseudo-inverse is as close as possible to an inverse if the matrix is not invertible and thus for the particular choice of $\mathbf{A}$, $\mathbf{A}^\dagger\Phi\mathbf{A}$ is the best approximation of Φ that we can have. Moreover, by choosing an appropriate notion of distance we can measure this closeness to get a quantitative estimate of the information lost in the abstraction process [12].

3 Approximations: A Classical Example

Classical abstract interpretation and probabilistic abstract interpretation provide "approximations" for completely different mathematical structures, namely partial orders vs vector spaces. In order to illustrate and compare their features we therefore need a setting where the domain in question in some way naturally provides both structures. One such situation is in the context of classical function interpolation or approximation.

The set of real-valued functions on real interval $[a, b]$ obviously comes with a canonical partial order, namely the point-wise ordering, and at the same time is equiped with a vector space structure, again the point-wise addition and scalar multiplication. Some care has to be taken in order to define an inner product, e.g. one could consider only the square integrable functions $L^2([a, b])$. In order to avoid mathematical (e.g. measure-theoretic) details we simplify the situation by just considering the step functions on the interval $[a, b]$.

For a (closed) real interval $[a, b] \subseteq \mathbb{R}$ we call the set of subintervals $[a_i, b_i]$ with $i = 1, \ldots, n$ the *n-subdivision* of $[a, b]$ if $\bigcup_{i=1}^{n} [a_i, b_i] = [a, b]$ and $b_i - a_i = \frac{b-a}{n}$ for all $i = 1, \ldots, n$. We assume that the subintervals are enumerated in the obvious way, i.e. $a_i < b_i = a_{i+1} < b_{i+1}$ for all i and in particular that $a = a_1$ and $b_n = b$.

Definition 1. *The set of* n-step functions $\mathcal{T}_n([a,b])$ *on* $[a,b]$ *is the set of real-valued functions* $f : [a,b] \to \mathbb{R}$ *such that* f *is constant on each subinterval* (a_i, b_i) *in the* n*-subdivision of* $[a,b]$.

Note that since L^2 is the set of equivalence classes of functions with respect to the equivalence relation $\sim$ defined by $f \sim g$ iff $\int |f(x) - g(x)|^2 dx = 0$, we can identify functions if they differ in only finitely many points. Thus, the values $f(a_i)$ and $f(b_i)$ are irrelevant for our purpose.

We define a partial order on $\mathcal{T}_n([a,b])$ in the obvious way: for $f, g \in \mathcal{T}_n([a,b])$:

$$f \sqsubseteq g \text{ iff } f(\frac{b_i - a_i}{2}) \leq g(\frac{b_i - a_i}{2}), \text{ for all } 1 \leq i \leq n$$

i.e. iff the value of f (which we obtain by evaluating it on the mid-point in (a_i, b_i)) on all subintervals (a_i, b_i) is less or equal to the value of g.

It is also obvious to see that $\mathcal{T}_n([a,b])$ has a vector space structure isomorphic to $\mathbb{R}^n$ and thus is also provided with an inner product. More concretely we define the vector space operations $. \cdot . : \mathbb{R} \times \mathcal{T}_n([a,b]) \to \mathcal{T}_n([a,b])$ and $. + . : \mathcal{T}_n([a,b]) \times \mathcal{T}_n([a,b]) \to \mathcal{T}_n([a,b])$ pointwise as follows:

