Probabilistic Abstract Interpretation and Statistical Testing

Alessandra Di Pierro[1] and Herbert Wiklicky[2]

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

Abstract. Although generally too weak to guarantee correctness, testing is an indispensable technique for the validation of reactive systems. Abstract Interpretation has been used to overcome some of the problems associated with testing such as the termination of a program run within an acceptable interval of time. We propose the use of Probabilistic Abstract Interpretation for measuring the error made in the evaluation of the reliability of systems via Monte-Carlo testing. The problem we consider is to determine the probability that a reactive system's response falls within a certain set of possible outputs, i.e. that it passes a certain test. As Monte-Carlo testing of the concrete system is not always feasible, tests can be performed instead on an abstract system. This abstraction introduces a further approximation which must be taken into account in the estimation of the error made by testing. The problem is to find out how much the expectation value (or average) of the abstract system differs from the one of the concrete system. We show how to measure such difference by lifting the abstraction to a probabilistic abstract interpretation.

1 Introduction

Software testing is the process of executing a software system to determine whether it matches its specification and executes in its intended environment. It can be used to assess how good software is, or to find faults and thus improve the software. Literature on software testing has appeared since the beginning of computer science but the paper that defined the field and inspired much of today's research dates back to the 1975 [9]. A solid introduction to the subject can be found in [11, 2].

One important phase of the testing process is *test selection* in which test scenarios are selected according to some given criteria (e.g. program statement or branch coverage). While in general test selection can involve quite elaborate considerations, for simple programs like controllers, embedded systems, etc. *statistical testing*, i.e. the random selection of test data is generally regarded as a feasible approach. Various statistical tools can then be adopted to obtain statistical information about the tested system. Among these, Monte-Carlo methods allow us to approximate the *expected value* associated to a test t of some system property by *averages*.

With respect to its final aim testing is very similar to verification [6]. The methodological difference between the two approaches is that with testing the relation between desired and actual behaviour of a system is not established by *formal reasoning* but via *empirical experiments*.

One major problem with testing is that it supposes that the program terminates every time a test is performed (within an acceptable delay). In order to overcome this termination problem the use of Monte-Carlo methods on an abstract system — which is the result of a classical abstract interpretation of the original system — has been proposed in [13]. This kind of *abstract* testing also is an extension of Monte-Carlo methods so as to deal with non-deterministic programs. We will instead concentrate in this paper on a purely probabilistic approach. That means we will select test cases randomly (according to some probabilistic model of the input), observe the probabilistic behaviour of a system and consider a statistical framework for the analysis. The focus of our interest is on deterministic or probabilistic systems and the analysis we obtain corresponds to an *average case analysis*. Although for purely non-deterministic systems it is impossible to obtain an average case analysis we still could simulate a *worst case analysis* within our framework.

The abstract Monte-Carlo testing allows us to obtain an abstract average $\overline{T}$, which gives an estimate for the expectation value, $\mathbf{E}(T)$, that the abstract system passes the (abstract) test T. As usual, standard statistical results guarantee that the estimate $\overline{T}$ of $\mathbf{E}(T)$ will improve the more test runs are performed. For probabilistic programs, the problem, however, remains of how much the expectation value $\mathbf{E}(T)$ or average $\overline{T}$ of the abstract system differs from the expectation value $\mathbf{E}(t)$ or average $\overline{t}$ of the concrete system. In this paper we address this question and show how to solve it by means of *Probabilistic Abstract Interpretation* (PAI) [7, 8].

The essential element of a probabilistic abstract interpretation is the definition of the abstraction function $\boldsymbol{\alpha}$ as a linear operator between two vector spaces representing the concrete and abstract domains of the approximated function. The corresponding concretisation function $\boldsymbol{\gamma}$ is defined by using the notion of Moore-Penrose pseudo-inverse, which shows some important structural similarities with the Galois connection employed in the classical framework. This vector space setting allows for a quantitative reasoning about the "size" of the error introduced by the abstraction. By applying the PAI framework to the abstract Monte-Carlo testing method, we are able to relate $\mathbf{E}(t)$ and $\mathbf{E}(T)$ thus obtaining a method for calculating a numerical estimate of the error made in the approximation of the expectation value of the concrete testing function with the one of its abstraction.

The structure of this paper is as follows: In Section 2 we summarise the main facts about Probabilistic Abstract Interpretation as introduced by the authors in [7]. In Section 3 we dicuss the framework of abstract Monte-Carlo testing [13]. Section 4 contains the main contribution of this paper on the relation between $\mathbf{E}(t)$ and $\mathbf{E}(T)$. Our approach is then illustrated in a very simple example in Section 5.

2 Probabilistic Abstract Interpretation

The basic idea behind abstract interpretation is to analyse a program not in terms of its standard or *concrete semantics*, but rather in terms of an appropriately simplified *abstract semantics*, which only registers aspects of the program which are of relevance with respect to a certain type of analysis (e.g. [5, 1, 15]).

In [7] the authors introduce an abstract interpretation methodology based on a probabilistic semantics instead of the standard order theoretic one. Thus, the semantical domains we use are linear spaces and not cpo's. Another approach to an abstract interpretation based analysis of probabilistic programs is presented in [12], where the domains are defined according to the *random variables* model proposed by Kozen [10].

In our approach, the semantical domains are defined according to a *probabilistic choice* model and correspond to vector spaces of distributions. To keep the presentation simple, we will consider in the following only finite-dimensional spaces with inner product. That corresponds to assuming a finite-dimensional Hilbert space structure. Nevertheless, the framework also applies to infinite-dimensional spaces [8]. Both the abstraction and concretisation function are then linear maps between such semantical domains representing the concrete and abstract domains. The standard way to relate the abstraction and the concretisation function via the notion of Galois connection is replaced by the use of the Moore-Penrose pseudo-inverse.

