Linear algebra techniques for deciding the correctness of probabilistic programs with bounded resources *

Amílcar Sernadas , Jaime Ramos, and Paulo Mateus
SQIG - Instituto de Telecomunicações, Department of Mathematics, IST, TU Lisbon, Portugal

Abstract

An algorithm is outlined for deciding the correctness of (space and time) resource bounded, imperative, probabilistic programs, using linear algebra techniques encoded in the theory of real closed fields. A calculus suitable for reasoning by hand is derived from the proposed encoding. The approach is feasible also for classical, non deterministic, and quantum programs.

1 Introduction

Our goal is to present an algorithm for deciding the total correctness of probabilistic programs with bounded resources. To this end we will use linear algebra techniques encoded in the theory of real closed fields. The key idea is to consider a probabilistic program to be a linear transformation over a suitable vector space. A decision algorithm for real closed fields may then be used ([10, 1]). Tarski's original algorithm is rather inefficient, but recent developments motivated by applications in Robotics have improved it significantly. The encoding is also expensive, given the exponential nature of the state space as a function of the number of memory registers.

Herein, we consider bounded space and time resources. By bounded space, we mean that we only allow for a finite number of program registers, each of these taking values in a finite data domain. By bounded time, we mean that a loop will only perform a finite number of iterations. These limitations are realistic in many applications and are essential to using linear algebra techniques as proposed herein.

As usual, we consider a *proper classical state* to be a valuation on program registers. Observe that the set of proper classical states is finite because we have a finite number of registers and each of these registers ranges over a finite domain. We also consider an *improper state* for dealing with aborting computations. A *probabilistic state* is a probability distribution on such classical states. A key novelty here is to look at a probability state as a *vector* in a linear space where classical states constitute a basis.

In this setting, a program becomes a linear transformation mapping a vector (a state) into a vector (a state) that results from the execution of that program. A (state dependent) guard becomes a projector, mapping a vector into a vector containing only the valuations that satisfy that guard. Finally, a (state dependent) expression becomes an affine map, mapping a proper basis vector onto the value of the expression at that valuation.

Hence, we can reduce reasoning about bounded resources probabilistic programs to reasoning about linear transformations and real closed fields, which are known to be decidable. Looking at computation states as vectors is natural within the field of quantum computation, but it is also a valid approach in the case of probabilistic programs with bounded resources as shown in this paper. In fact, it is also applicable to non deterministic programs. The idea of applying linear algebra techniques outside the field of quantum computation was inspired by the exogenous approach to enriching logics [7, 5] where an encoding in the theory of real closed fields was also exploited. It should also be mentioned the work on probabilistic logic [2] that also capitalized on that decidable theory, as well as Nilson's proposal [8] who already advocated using linear algebra for probabilistic reasoning. The use of linear algebra techniques in the semantics of probabilistic programs was first proposed in [3, 4].

*This work was partially supported by FCT and FEDER, namely via the QuantLog POCI/MAT/55796/2004 (Quantum Logic), KLog PTDC/MAT/68723/2006 (Kleistic Logic) and QSec PTDC/EIA/67661/2006 (Quantum Security) projects.

2 Probabilistic programs as linear transformations

Syntax Assume fixed once and for all $M \in \mathbb{N}^+$, that bounds the *address space* $\mathbb{M} = \{1,\ldots,M\}$ of *program registers* $\mathsf{R} = \{\mathsf{r}_m : m \in \mathbb{M}\}$. Assume also fixed once and for all $K \in \mathbb{N}^+$, the bound on the *data domain* $\mathbb{K} = \{-K,\ldots,-1,0,1,\ldots,K\}$ and on the *number of iterations* of each loop. Program registers are able to store a value in $\mathbb{K}$. The programming language is depicted below using BNF notation.