$$(\alpha \cdot f)(x) = \alpha f(x)$$

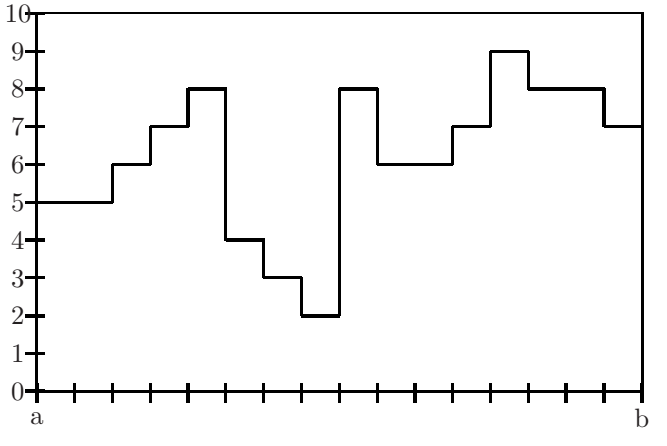
$$(f + g)(x) = f(x) + g(x)$$

for all $\alpha \in \mathbb{R}$, $f, g \in \mathcal{T}_n([a,b])$ and $x \in [a,b]$. The inner product is given by:

$$\langle f, g \rangle = \sum_{i=1}^{n} f(\frac{b_i - a_i}{2}) g(\frac{b_i - a_i}{2}).$$

In this setting we now can apply and compare both the classical and the quantitative version of abstract interpretation as in the following example.

Example 1. Let us consider a step function f in $\mathcal{T}_{16}$ (the concrete values of a and b don't really play a role in our setting) which can be depicted as:



We can also represent f by the vector in $\mathbb{R}^{16}$:

$$\begin{pmatrix} 5\,5\,6\,7\,8\,4\,3\,2\,8\,6\,6\,7\,9\,8\,8\,7 \end{pmatrix}$$

We then construct a series of abstractions which correspond to coarser and coarser sub-divisions of the interval $[a, b]$, e.g. considering 8, 4 etc. subintervals instead of the original 16. These abstractions are from $\mathcal{T}_{16}([a, b])$ to $\mathcal{T}_8([a, b])$, $\mathcal{T}_4([a, b])$ etc. and can be represented by 16×8, 16×4, etc. matrices. For example, the abstraction which joins two sub-intervals and which corresponds to the abstraction $\alpha_8 : \mathcal{T}_{16}([a, b]) \to \mathcal{T}_8([a, b])$ together with its Moore-Penrose pseudo-inverse is represented by:

$$\mathbf{A}_8 = \begin{pmatrix} 1&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&1 \end{pmatrix} \qquad \mathbf{G}_8 = \begin{pmatrix} \frac{1}{2}&\frac{1}{2}&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&\frac{1}{2}&\frac{1}{2}&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&\frac{1}{2}&\frac{1}{2}&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&\frac{1}{2}&\frac{1}{2}&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&\frac{1}{2}&\frac{1}{2}&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&\frac{1}{2}&\frac{1}{2}&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&\frac{1}{2}&\frac{1}{2}&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&\frac{1}{2}&\frac{1}{2} \end{pmatrix}$$

With the help of $\mathbf{A}_j$, $j \in \{1, 2, 4, 8\}$, we can easily compute the abstraction of f as $f\mathbf{A}_j$, which in order to compare it with the original f we can then again concretise using $\mathbf{G}$, i.e. computing $f\mathbf{AG}$. In a similar way we can also compute the over- and under-approximation of f in $\mathcal{T}_i$ based on the above pointwise ordering and its reverse ordering. The result of these abstractions is depicted geometrically in Figure 1.

The individual diagrams in this figure depict the original, i.e. concrete step function $f \in \mathcal{T}_{16}$ together with its approximations in $\mathcal{T}_8$, $\mathcal{T}_4$, etc. On the left hand side the PAI abstractions show how coarser and coarser interval subdivisions result in a series of approximations which try to interpolate the given function as closely as possible, sometimes below, sometimes above the concrete values. The diagrams on the right hand side depict the classical over- and under-approximations: In each case the function f is entirely below or above these approximations, i.e. we have safe but not necessarily close approximations. Additionally, one can also see from these figures not only that the PAI interpolation is in general closer to the original function than the classical abstractions (in fact it is the closest possible) but also that the PAI interpolation is always between the classical over- and under-approximations.