The main properties of the classical Galois connection can be transferred quite easily into a Hilbert space setting. The condition $\alpha \circ \gamma(d) \sqsubseteq d$ becomes the condition that $\alpha \circ \gamma$ is the orthogonal projection in the image of α, that is $\boldsymbol{\alpha} \circ \boldsymbol{\gamma} = \pi_\alpha$, and $\gamma \circ \alpha(c) \geq c$ turns into $\boldsymbol{\gamma} \circ \boldsymbol{\alpha} = \pi_\gamma$. These new conditions define exactly the so-called *Moore-Penrose pseudo-inverse* [4]:

Definition 1. Let $\mathcal{C}$ and $\mathcal{D}$ be two finite-dimensional vector spaces and $\boldsymbol{\alpha} : \mathcal{C} \mapsto \mathcal{D}$ a linear map between them. A linear map $\boldsymbol{\alpha}^\dagger = \boldsymbol{\gamma} : \mathcal{D} \mapsto \mathcal{C}$ is the (unique) *Moore-Penrose pseudo-inverse* of $\boldsymbol{\alpha}$ iff

1. $\boldsymbol{\alpha} \circ \boldsymbol{\gamma} = \pi_{\boldsymbol{\alpha}}$, and
2. $\boldsymbol{\gamma} \circ \boldsymbol{\alpha} = \pi_{\boldsymbol{\gamma}}$.

The Moore-Penrose pseudo-inverse is unique and always exists. It can be defined for real and complex finite-dimensional vector spaces as well as for infinite-dimensional Hilbert spaces [3], which makes the above-mentioned extension possible.

One way to construct a probabilistic abstract interpretation is via the *lifting* of a classical abstract interpretation: By means of a vector space construction on the cpo's the classical domains are lifted to probabilistic domains and the abstraction function is transformed into a linear map [8]. The general scheme of the construction of a probabilistic abstract interpretation $(\mathcal{A}^\#, \mathcal{B}^\#, f^\#)$ for a semantical function $f : \mathcal{A} \mapsto \mathcal{B}$ is illustrated in Figure 1, where $\boldsymbol{\gamma}$ and $\boldsymbol{\gamma}'$ are the Moore-Penrose pseudo-inverses of the abstraction functions $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}'$, respectively.

$$\begin{tikzcd}
\mathcal{V}(\mathcal{A}) \arrow[r, shift left, "\boldsymbol{\alpha}"] \arrow[d, "f"'] & \mathcal{V}(\mathcal{A}^{\#}) \arrow[l, shift left, "\boldsymbol{\gamma}"] \arrow[d, "f^{\#}"] \\
\mathcal{V}(\mathcal{B}) \arrow[r, shift left, "\boldsymbol{\alpha}'"] & \mathcal{V}(\mathcal{B}^{\#}) \arrow[l, shift left, "\boldsymbol{\gamma}'"]
\end{tikzcd}$$

Fig. 1. Probabilistic Abstract Interpretation of f

One of the important aspect of the Probabilistic Abstract Interpretation (PAI) methodology is the possibility of measuring the precision of the abstraction in numerical terms. In the general case of a generic semantic function $f : \mathcal{A} \mapsto \mathcal{B}$ it seems reasonable to measure the precision of a probabilistic abstract interpretation defined by $(\mathcal{A}^{\#}, \mathcal{B}^{\#}, f^{\#})$ via the norm of the "pseudo-quotient" [8]:

$$\Delta = (\boldsymbol{\alpha}' \circ f) \circ (f^{\#} \circ \boldsymbol{\alpha})^{\dagger}.$$

This corresponds to a *backward* analysis. Analogously, the "pseudo-quotient" corresponding to a *forward* analysis is given by:

$$\nabla = (f^{\#} \circ \boldsymbol{\alpha}) \circ (\boldsymbol{\alpha}' \circ f)^{\dagger}$$

We will show an application of the PAI framework to give a precise estimate of the error made in testing a program via the abstract Monte-Carlo testing method introduced in [13].

3 Monte-Carlo Testing

3.1 Concrete Monte-Carlo Testing

As shown in [13], the testing problem can be formalised as follows. Consider a program c with input/output semantics $[\![c]\!]$, i.e. a map from an *input space* X to an *output space* Z:

$$[\![c]\!] : X \mapsto Z.$$

Contrary to [13], we assume here that the program c is a deterministic or a probabilistic one. The problem is to determine the probability that the output of c falls within a certain set $W \subseteq Z$. We denote this probability by $\mathbf{P}([\![c]\!] \in W)$; it corresponds to the probability that a certain *property* represented by W holds for the (outputs of the) program c — e.g. that some control variable is within a critical interval, or that the computation is correct is some sense.

The probability that the output of c lies within W depends obviously on the probability distribution on the input space. In fact, $\mathbf{P}([\![c]\!] \in W)$ can also be interpreted as the probability that an input x is contained in the set of those inputs which are mapped into W, i.e. that $x \in [\![c]\!]^{-1}(W)$.

The probability that property W holds can be then described as the *measure* of $[\![c]\!]^{-1}(W)$:

$$\mathbf{P}([\![c]\!] \in W) = \mu_X([\![c]\!]^{-1}(W))$$

with respect to the probability distribution μ_X on the input space X.