- $\eta := \mathsf{r}_m \,[\!]\, x_i \,[\!]\, n \,[\!]\, (\eta + \eta) \,[\!]\, (\eta\,\eta)$
- $\gamma := (\eta \leq \eta) \,[\!]\, (\neg\,\gamma) \,[\!]\, (\gamma \wedge \gamma)$
- $\theta := \mathsf{skip} \,[\!]\, \mathsf{abort} \,[\!]\, (\gamma!)(\mathsf{r}_m \leftarrow \eta) \,[\!]\, (\theta;\theta) \,[\!]\, (\theta\, n|n\,\theta) \,[\!]\, (\mathsf{if}\ \gamma\ \mathsf{then}\ \theta\ \mathsf{else}\ \theta) \,[\!]\, (\mathsf{while}\ \gamma\ \mathsf{do}\ \theta)$

Expressions, $\eta, \eta_1, \ldots$, are built from program registers, logical variables and *numerals* using addition and multiplication. The computation of the value of an expression succeeds only if that value is in $\mathbb{K}$. Otherwise the computation aborts. Guards, $\gamma, \gamma_1, \ldots$, are built up from comparison between expressions using propositional connectives like $\neg$ and $\wedge$. As usual, other connectives (such as $\Rightarrow, \vee, \ldots$) can be introduced as abbreviations. The computation of a guard aborts if any of its subcomputations aborts.

Programs, $\theta, \theta_1, \ldots$ are built as expected. Statement $(\gamma!)$ is an abortive test: if the computation of γ succeeds to true then the program succeeds otherwise it aborts. Statement $\mathsf{r}_m \leftarrow \eta$ is the assignment of value denoted by η to program register r_m. Such an assignment aborts whenever the computation of η aborts. Statement $(\theta_1\, n_1|n_2\, \theta_2)$ is the probabilistic choice, i.e. θ_1 is executed with probability $\frac{n_1}{n_1+n_2}$ and θ_2 is executed with probability $\frac{n_2}{n_1+n_2}$. Such a choice aborts whenever n_1 or n_2 is outside of $\mathbb{K}$. It is a simple exercise to extend binary probabilistic choice to any number of alternatives. Note that in the case of the while the number of iterations is bounded by K. If the guard is still true after K iterations then the computation aborts. In all cases a program aborts if any of its components aborts.

Semantics Programs are linear transformations over the real vector space generated by the classical states. The set of *proper classical states* or *valuations* is $\mathbb{K}^{\mathbb{M}}$. We consider the additional *improper state* $\star$ in order to represents the state resulting from an aborted computation. We will denote by $\mathscr{B}$ the set $\mathbb{K}^{\mathbb{M}} \cup \{\star\}$, the computational basis of the vector space $\mathscr{V}$. So, in general, a vector written in Dirac notation is of the form $\sum_{v \in \mathbb{K}^{\mathbb{M}}} \xi_v |v\rangle + \xi_\star |\star\rangle$.

Expressions correspond to *affine maps* $\mathscr{V} \to \mathbb{R}$ as depicted below. Observe that, as desired, $(\mathsf{AM}\,\eta)|v\rangle$ returns the value of expression η given by valuation v. Extending this to an arbitrary vector $|\psi\rangle$ yields the sum of the value of η at each of the valuations weighted by the corresponding vector component.

- $\mathsf{AM}\,\mathsf{r}_m$ for $\sum_{v \in \mathbb{K}^{\mathbb{M}}} v(m)\langle v|$
- $\mathsf{AM}\,x_i$ for x_i
- $\mathsf{AM}\,n$ for n for $n \in \mathbb{Z}$
- $\mathsf{AM}(\eta_1 + \eta_2)$ for $(\mathsf{AM}\,\eta_1) + (\mathsf{AM}\,\eta_2)$

Guards correspond to *projectors* $\mathscr{V} \to \mathscr{V}$ as depicted below. We also need *proper projectors* that return the proper component satisfying the guard.