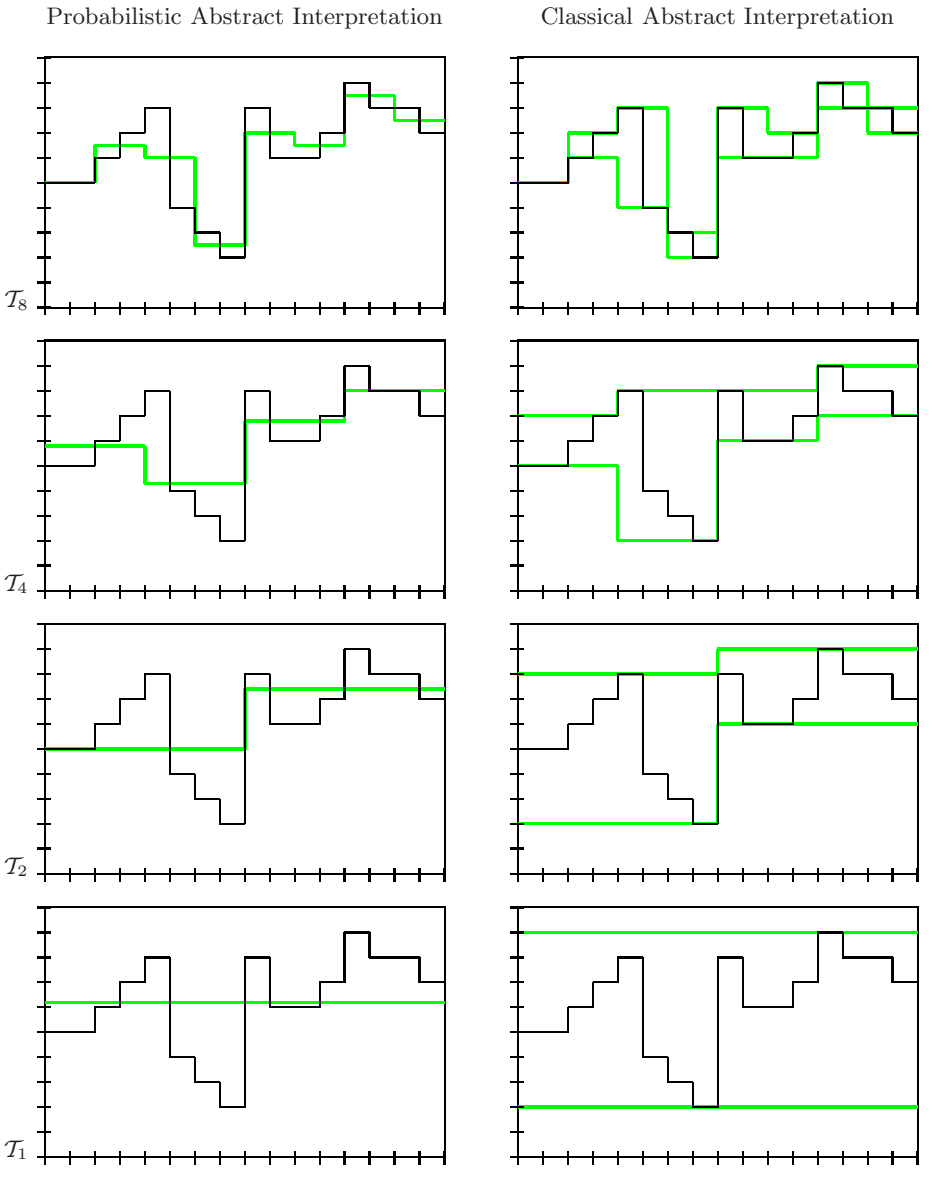


Fig. 1. Average, Over- and Under-Approximation

The vector space framework also allows us to judge the quality of an abstraction or approximation via the Eucledian distance between the concrete and abstract version of a function. We can compute the *least square error* as

$$\|f - f\mathbf{AG}\|.$$

In our case we get for example:

$$\begin{aligned}\|f - f\mathbf{A}_8\mathbf{G}_8\| &= 3.5355\\ \|f - f\mathbf{A}_4\mathbf{G}_4\| &= 5.3151\\ \|f - f\mathbf{A}_2\mathbf{G}_2\| &= 5.9896\\ \|f - f\mathbf{A}_1\mathbf{G}_1\| &= 7.6444\end{aligned}$$

which illustrates, as expected, that the coarser our abstraction is the larger is also the mistake or error.