In order to determine the probability $\mathbf{P}([\![c]\!] \in W)$ we first introduce a *testing function*[1] $t_W : X \mapsto \mathbb{R}$, i.e. a *random variable*, as:

$$t_W(x) = \begin{cases} 1 & \text{iff } [\![c]\!](x) \in W \\ 0 & \text{otherwise.} \end{cases}$$

This allows us to express the probability that W holds as the *expectation value* of the random variable t_W:

$$\mathbf{P}([\![c]\!] \in W) = \mu_X([\![c]\!]^{-1}(W)) = \mathbf{E}(t_W) = \int_X t_W(x)\mu_X.$$

In general it might be rather difficult to calculate the expectation value $\mathbf{E}(t_W)$. An "estimate" of such an integral can be obtained using a Monte-Carlo method. That is we calculate the average of several test runs of c:

$$\overline{t_W} = \frac{\#\text{ of outputs of } c \text{ in W}}{\#\text{ of test runs of } c}.$$

Various statistical results guarantee that the true expectation value $\mathbf{E}(t_W)$ and the estimated average $\overline{t_W}$ get closer and closer the more test runs are performed. We can, for example, exploit the following result [18, 17, p853]:

Theorem 2. *(Chernoff-Hoeffding Bounds) Let $X_1, \ldots, X_n$ be independent random variables with $0 \leq X_k \leq 1$ and expectation value $\mathbf{E}(X_k) = \mu_k$ for $1 \leq k \leq n$. Define*

$$\overline{\mu} = \frac{\sum_{k=1}^n \mu_k}{n} \text{ and } \overline{X} = \frac{\sum_{k=1}^n X_k}{n}.$$

Then for $0 < \varepsilon < 1 - \overline{\mu}$

$$Pr\left(\sqrt{n}(\overline{X} - \mathbf{E}(X)) \geq \varepsilon\right) = \begin{cases} \exp(-2\varepsilon^2) & \text{for } 0 < \overline{\mu} \leq \frac{1}{2} \\ \exp(\frac{-\varepsilon^2}{2\overline{\mu}(1-\overline{\mu})}) & \text{for } \frac{1}{2} < \overline{\mu} \leq 1. \end{cases}$$

This allows us to determine the *error probability* — i.e. the probability that the average $\overline{t_W}$ and the true expectation value $\mathbf{E}(t_W)$ differ by more than ε after n experiments:

$$\mathbf{P}(\mathbf{E}(t_E) \geq \overline{t_W} + \varepsilon) \leq \exp(-2n\varepsilon^2).$$

[1] We will consider in this paper only non-trivial tests, i.e. tests where not all inputs pass the test or fail it.

3.2 Abstract Testing

As stated by Cousot & Cousot [6, Sec II], abstract testing is the verification that the abstract semantics of a program satisfies an abstract specification. Although, experimental testing can in general not replace formal verification, the abstraction allows to solve problems such as the termination of the test runs.

Depending on the testing aims, the abstract system and specification can be obtained via (*i*) classical abstract interpretation, or (*ii*) Probabilistic Abstract Interpretation (PAI).

Constructing the abstract system by classical abstract interpretation necessarily "destroys" the possible probabilistic nature of the concrete system. This means we can only look at a worst-case analysis of the system once we use classical AI to construct the abstract system.

Constructing the abstract system by PAI can only be applied to a concrete system with probabilistic behaviour, but allows for both worst-case and average-case analysis. By imposing an "artificial randomisation" of non-deterministic elements of the concrete system (e.g. utilising a uniform distribution) allows us to abstract also a non-deterministic system. However it is evident that the obtained abstract system should in general only be subject to a worst-case analysis as the average-case analysis is based on artificial and arbritrary probability assumptions made about the concrete system.

3.3 Abstract Monte-Carlo Testing

There is at least one major drawback of the Monte-Carlo method: c might not terminate. In this case, we are unable to estimate the average $\overline{t_W}$. This problem can be overcome by using the abstract Monte-Carlo method approach proposed [13]. Besides the termination problems this approach was proposed also to deal with another problem, namely the testing of heterogenous systems which exhibit both probabilistic and non-deterministic features. Our declared focus is on the average-case analysis of probabilistic systems and we will therefore ignore here the worst-case analysis of heterogenous systems.

The main idea of this approach is to replace the Monte-Carlo testing of a concrete system [13, 2.1] by a Monte-Carlo-testing of an abstracted system [13, 2.2]. In essence, we can abstract the concrete program c, using the classical abstract interpretation framework, and introduce an abstract testing function T_W which is a *safe* abstraction of concrete t_W, i.e. which fulfils:

$$t_W(x) \leq T_W(x)$$

for all $x \in X$, and for which we can guarantee termination.

Based on the Galois connection (α, γ) in Fig 2, the abstract testing function T_W can be defined as

$$T_W(x) = \begin{cases} 1 & \text{iff } (\alpha[\![c]\!])(\alpha(x)) \in \alpha(W) \\ 0 & \text{otherwise.} \end{cases}$$

$$\begin{tikzcd}
X \arrow[r, "\llbracket c \rrbracket"] \arrow[d, "\alpha"'] & W \subseteq Z \arrow[d, "\alpha"] \\
\alpha(X) \arrow[r, "\alpha\llbracket c \rrbracket"] & \alpha(W) \subseteq \alpha(Z)
\end{tikzcd}$$

Fig. 2. Classical Abstract Interpretation of c

By performing Monte-Carlo testing of the abstract system via the abstract testing function T_W, we obtain a Monte-Carlo estimate for $\mathbf{E}(T_W)$ as the average $\overline{T_W}$. Standard results again guarantee that $\mathbf{E}(T_W)$ and $\overline{T_W}$ get closer the more experiments n we perform, i.e.

$$\mathbf{P}(\mathbf{E}(T_E) \geq \overline{T_W} + \varepsilon) \leq \exp(-2n\varepsilon^2).$$

Furthermore, as T_W is a safe approximation of t_w we can conclude that:

$$\mathbf{E}(t) \leq \mathbf{E}(T)$$

and thus that

$$\mathbf{P}(\mathbf{E}(t_W) \geq \overline{T_W} + \varepsilon) \leq \mathbf{P}(\mathbf{E}(T_W) \geq \overline{T_W} + \varepsilon) \leq \exp(-2n\varepsilon^2).$$

4 Relating $\mathrm{E}(t_W)$ and $\mathrm{E}(T_W)$ via Probabilistic Abstract Interpretation

The above relation between $\mathbf{E}(t_W)$ and $\overline{T_W}$ allows us to estimate the error made in evaluating the expected value of the concrete testing function by means of a statistical average of its abstraction. Our aim is to relate $\overline{T_W}$ and $\mathbf{E}(t_W)$ by computing the difference between $\mathbf{E}(t_W)$ and $\mathbf{E}(T_W)$. We will approach this problem by recasting abstract Monte-Carlo testing within the framework of probabilistic abstract interpretation [7, 8].