- $\mathsf{Pr}\,\gamma$ for $\mathsf{VP}\,\gamma + |\star\rangle\langle\star|$
- $\mathsf{VP}(\eta_1 \leq \eta_2)$ for $\sum_{v \in \mathbb{K}^{\mathbb{M}}} \mathsf{leq}((\mathsf{AM}\,\eta_1)|v\rangle, (\mathsf{AM}\,\eta_2)|v\rangle)|v\rangle\langle v|$
- $\mathsf{VP}(\neg\,\gamma)$ for $\mathsf{V} - \mathsf{VP}\,\gamma$
- $\mathsf{VP}(\gamma_1 \wedge \gamma_2)$ for $(\mathsf{VP}\,\gamma_1) \circ (\mathsf{VP}\,\gamma_2)$

With this semantics we impose that the improper valuation satisfies everything. The projector of $\eta_1 \leq \eta_2$ acts on each proper valuation v as follows: expression $\mathsf{leq}((\mathsf{AM}\,\eta_1)|v\rangle, (\mathsf{AM}\,\eta_2)|v\rangle)$ yields either the value 1, if $(\mathsf{AM}\,\eta_1)|v\rangle \leq (\mathsf{AM}\,\eta_2)|v\rangle$, or 0 otherwise. In the first case, the valuation will be kept in the resulting vector, otherwise, it will be dropped from the resulting vector. Observe that, for a proper valuation v, $(\mathsf{AM}\,\eta_1)|v\rangle$ is the value of expression η_1 in that valuation.

Finally, we present the linear transformation semantics of our probabilistic programs.

- $\mathsf{LT}\,\mathsf{skip}$ for I
- $\mathsf{LT}\,\mathsf{abort}$ for A

- $\mathsf{LT}(\gamma!)$ for $\mathsf{VP}(\gamma\wedge\gamma\downarrow)+\mathsf{A}\circ(\mathsf{VP}(\neg(\gamma\wedge\gamma\downarrow)))$
- $\mathsf{LT}(\mathsf{r}_m\leftarrow\eta)$ for $\left(\sum_{v,v'\in\mathbb{K}^{\mathbb{M}}}\mathsf{asg}_m(|v\rangle,|v'\rangle,(\mathsf{AM}\,\eta)|v\rangle)\,|v'\rangle\langle v|\right)\circ(\mathsf{VP}\,\eta\downarrow)+\mathsf{A}\circ(\mathsf{VP}(\neg(\eta\downarrow)))$
- $\mathsf{LT}(\theta_1;\theta_2)$ for $(\mathsf{LT}\,\theta_2)\circ(\mathsf{LT}\,\theta_1)$
- $\mathsf{LT}(\theta_1\,n_1|n_2\,\theta_2)$ for $\frac{1}{n_1+n_2}(n_1\,\mathsf{LT}\,\theta_1+n_2\,\mathsf{LT}\,\theta_2)\circ\mathsf{VP}(n_1>0\wedge n_2>0\wedge(n_1+n_2)\downarrow)+\mathsf{A}\circ\mathsf{Pr}(\neg(n_1>0\wedge n_2>0\wedge(n_1+n_2)\downarrow))$
- $\mathsf{LT}(\mathsf{if}\,\gamma\,\mathsf{then}\,\theta_1\,\mathsf{else}\,\theta_2)$ for $((\mathsf{LT}\,\theta_1)\circ(\mathsf{VP}\,\gamma)+(\mathsf{LT}\,\theta_2)\circ(\mathsf{VP}(\neg\gamma)))\circ(\mathsf{VP}(\gamma\downarrow))+\mathsf{A}\circ(\mathsf{VP}(\neg(\gamma\downarrow)))$
- $\mathsf{LT}(\gamma?)$ for $\mathsf{VP}(\gamma\wedge\gamma\downarrow)$

We briefly explain some of these linear transformations. The linear transformation for abort is A, the linear transformation that maps every vector to vector $|\star\rangle$. The linear transformation for $\gamma!$ behaves like identity on the proper components that satisfy γ, aborts for the other components, and preserves the improper valuation. Note that we are also taking into account the possibility of the computation of guard $(\gamma\downarrow)$ aborting. The linear transformation for probabilistic choice $\theta_1\,n_1|n_2\,\theta_2$ is as expected: if both n_1 and n_2 are greater than 0 and their sum is within bounds then the resulting state is the weighted sum of applying the linear transformations for θ_1 and θ_2 to the original state. Otherwise, the execution aborts.