This example also illustrates how PAI and averages are closely related. The coarsest abstraction $\alpha_1 : \mathcal{T}_{16} \to \mathcal{T}_1$ computes in effect the average of all the values of f, i.e. the (constant) value of $f\mathbf{A}_1\mathbf{G}_1$ is exactly the average of

$$5, 5, 6, 7, 8, 4, 3, 2, 8, 6, 6, 7, 9, 8, 8, 7.$$

In general, we can always compute such an average via the one point abstraction of an n-dimensional space given by

$$\mathbf{A}^T = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \end{pmatrix}$$

with Moore-Penrose pseudo-inverse given by

$$\mathbf{A}^\dagger = (\frac{1}{n}, \frac{1}{n}, \dots, \frac{1}{n}).$$

4 Parallel Systems

Multi-core processors and multi-threaded applications are emerging as the new models in computing. This makes the development of adequate tools and techniques for code parallelisation one of today's major challenges. At the same time the study of parallel systems and their performance also poses many difficulties due to their complexity deriving from the exponential growth of the number of states.

Automatic tools such as compilers must provide code that works in all cases. Therefore, when a compiler has to parallelise two pieces of code it has to consider all potential dependencies. Current techniques are over-conservative: if a dependency cannot be proved, the compiler assumes that it does occur. As a consequence, many opportunities for parallelisation are missed in the code generation.

Contrary to this "worse-case" conservative approach, speculative threading is an emerging technique which allows for some incorrect thread parallelisation by postponing the check to some later time. In the following example we will show how this "optimistic" approach can be justified via PAI. We will also show how

PAI can be used to face the state space explosion problem in the performance evaluation of parallel systems.

4.1 Tensor Models

We will now discuss how to abstract or simplify dynamical systems described by iterations of linear maps, in particular stochastic systems, i.e. Discrete Time Markov Chains (DTMCs).

There exist numerous formalisms to describe the semantics or dynamics of discrete as well as continuous time (stochastic) systems like, for example, Stochastic Petri Nets (SPNs) [4,5], Stochastic Automata Networks (SANs) [6], etc.

In practice, in most of these models, as well as in the authors' Probabilistic Chemical Abstract Machine (pCHAM) [19] and the linear semantics of Probabilistic KLAIM (pKLAIM) [20], the operation which combines the semantics of individual components of the system (nodes, etc.) in order to describe the global structure (network, etc.) is the *tensor product*. The tensor or Kronecker product $\mathbf{A} \otimes \mathbf{B}$ of an $n \times m$ matrix $\mathbf{A}$ and a $k \times l$ matrix $\mathbf{B}$ is an $nk \times ml$ matrix:

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{11}\mathbf{B} & \dots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{1m}\mathbf{B} & \dots & a_{nm}\mathbf{B} \end{pmatrix}$$

This construction has the advantage of a clean separation of local and global aspects, i.e. of the micro and macro dynamics of a system. However, the price to pay for this is the (exponential) state explosion introduced by the tensor product. The issue we discuss next is how to control this explosion, at least partially, by simplifying the model using the PAI framework.

Before we look at an example of such an abstraction, let us first present the basic elements of "tensor models". The basic entities of these models are simple counters, which could be interpreted as "token stores" (as in pKLAIM) or to indicate the "multiplicity of molecules" (as in the pCHAM model).