4.1 Testing as a Linear Map

We lift the test function $t : X \mapsto \mathbb{B}$ for a domain X to a linear operator $\boldsymbol{t} : \mathcal{V}(X) \mapsto \mathcal{V}(\mathbb{B})$ from the free vector space $\mathcal{V}(X)$ on X to the free vector space $\mathcal{V}(\mathbb{B})$ on $\mathbb{B}$ (cf. [7]). We will assume that X is of cardinality $n < \infty$, so as we can restrict our treatment to finite-dimensional Hilbert spaces.

Definition 3. Given a test function $t : X \mapsto \mathbb{B}$ for a domain X of cardinality n, its lifting $\boldsymbol{t} : \mathcal{V}(X) \mapsto \mathcal{V}(\mathbb{B})$ is given by the $n \times 2$-matrix defined by:

$$\boldsymbol{t}_{i1} = \begin{cases} 1 & \text{iff } t(x_i) \in W \\ 0 & \text{otherwise} \end{cases} \quad \text{and } \boldsymbol{t}_{i2} = \begin{cases} 1 & \text{iff } t(x_i) \notin W \\ 0 & \text{otherwise} \end{cases}$$

Our convention is that the first column of these matrices corresponds to *true*. i.e. x_i passes the test, while the second column corresponds to *false*. i.e. x_i fails the test.

The expectation value $\mathbf{E}(t)$ for a test function t with respect to a distribution $\boldsymbol{d} \in \mathcal{V}(X)$ can be computed in a simple way by applying its lifting $\boldsymbol{t}$ to the vector $\boldsymbol{d}$, as shown by the following lemma.

Lemma 4. *Given a test function $t : X \mapsto \mathbb{B}$ for a domain X of cardinality n, its lifting $\boldsymbol{t} : \mathcal{V}(X) \mapsto \mathcal{V}(\mathbb{B})$, and the expectation value $\mathbf{E}_d(t)$ of t with respect to a distribution $\boldsymbol{d} \in \mathcal{V}(X)$, the following holds:*

$$\boldsymbol{t}(\boldsymbol{d}) = (\mathbf{E}_d(t), 1 - \mathbf{E}_d(t)).$$

4.2 The Pseudo-Inverse of Tests

For the special type of probabilistic semantic function $\boldsymbol{t}$, which we can represent as in Definition 3 by a $n \times 2$-matrix, the pseudo-inverse $\boldsymbol{t}^\dagger$ is a $2 \times n$ matrix whose entries are defined in terms of the numbers p and q of the elements which pass and fail the test respectively.

Lemma 5. *Given a test function $t : X \mapsto \mathbb{B}$ for a domain X of cardinality n and its lifting $\boldsymbol{t} : \mathcal{V}(X) \mapsto \mathcal{V}(\mathbb{B})$, the Moore-Penrose pseudo-inverse $\boldsymbol{t}^\dagger : \mathcal{V}(\mathbb{B}) \mapsto \mathcal{V}(X)$ of $\boldsymbol{t}$ is represented by a $2 \times n$-matrix of the form:*

$$\boldsymbol{t}^\dagger = \begin{pmatrix} \frac{1}{p} & 0 & 0 & \dots & \frac{1}{p} & 0 \\ 0 & \frac{1}{q} & \frac{1}{q} & \dots & 0 & \frac{1}{q} \end{pmatrix}$$

where p is the number of elements in X which pass the test t, and q the number of those which fail the test.

The application of the pseudo-inverse of a (lifted) test $\boldsymbol{t}$ to a vector in $\mathcal{V}(\mathbb{B})$ gives a vector in $\mathcal{V}(X)$. There is an important case where we can say something more:

Proposition 6. *Given a test function $t : X \mapsto \mathbb{B}$ for a domain X of cardinality n, its lifting $\boldsymbol{t} : \mathcal{V}(X) \mapsto \mathcal{V}(\mathbb{B})$ and the Moore-Penrose pseudo-inverse $\boldsymbol{t}^\dagger : \mathcal{V}(\mathbb{B}) \mapsto \mathcal{V}(X)$ of $\boldsymbol{t}$, let $\boldsymbol{e_d} = (\mathbf{E}_d(t), 1 - \mathbf{E}_d(t)) \in \mathcal{V}(\mathbb{B})$. Then, for all $\boldsymbol{d} \in \mathcal{V}(X)$ the map $\boldsymbol{t}^\dagger$ maps $\boldsymbol{e_d}$ into the uniform distribution over X, i.e.*

$$\boldsymbol{t}^\dagger(\boldsymbol{e_d}) = (\frac{1}{n}, \frac{1}{n}, \dots, \frac{1}{n}) \in \mathcal{V}(X).$$

Proof. We know from Lemma 5 that the pseudo-inverse in this case is of the form:

$$\boldsymbol{t}^\dagger = \begin{pmatrix} \frac{1}{p} & 0 & 0 & \dots & \frac{1}{p} & 0 \\ 0 & \frac{1}{q} & \frac{1}{q} & \dots & 0 & \frac{1}{q} \end{pmatrix}$$

If we now apply it to the vector $\boldsymbol{e}$ we get a vector in $\mathcal{V}(X)$ with the following coordinates:

$$(\boldsymbol{t}^{\dagger}(\mathbf{E}(t), 1-\mathbf{E}(t)))_i = \begin{cases} \frac{\mathbf{E}(t)}{p} & \text{iff } x_i \text{ passes the test } t \\ \frac{1-\mathbf{E}(t)}{q} & \text{iff } x_i \text{ fails the test } t. \end{cases}$$