The basic local operations we allow for are essentially just counting operations in steps of $+1$ and -1. These are represented by so called shift operators:

$$(\mathbf{C})_{ij} = \begin{cases} 1 \text{ for } j = i+1 \\ 0 \text{ otherwise} \end{cases} \quad \text{and} \quad (\mathbf{D})_{ij} = \begin{cases} 1 \text{ for } j = i-1 \\ 0 \text{ otherwise} \end{cases}$$

or more concretely:

$$\mathbf{C} = \begin{pmatrix} 0\,1\,0 \dots 0\,0 \\ 0\,0\,1 \dots 0\,0 \\ 0\,0\,0 \dots 0\,0 \\ \vdots \quad \ddots \quad \vdots \\ 0\,0\,0 \dots 0\,1 \\ 0\,0\,0 \dots 0\,0 \end{pmatrix} \quad \mathbf{D} = \begin{pmatrix} 0\,0\,0 \dots 0\,0 \\ 1\,0\,0 \dots 0\,0 \\ 0\,1\,0 \dots 0\,0 \\ \vdots \quad \ddots \quad \vdots \\ 0\,0\,0 \dots 0\,0 \\ 0\,0\,0 \dots 1\,0 \end{pmatrix}$$

Additionally we need to provide for testing operations which determine whether a certain "counter" is below or above a certain threshold. We represent these by projection operators:

$$(\mathbf{T}_m^{\min})_{ij} = \begin{cases} 1 \text{ for } i = j \geq m \\ 0 \text{ otherwise} \end{cases} \quad \text{and} \quad (\mathbf{T}_m^{\max})_{ij} = \begin{cases} 1 \text{ for } i = j \leq n \\ 0 \text{ otherwise} \end{cases}$$

or more concretely again:

$$\mathbf{T}_m^{\min} = \begin{pmatrix} 0\,0\,0\,\ldots\,0\,0 \\ 0\,0\,0\,\ldots\,0\,0 \\ 0\,0\,0\,\ldots\,0\,0 \\ \vdots \quad\quad \ddots \quad\quad \vdots \\ 0\,0\,0\,\ldots\,1\,0 \\ 0\,0\,0\,\ldots\,0\,1 \end{pmatrix} \quad \mathbf{T}_n^{\max} = \begin{pmatrix} 1\,0\,0\,\ldots\,0\,0 \\ 0\,1\,0\,\ldots\,0\,0 \\ 0\,0\,1\,\ldots\,0\,0 \\ \vdots \quad\quad \ddots \quad\quad \vdots \\ 0\,0\,0\,\ldots\,0\,0 \\ 0\,0\,0\,\ldots\,0\,0 \end{pmatrix}$$

To simplify our treatment we assume in the following that all operators have the right dimension, e.g. are all $s \times s$ matrices. It could also be noted that by moving from finite dimensional matrix algebras to infinite dimensional operator algebras, e.g. C* algebras, one can drop limits for the counting operations, i.e. $s = \infty$.

Due to the huge size of "tensor models" it is necessary to simplify them in some way in order to extract certain global behavioural properties. The underlying idea is somewhat similar to the approach in statistical mechanics, where macroscopic parameters like pressure or temperature are explained or aggregated from the statistics of the microscopic features of individual molecules, such as their momentum, mass, velocity, etc.

The simplification of tensor product models using probabilistic abstract interpretation can drastically reduce the complexity of the analysis while we can still guarantee that the obtained results are as close as possible to the exact values. It also allows a kind of compositional analysis as Moore-Penrose pseudo-inverse and tensor product are compatible in the following sense: Given two (bounded) linear operators $\mathbf{A}_1$ and $\mathbf{A}_2$ on a Hilbert space, then

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

Therefore, by exploiting the algebraic properties of the tensor product, we can abstract tensor models component wise, i.e.

$$(\mathbf{A}_1 \otimes \mathbf{A}_2)^\dagger (\mathbf{T}_1 \otimes \mathbf{T}_2)(\mathbf{A}_1 \otimes \mathbf{A}_2) = (\mathbf{A}_1^\dagger \mathbf{T}_1 \mathbf{A}_1) \otimes (\mathbf{A}_2^\dagger \mathbf{T}_2 \mathbf{A}_2)$$

This is illustrated in the following example where we only abstract one part of the system while another one stays unchanged, i.e. we use an abstraction of the form $\mathbf{I} \otimes \mathbf{A}$ with $\mathbf{I}$ the identity.

4.2 Average Running Time

A simple example which relates to the issue of speculative threading or scheduling we discussed before concerns the parallel execution of two processes with

the following constraint: If thread A finishes before thread B then B has to be restarted. The reason could be, for example, the following scenario: process A puts its final result into some storage area or memory which is used by process B to store intermediate results: as long as process A does not finish, the results/computations in thread B are correct, but if process A finishes too early than we have to redo all its work again. The problem is to estimate the maximal and average chance that this happens.

Let us consider a slightly simplified version where we are interested in analysing the behaviour of A in parallel with B until a restarting is needed. Concretely we have two processes: A just executes three steps and then stops; B counts up to some number n, e.g. $n = 100$, and then terminates. However at every step B can also terminate immediately; the chances for continuing counting or for termination are $50 : 50$. The tensor model for this example is simply given by:

$$\mathbf{T}_n = \mathbf{A} \otimes \mathbf{B}_n$$

with A being represented simply by a creation operator in four dimensions:

$$\mathbf{A} = \mathbf{C}_4 = \begin{pmatrix} 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \\ 0\,0\,0\,0 \end{pmatrix}.$$

The representation of B_n is a bit more complicated:

$$\mathbf{B}_n = \frac{1}{2} \cdot \mathbf{C}_n + \frac{1}{2} \cdot (\sum_{i=0}^{n-1} \mathbf{T}_i^{\min} \mathbf{T}_i^{\max} (\mathbf{C}_n)^{n-i}).$$

In other words, this process either continues counting from i to $i + 1$, or if has already reached exactly i (for which we have a min and a max test) then by applying the $(n - i)$-th power of the creation operator produces the n which leads to the termination of the whole process. More concretely, for $n = 4$ we have:

$$\mathbf{B}_4 = \begin{pmatrix} 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The worst case analysis of these processes needs, in effect, to determine the longest running time of both process. That means we have to determine the *nilpotency* of $\mathbf{T}_n$ or at least of $\mathbf{A}$ and $\mathbf{B}_n$. It is easy to see that the longest possible running time, i.e. the nilpotency, for $\mathbf{A}$ is 3 steps, while for $\mathbf{B}_n$ is n. If we look at $\mathbf{T}_n = \mathbf{A} \otimes \mathbf{B}_n$, then it is easy to see that (for $n > 3$) the maximal running time is also as n.

There are a few problems with this analysis: firstly, in complexity theoretic terms, it requires to analyse a rather large $4n \times 4n$ matrix (and with more

processes the tensor product will cause a huge explosion in the dimension of the exact model); secondly, for unbounded counting in B we don't get any meaningful result; and thirdly, the results are somewhat meaningless as the chances that B counts to n stepwise, instead of terminating early are rather small such that the average running time of B is much smaller than n, i.e. in the scenario discussed earlier it is rather unlikely that we have to restart B.