Furthermore, from Lemma 5) it follows that $n = p + q$ and that $\mathbf{E}(t) = \frac{p}{n}$, i.e. the number p of x_i which pass the test divided by the total number n of elements in X. We also have for the failing tests $\mathbf{E}(t) = \frac{p}{n} = \frac{n-q}{n} = \frac{n}{n} - \frac{q}{n} = 1 - \frac{q}{n}$, and thus $1 - \mathbf{E}(t) = 1 - 1 + \frac{q}{n} = \frac{q}{n}$. Therefore,

$$(\boldsymbol{t}^{\dagger}(\mathbf{E}(t), 1-\mathbf{E}(t)))_i = \begin{cases} \frac{\mathbf{E}(t)}{p} & = \frac{p}{n}\frac{1}{p} = \frac{1}{n} & \text{iff } x_i \text{ passes the test } t \\ \frac{1-\mathbf{E}(t)}{q} & = \frac{q}{n}\frac{1}{q} = \frac{1}{n} & \text{iff } x_i \text{ fails the test } t. \end{cases}$$

□

4.3 The Pseudo-Quotient

For the special case of a test function $f = t : X \mapsto \mathbb{B}$ and its abstraction $f^{\#} = T : Y \mapsto \mathbb{B}$, the "pseudo-quotient" is represented by a 2×2 matrix, as it is a map from the two-dimensional space $\mathcal{V}(\mathbb{B})$ into itself. Since $\alpha' = \mathbf{Id}$, the expression given in Section 2 for the backward "pseudo-quotient" simplifies to:

$$\Delta = \boldsymbol{t} \circ (\boldsymbol{T} \circ \boldsymbol{\alpha})^{\dagger}$$

Theorem 7. *Given a test function $t : X \mapsto \mathbb{B}$ for a domain X with cardinality n, and an abstract test function $T : Y \mapsto \mathbb{B}$ which is obtained via the abstraction function $\alpha : X \mapsto Y$, let $\boldsymbol{t}$ and $\boldsymbol{T}$ be the lifting of t and T respectively. Then the backward "pseudo-quotient" is of the form:*

$$\Delta = \begin{pmatrix} \frac{\mathbf{E}(t)}{\mathbf{E}(T)} & 1 - \frac{\mathbf{E}(t)}{\mathbf{E}(T)} \\ 0 & 1 \end{pmatrix}$$

where the expectation values $\mathbf{E}(t) = \mathbf{E}(t)_{\boldsymbol{d}}$ and $\mathbf{E}(T) = \mathbf{E}(T)_{\boldsymbol{\alpha}(\boldsymbol{d})}$ are calculated with respect to the uniform distribution $\boldsymbol{d} \in \mathcal{V}(X)$ on X and its abstraction $\boldsymbol{\alpha}(\boldsymbol{d})$ respectively.

Proof. By applying $\boldsymbol{t}$ to $\boldsymbol{d}$ we get, by Lemma 4, the vector:

$$\boldsymbol{t}(\boldsymbol{d}) = (\mathbf{E}(t)_{\boldsymbol{d}}, 1 - \mathbf{E}(t)_{\boldsymbol{d}}).$$

For the abstract test function T, we first have to apply the abstraction α and then perform the test. We get a similar result:

$$\boldsymbol{T}(\boldsymbol{\alpha}(\boldsymbol{d})) = (\mathbf{E}(T)_{\boldsymbol{\alpha}(\boldsymbol{d})}, 1 - \mathbf{E}(T)_{\boldsymbol{\alpha}(\boldsymbol{d})}).$$

For simplicity we will omit the indices from now on, and refer to $\mathbf{E}(t)_{\boldsymbol{d}}$ as $\mathbf{E}(t)$ and to $\mathbf{E}(T)_{\boldsymbol{\alpha}(\boldsymbol{d})}$ as $\mathbf{E}(T)$.

Since T is the lifting of a correct abstraction of t, Δ must be of the form [7]:

$$\begin{pmatrix} \Delta_{11} & \Delta_{12} \\ 0 & 1 \end{pmatrix},$$

i.e. if x_i fails the abstract test it can never have passed the concrete one, i.e. $\Delta_{21} = 0$; and all abstractly failing x_i also failed the concrete test, i.e. $\Delta_{22} = 1$. The entries Δ_{11} and Δ_{12} can be determined by observing that for the uniform distribution d we have that:

$$\Delta(\boldsymbol{T}(\boldsymbol{\alpha}(\boldsymbol{d}))) = \boldsymbol{t}(\boldsymbol{d}).$$

In fact, $\Delta(\boldsymbol{T}(\boldsymbol{\alpha}(\boldsymbol{d}))) = \boldsymbol{t} \circ (\boldsymbol{T} \circ \boldsymbol{\alpha})^\dagger (\boldsymbol{T} \circ \boldsymbol{\alpha})(\boldsymbol{d})$, where $(\boldsymbol{T} \circ \boldsymbol{\alpha})^\dagger (\boldsymbol{T} \circ \boldsymbol{\alpha})(\boldsymbol{d}) = \boldsymbol{d}$, because $\boldsymbol{d}$ is the uniform distribution (cf. Proposition 6).

The explicit representation of the transformation $\Delta(\boldsymbol{T}(\boldsymbol{\alpha}(\boldsymbol{d})))$ gives the following vector:

$$(\mathbf{E}(T), 1 - \mathbf{E}(T)) \cdot \begin{pmatrix} \Delta_{11} & \Delta_{12} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \Delta_{11} \cdot \mathbf{E}(T) \\ \Delta_{12} \cdot \mathbf{E}(T) + (1 - \mathbf{E}(T)) \end{pmatrix}$$

Since the result of this transformation must be the vector $(\mathbf{E}(t), 1 - \mathbf{E}(t))$, we get the equation:

$$\begin{pmatrix} \Delta_{11} \cdot \mathbf{E}(T) \\ 1 - (1 - \Delta_{12}) \cdot \mathbf{E}(T) \end{pmatrix} = \begin{pmatrix} \mathbf{E}(t) \\ 1 - \mathbf{E}(t) \end{pmatrix},$$

from which we get:

$$\Delta_{11} = \frac{\mathbf{E}(t)}{\mathbf{E}(T)} \text{ and } \Delta_{12} = 1 - \frac{\mathbf{E}(t)}{\mathbf{E}(T)}$$

□

Combing this with statistical results (like Theorem 2) we can relate $\overline{T_W}$ and $\mathbf{E}(t_W)$. In fact, if we perform more and more experiments the difference between $\mathbf{E}(T_W)$ and $\overline{T_W}$ will tend to zero, so as we can approximate $\mathbf{E}(t_W)$ with $\overline{T_W} \cdot \Delta_{11}$.