In order to see this we can compute an abstracted version of B and compare this to the worst case estimates and the exact results. For this we identify all intermediate states except for the terminating state n. We can use the following simple abstraction represented by the $n \times 2$ matrix

$$(\mathbf{K})_{ij} = \begin{cases} 1 \text{ for } i < n \text{ and } j = 1 \\ 1 \text{ for } i = n \text{ and } j = 2 \\ 0 \text{ otherwise} \end{cases} \qquad \text{e.g. } \mathbf{K}_4 = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \mathbf{K}_4^\dagger = \begin{pmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

It is particular easy to compute $\mathbf{K}^\dagger$ for abstractions like $\mathbf{K}$ where every concrete state is uniquely classified, i.e. where every row of $\mathbf{K}$ contains only one non-zero entry equal to 1: $\mathbf{K}^\dagger$ is in this case obtained by transposing $\mathbf{K}$ followed by a row-normalisation. With this we can construct abstract versions $\mathbf{B}_n^\# = \mathbf{K}^\dagger \mathbf{B} \mathbf{K}$ of $\mathbf{B}_n$ which are given by only 2×2 matrices, i.e. the combination of the concrete A process and the abstract process $B_n^\#$ is represented by just a 6×6 matrix.

If we compare the concrete average running time of $\mathbf{B}_n$, which means we have to iterate a $n \times n$ matrix, with the average running times of its abstract 2×2 version we get the following numerical results.

	avg	avg$^\#$	$P(3)$	$P(3)^\#$	max
$\mathbf{B}_5^\# = \begin{pmatrix} 0.38 & 0.62 \\ 0.00 & 0.00 \end{pmatrix}$	1.8750	1.6000	0.8750	0.9473	5
$\mathbf{B}_{10}^\# = \begin{pmatrix} 0.44 & 0.56 \\ 0.00 & 0.00 \end{pmatrix}$	1.9961	1.8000	0.8750	0.9122	10
$\mathbf{B}_{25}^\# = \begin{pmatrix} 0.48 & 0.52 \\ 0.00 & 0.00 \end{pmatrix}$	2.0000	1.9200	0.8750	0.8750	25
$\mathbf{B}_{100}^\# = \begin{pmatrix} 0.49 & 0.51 \\ 0.00 & 0.00 \end{pmatrix}$	2.0000	1.9800	0.8750	0.8750	100

Here we present the simplified 2×2 systems and give the exact average running time 'avg', the abstract average running time (of the simplified system) 'avg$^\#$', the exact and abstract probabilities for terminating after three steps $P(3)$ and $P(3)^\#$, and finally the maximal, i.e. worst case running time max.

This example illustrates a number of important technical issues, e.g.: (i) how PAI can be used to reduce the complexity of a model, in this case from $4n \times 4n$ to just 6×6 matrices; and (ii) the small difference between the concrete properties and the PAI approximated values, e.g. of the average running time or the probability of stopping after a certain number of steps.

This example also demonstrates a case where worst case analysis, although correct, is practically nearly useless: The worst case running time of 100 steps for the process B_{100} is much larger than the 3 steps for A. This would suggest in a speculative threading situation that it would be pointless to execute A and B_{100} in parallel as B_{100} would finish too late and thus have to be restarted anyway. However, the average case analysis and in particular the value for $P(3)$ tells us that B_{100} will terminate with nearly 90% chance before A, and that it would therefore make sense to schedule these two processes in parallel.

5 Conclusions

We started by reviewing Reinhard's work on WCET analysis. Knowledge of worst case execution times is absolutely essential in many applications. The field is quite mature and Reinhard's work provides an excellent example of a semantics-based approach to this problem. In the body of this paper, we have argued for the equal importance of average case execution times. The need for such analyses is well-recognised by the soft real-time community where average values are much more useful in the optimisation of designs (average power consumption, average heap usage, ...).

We have shown how Probabilistic Abstract Interpretation may be used to support average case analysis; this contrasts with classical abstract interpretation where safety constrains us to work with worst cases. This paper presents first tentative steps in this direction. There are many open problems; one of the most fundamental is how to present operator algebra semantics in a compositional way. This is a necessary development to allow us to construct automatic tools for analysis as alternatives to the model checking approaches discussed in the Introduction.

References

1. Wilhelm, R.: Timing analysis and timing predictability. In de Boer, F., Bonsangue, M., Graf, S., de Roever, W.P., eds.: Formal Methods for Components and Objects. Volume 3657 of Lecture Notes in Computer Science. Springer (2004) 317–323
2. Shaw, A.: Reasoning about time in higher-level language software. IEEE Transactions on Software Engineering **15** (1989) 875–889
3. Lundquist, T., Stenström, P.: Timing anomalies in dynamically scheduled microprocessors. In: 20th IEEE Real-Time Systems Symposium. (1999)
4. Balbo, G.: Introduction to stochastic Petri nets. In: Springer Lectures on Formal Methods and Performance Analysis. Springer (2002) 84–155
5. Bause, F., Kritzinger, P.S.: Stochastic Petri Nets – An Introduction to the Theory. second edn. Vieweg Verlag (2002)
6. Plateau, B., Atif, K.: Stochastic automata network of modeling parallel systems. IEEE Transactions on Software Engineering **17** (1991) 1093–1108
7. Norman, G., Parker, D., Kwiatkowska, M., Shukla, S., Gupta, R.: Formal analysis and validation of continuous time Markov chain based system level power management strategies. In Rosenstiel, W., ed.: Proc. 7th Annual IEEE International Workshop on High Level Design Validation and Test (HLDVT'02), IEEE Computer Society Press (2002) 45–50

8. Di Pierro, A., Wiklicky, H.: Concurrent Constraint Programming: Towards Probabilistic Abstract Interpretation. In Gabbrielli, M., Pfenning, F., eds.: Proceedings of PPDP'00 – Priciples and Practice of Declarative Programming, Montréal, Canada, ACM SIGPLAN, Association of Computing Machinery (2000) 127–138
9. Cousot, P., Cousot, R.: Abstract Interpretation and Applications to Logic Programs. Journal of Logic Programming **13** (1992) 103–180
10. Abramsky, S., Hankin, C., eds.: Abstract Interpretation of Declarative Languages. Ellis-Horwood, Chichester, England (1987)
11. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer Verlag, Berlin – Heidelberg (1999)
12. Di Pierro, A., Wiklicky, H.: Measuring the precision of abstract interpretations. In: Proceedings of LOPSTR'00 – 10th International Workshop on Logic-Based Program Synthesis and Transformation, London, UK. Volume 2042 of Lecture Notes in Computer Science., Berlin – New York, Springer Verlag (2001) 147–164
13. Monniaux, D.: Abstract interpretation of probabilistic semantics. In: Proceedings of SAS'00. Volume 1824 of Lecture Notes in Computer Science., Springer Verlag (2000)
14. Monniaux, D.: Abstract interpretation of probabilistic semantics. In: Seventh International Static Analysis Symposium (SAS'00). Number 1824 in Lecture Notes in Computer Science, Springer Verlag (2000) 322–339
15. Eaton, J.: Gnu Octave Manual. www.octave.org (2002)
16. Cousot, P., Cousot, R.: Systematic Design of Program Analysis Frameworks. In: Symposium on Principles of Programming Languages (POPL), San Antonio, Texas (1979) 269–282
17. Deutsch, F.: Bet Approximation in Inner Product Spaces. Volume 7 of CMS Books in Mathematics. Springer Verlag, New York — Berlin (2001)
18. Ben-Israel, A., Greville, T.: Generalised Inverses — Theory and Applications. second edn. Volume 15 of CMS Books in Mathematics. Springer Verlag, New York — Berlin (2003)
19. Di Pierro, A., Hankin, C., Wiklicky, H.: Probabilistic chemical abstract machine and the expressiveness of linda languages. In de Boer, F., Bonsangue, M., Graf, S., de Roever, W.P., eds.: Proceedings of FMCO 2005, 4th International Symposium on Formal Methods for Components and Object. Volume 4111 of Lecture Notes in Computer Science., Springer Verlag (2006) 388–407
20. Di Pierro, A., Hankin, C., Wiklicky, H.: Quantitative static analysis of distributed systems. Journal of Functional Programming **15** (2005) 1–47