4.4 Norm and Precision

Theorem 7 allows us to estimate the "stretching" factor, or ratio between $\mathbf{E}(t)$ and $\mathbf{E}(T)$ via the entries in the pseudo-quotient Δ. This defines the precision of the probabilistic abstract interpretation for the testing function t. In [8] the precision of a general probabilistic abstract interpretation is defined via the norm of the pseudo-quotient, and in particular via the 1-norm defined for a matrix $\mathbf{Q}$ by:

$$\|\mathbf{Q}\|_1 = \sup_{\|x\|_1 = 1} \|\mathbf{Q}x\|_1.$$

The 1-norm of a matrix is the same as the maximal 1-norm of one of its column vectors $(\mathbf{Q})_i$, i.e. $\|\mathbf{Q}\|_1 = \sup \|(\mathbf{Q})_i\|_1$ where the 1-norm of a vector is defined as sum of the absolute values of the coefficients of the vector:

$$\|(q_i)_i\|_1 = \sum_i |q_i|.$$

In our particular case, the pseudo-quotient, Δ, comparing concrete and abstract statistical testing is of a very simple form, namely:

$$\Delta = \begin{pmatrix} q & 1-q \\ 0 & 1 \end{pmatrix},$$

with $q = \frac{\mathbf{E}(t)}{\mathbf{E}(T)}$ and $0 \leq q \leq 1$. Thus,

$$\|\Delta\|_1 = 2 - \frac{\mathbf{E}(t)}{\mathbf{E}(T)}.$$

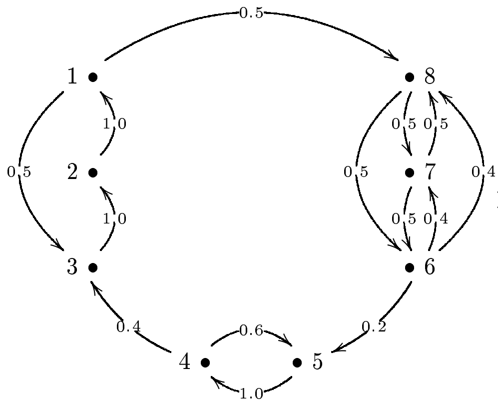
As discussed in [8], this number gives a measure of the precision of the abstraction T of the testing function t. Moreover, it is immediate to see that if $\|\Delta\|_1 = 1$, that is the abstraction is complete [8], then $\frac{\mathbf{E}(t)}{\mathbf{E}(T)} = 1$, or in other words, the two expectation values are the same. Analogously, one can see that if $\|\Delta\|_1 > 1$, that is the abstraction is correct [8], then $0 \leq \frac{\mathbf{E}(t)}{\mathbf{E}(T)} < 1$ which implies $\mathbf{E}(t) < \mathbf{E}(T)$.

5 An Example Application

Statistical testing is a useful tool when it comes to the quality assessment of various heuristic algorithms. We have investigated this previously in the context of the so-called "proof-by-nine" method for checking the correctness of simple multiplications [8]. In the following we will look instead at a different area: Statistical testing is of particular importance when it comes to computer systems interacting with the physical world as in the case of reactive systems, e.g. controllers, embedded systems, etc. Typically these systems interact with some physical system in order to control, supervise, forecast or observe it.

Forecasting Models. Let us consider for example a weather (economical, etc.) forecasting system. The concrete system is a highly complex program running maybe on the largest computers built. The task is to determine the probability that its forecasts are correct.

The first problem is, that "correctness" of meteorological forecasts is not formally specified, as we do not have a perfect or complete theory of the weather. At best one can compare the forecast with the actually observed weather. It is therefore impossible to use some formal methods to asses the quality of the forecasts, instead we are effectively forced to consider statistical testing in order to determine the accuracy of the forecast.



$$\mathbf{M} = \begin{pmatrix} 0.0 & 0.0 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 & 0.5 \\ 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.4 & 0.0 & 0.6 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.2 & 0.0 & 0.4 & 0.4 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.5 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.5 & 0.5 & 0.0 \end{pmatrix}$$

Fig. 3. A Forecasting Model

A second problem is that all the data describing the forecast and the actual weather situation is difficult to obtain. The idea is thus straight forward not to test if all the millions of forecasted temperatures, pressures, etc. coincide with the real situation, but to see if the system is correct on the level of — let's say — a newspaper forecasts. In order to check if the meteorological model is sound one may try to look at an abstracted system where complex meteorological data is aggregated to simple statements like "sunny", "cloudy", "70% chance of rain", etc. It is then possible to compare the simplified forecasts with the actual situation and find out how precise the forecasts of the simplified system are.

A Concrete Forecasting Model. More specifically let us look at the following scenario: Some metrological institute produces a weather forecasting model and claims a 90% success rate. A customer which would like to buy this forecasting system now faces the problem to check if this claim is correct. Due to various limitations the customer is unable consider all the forecast data produced by the model and check if they coincide with the actual measurements. Instead the customer has to base the assessment of the system quality on tests involving only some aggregated or abstracted data. Suppose that on this abstract level the forecasts are correct with some probability: What does this tell us about the quality of the concrete weather model?

First, let us look at a concrete forecasting system: Models for many real world systems — for example metrological models like global circulation model e.g. [16] — are often described by specifying how different components of the system interact with or influence other parts. In order to illustrate our approach, let us look at an artificial example of such a system description is depicted in Figure 3. For our purposes it is irrelevant what interpretation one may have for the different elements or states of the system: they may describe the circulation of goods, services and money in an economical model, or the energy balance in different components of a climatic model.

$$\mathbf{W} = \begin{pmatrix} 0.0 & 0.2 & 0.3 & 0.0 & 0.0 & 0.0 & 0.0 & 0.5 \\ 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.4 & 0.0 & 0.6 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.4 & 0.0 & 0.3 & 0.3 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.5 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.5 & 0.5 & 0.0 \end{pmatrix} \qquad \boldsymbol{t} = \begin{pmatrix} 0.6 & 0.4 \\ 1.0 & 0.0 \\ 1.0 & 0.0 \\ 1.0 & 0.0 \\ 1.0 & 0.0 \\ 0.6 & 0.4 \\ 1.0 & 0.0 \\ 1.0 & 0.0 \end{pmatrix}$$

Fig. 4. Underlying Dynamics and Concrete Test

The system in Figure 3 can be seen as the description of a stochastic system (the sum of all weights or probabilities associated to each state add up to one) specifying the probabilities that "rain" is followed by "sunshine". We can represent the model by the stochastic matrix $\mathbf{M}$ in Figure 3.

Concrete Testing. Neither the developer of the model $\mathbf{M}$ nor the customer do know the true dynamics of the system, e.g. the weather or climate. However, the developer can test how often the forecast coincides with the observed weather.

To analyse this situation more detailed: Let us assume that the *true* weather dynamics is described by the matrix $\mathbf{W}$ in Figure 4. This true weather model is unknown to the developer of the model $\mathbf{M}$, it is only possible to test $\mathbf{M}$ against the true weather dynamics $\mathbf{W}$, i.e. see if the forecast of $\mathbf{M}$ coincides with what is actually happening in $\mathbf{W}$. In our case we see, for example, that the forecasts for state 2 are always correct: For both matrices $\mathbf{M}$ and $\mathbf{W}$ the second row is the same; for state 1 the forecasts will however be wrong with a 40% chance: The relation between state 1 and states 2 and 3 are each wrong by 20%.

If we therefore test how often the forecasts produced by the model $\mathbf{M}$ coincides with the true behaviour of the weather $\mathbf{W}$ we obtain the concrete test t depicted in Figure 4. The matrix $\boldsymbol{t}$ records for each of the eight states the probability that the model's forecasts are correct (first column) or wrong (second column). We can also compute from $\boldsymbol{t}$ that the overall success rate is 90% (if we assume that each state appears in reality with the same probability).

If we know the true weather dynamics we can easily compute $\boldsymbol{t}$ from $\mathbf{M}$ and $\mathbf{W}$. But in reality the developer will not know $\mathbf{W}$; instead the entries in $\boldsymbol{t}$ have to be estimated, e.g. by Monte Carlo testing. If the number of tests is sufficiently large the usual statistical results (e.g. central limit theorem) guarantee that the observed averages $\bar{t}$ get as close as necessary to the theoretical values in $\boldsymbol{t}$.

Abstract Testing. The potential customer now attempts to test the quality of the model using various abstract tests. These abstract tests could be based on a simplified, abstract description the system. One may try, for example, to take nodes 1, 2 and 3 to form an abstract entity A, the same for nodes 4 and 5 to form an abstract state B, and finally 6, 7 and 8 as group C. In this way we get a simplified forecasting model. There are also other aggregations possible, e.g. 1, 3 and 7 form node A, 2 and 4 state B, and the remaining concrete states are abstracted by an abstract state C. These abstractions from $\{1, \dots, 8\}$ unto $\{A, B, C\}$ are defined by the following matrices:

$$\boldsymbol{\alpha}_1 = \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix} \quad \boldsymbol{\alpha}_2 = \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 1.0 \\ 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix}$$

When the customer now observes the actual weather and compares it with the abstract forecasts, it is possible to assess the *abstract* quality of the forecast for each of the three abstract states. For the two abstractions above the customer would get — provided the tests are repeated often enough — the following sets test results (representing the expectation values for successful abstract forecasts):

$$\mathbf{T}_1 = \begin{pmatrix} 1.00 & 0.00 \\ 1.00 & 0.00 \\ 0.87 & 0.13 \end{pmatrix} \quad \mathbf{T}_2 = \begin{pmatrix} 0.87 & 0.13 \\ 1.00 & 0.00 \\ 0.80 & 0.20 \end{pmatrix}$$

The customer can also estimate the expectation value for the overall success: $\mathbf{E}(T_1) = 0.828$ for the first and $\mathbf{E}(T_2) = 0.872$ for the second abstraction using this framework for abstract statistical tests.

Note that the expectation values $\mathbf{E}(T_1)$ and $\mathbf{E}(T_2)$ are not directly computable from T_1 and T_2. The abstract tests are not based on the uniform distribution on the abstract states A, B and C (each probability $\frac{1}{3}$). The abstract tester cannot chose the input test data but is only allowed to observe the actual weather. The relevant probability distribution on the abstract states is therefore the one "inherited" from the concrete system.

Relating Abstract and Concrete Quality. The quality of the abstract model is (in general) different from the quality of the concrete model. It can be better or worse, and — as in the above examples — we see that it depends on the abstraction used. To take an extreme case: If the abstraction $\boldsymbol{\alpha}$ aggregates all concrete states to a single abstract state — this would correspond to an abstract test which only determines if there exists a model — the test would always succeed, i.e. $\mathbf{E}(\mathbf{T}) = 1$ independently of how the quality of the concrete model.

Applying our testing framework now allows us to relate the abstract model quality with the concrete one. To do this we have to look at the pseudo-quotients:

$$\Delta_1 = \boldsymbol{t}_1 \circ (\boldsymbol{T}_1 \circ \boldsymbol{\alpha})^\dagger = \begin{pmatrix} 0.92 & 0.08 \\ 0.52 & 0.48 \end{pmatrix}, \ \Delta_1 = \boldsymbol{t}_2 \circ (\boldsymbol{T}_2 \circ \boldsymbol{\alpha})^\dagger = \begin{pmatrix} 0.97 & 0.03 \\ 0.05 & 0.95 \end{pmatrix}.$$

With this data we now can determine the quality of the concrete model based on the observations of the abstract model via $\mathbf{E}(\mathbf{T}) \cdot \Delta_{11} = \mathbf{E}(t)$. In our case we indeed see that the the claim of 90% accuracy is supported by the two abstract tests we considered.

A Quality Testing Scenario. The model developer could try to optimise the quality of the system with respect to the abstract tests; for example by concentrating on forecasting aspects the customer will base its abstract tests on. If the developer then publishes misleading or false concrete testing results $\boldsymbol{t}$ the customer could assume that the performance of the system is better than it actually is.

To avoid this the potential customer can keep the set of abstract properties used in the abstract tests secret from the developer of the concrete system. Provided the developer of the model publishes a detailed testing protocol $\boldsymbol{t}$ the customer can decide independently on a certain abstract testing regime, i.e. fix some $\boldsymbol{\alpha}$. The customer can then test the abstract system in order to obtain an estimate for the abstract test $\mathbf{T}$. All this allows the customer to compute Δ and check if the success rate claimed by the developer is indeed correct. This assessment of the quality $\mathbf{E}(t)$ of the concrete system by the customer depends only on the quality of the estimation of the abstract test results $\mathbf{T}$ via $\overline{\mathbf{T}}$.

The customer does not need to understand or inspect the system $\mathbf{M}$ (or even $\mathbf{W}$) and can use a substantially simplified statistical test. The main work, i.e. testing the concrete system, has to be done by the system developer, the customer only has to check if the quality pass rates claimed by the developer are consistent with the results obtained by the much simpler abstract tests. This methodology can be seen as a kind of statistical version of so called "proof carrying code" [14].

6 Conclusions

We have proposed the use of probabilistic abstract interpretation [7, 8] for measuring the error made in the evaluation of the reliability of a system by statistical testing of an abstraction of the system. The instantiation of the framework of probabilistic abstract interpretation to the case of semantical functions which map concrete and abstract values into the booleans $\mathbb{B}$, such as testing functions, not only accommodates the abstract Monte-Carlo method proposed in [13] for the analysis of probabilistic programs, but additionally allows for a *quantitative* analysis of the relation between the expected values, $\mathbf{E}(t)$ and $\mathbf{E}(T)$, of the concrete and the abstract testing functions. The fact that we deal only with 2×2-matrices allowed us to compare $\mathbf{E}(t)$ and $\mathbf{E}(T)$ in a more direct way with respect to the method introduced in [8], which calculates a suitable norm of the pseudo-quotient. Furthermore, we discussed a scenario which allows a user with limited testing capabilities to check if claims a software developer makes about the quality of a system are correct.

References

1. S. Abramsky and C. Hankin, editors. *Abstract Interpretation of Declarative Languages.* Ellis-Horwood, Chichester, England, 1987.
2. Boris Beizer. *Software testing techniques.* Van Nostrand Reinhold, New York, 1990.
3. Frederick J. Beutler. The operator theory of the pseudo-inverse. *Journal of Mathematical Analysis and Applications*, 10:451–470,471–493, 1965.
4. Stephen L. Campbell and D. Meyer. *Generalized Inverse of Linear Transformations.* Constable and Company, London, 1979.
5. Patrick Cousot and Radhia Cousot. Abstract Interpretation and Applications to Logic Programs. *Journal of Logic Programming*, 13(2-3):103–180, July 1992.
6. Patrick Cousot and Radhia Cousot. Abstract interpretation based program testing. In *Proceedings of SSRGG 2000 Computer & eBusiness International Conference*, l'Aquila, July 31 – August 6 2000. Scuola Superiore G. Reiss Romoli. CD ROM paper 248.
7. Alessandra Di Pierro and Herbert Wiklicky. Concurrent Constraint Programming: Towards Probabilistic Abstract Interpretation. In M. Gabbrielli and F. Pfenning, editors, *Proceedings of PPDP'00 – Priciples and Practice of Declarative Programming*, pages 127–138, Monteéal, Canada, September 2000. ACM SIGPLAN, Association of Computing Machinery.
8. Alessandra Di Pierro and Herbert Wiklicky. Measuring the precision of abstract interpretations. In *Proceedings of LOPSTR'00 – 10th International Workshop on Logic-Based Program Synthesis and Transformation, London, UK*, volume 2042 of *Lecture Notes in Computer Science*, pages 147–164, Berlin – New York, 2001. Springer Verlag.
9. John B. Goodenough and Susan L. Gerahart. Toward a theory of test data selection. *IEEE Trans. Software Eng.*, 2(2):156–173, 1975.
10. Dexter Kozen. Semantics for probabilistic programs. *Journal of Computer and System Sciences*, 22:328–350, 1981.
11. Brian Marick. *The Craft of Software Testing.* Prentice Hall, 1995.
12. David Monniaux. Abstract interpretation of probabilistic semantics. In *Proceedings of SAS'00 — Seventh International Static Analysis Symposium*, volume 1824 of *Lecture Notes in Computer Science*. Springer Verlag, 2000.
13. David Monniaux. An abstract monte-carlo method for the analysis of probabilistic programs. In *Symposium on Principles of Programming Languages (POPL)*, pages 93–101, London, 2001. ACM Press.
14. George C. Necula. Proof-carrying code. In *Symposium on Principles of Programming Languages (POPL)*, pages 106–119, Paris, 1997. ACM Press.
15. Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Aanalysis.* Springer Verlag, Berlin – Heidelberg, 1999.
16. Peter J. Robinson and Ann Henderson-Sellers. *Contemporary Climatology.* Addison Wesley Longman, second edition, 1999.
17. Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, pages 331–340, 1993.
18. Galen R. Shorack and Jon A. Wellner. *Empirical Processes with Applications to Statistics.* John Wiley & Sons, Chicester, 1986.