We only consider time bounded loops, hence $(\mathsf{while}\,\gamma\,\mathsf{do}\,\theta)$ stands for $(\mathsf{if}\,\gamma\,\mathsf{then}\,\theta)^K;((\neg\gamma)!)$. Recall, that as test $(\neg\gamma)!$ is abortive if after K iterations the guard still succeeds to true then the loop aborts.

3 Asserting properties of probabilistic programs

We now focus on the language of assertions about probabilistic programs. We will be interested in reasoning about terms of the form $(\int\sigma)$, where σ is a state dependent formula. Intuitively, such a term denotes the probability of the formula σ being true at a given state. The envisaged language is:

- $\delta := \xi \,[\!]\, (\int\sigma) \,[\!]\, (\oint\sigma) \,[\!]\, (\delta+\delta) \,[\!]\, (\delta\delta) \,[\!]\, (\delta\circ\overline{\theta})$
- $\sigma := \gamma \,[\!]\, (\eta\downarrow) \,[\!]\, (\gamma\downarrow) \,[\!]\, (\neg\sigma) \,[\!]\, (\sigma\wedge\sigma)$
- $\overline{\sigma} := \sigma \,[\!]\, (\delta\leq\delta) \,[\!]\, (\neg\overline{\sigma}) \,[\!]\, (\overline{\sigma}\wedge\overline{\sigma}) \,[\!]\, (\forall\overline{\sigma})$
- $\alpha := (\xi\leq\xi) \,[\!]\, (|\psi\rangle\mathsf{adm}) \,[\!]\, (|\psi\rangle\,\Box\,\overline{\sigma}) \,[\!]\, (\Box\,\overline{\sigma}) \,[\!]\, (\neg\alpha) \,[\!]\, (\alpha\wedge\alpha) \,[\!]\, (\forall x_i\,\alpha)$

State terms, $\delta,\delta_1,\ldots$, include program expressions, mass terms $\int\sigma$ and $\oint\sigma$, and are closed for addition, pointwise multiplication and transformation by a program. Term $\int\sigma$ refers to the (total) mass of state formula σ in a given state, that is, the sum of the coefficients of all valuations satisfying σ (the *probability* of σ in an admissible state). This mass includes the improper state. Term $\oint\sigma$ refers to the proper mass of state formula σ, i.e the mass of σ at a given state not considering the improper state. Term $\delta\circ\overline{\theta}$ denotes the value of term δ after the execution of $\overline{\theta}$.

State formulas, $\sigma,\sigma_1,\ldots$, include program guards and other atomic formulas and are closed for propositional connectives. Formula $(\eta\downarrow)$ expresses that the computation of expression η terminates in a given state. Similarly for $(\gamma\downarrow)$.

Extended state formulas, $\overline{\sigma},\overline{\sigma}_1,\ldots$, include state formulas and comparison between state terms. Observe that, with this syntax, we do not allow for nested assertions about mass, and proper mass, terms. The extended state formula $(\forall\overline{\sigma})$ expresses that $\overline{\sigma}$ holds in every state. As usual, other formulas can be obtained as abbreviations. For instance, $\delta_1=\delta_2$ stands for $\delta_1\leq\delta_2\wedge\delta_2\leq\delta_1$.

Formulas, $\alpha,\alpha_1,\ldots$, are built from a set of atomic formulas described below and using the usual first order constructions. Note that these new formulas are reducible to formulas over the theory of real closed fields. Furthermore, we have opted not to present here the language for real closed fields, vectors and linear transformations to keep the presentation simpler without compromising readability.

Atomic formula $\xi_1\leq\xi_2$ stands for comparison between two real numbers ξ_1 and ξ_2. Formula $\mathsf{adm}|\psi\rangle$ expresses that vector $|\psi\rangle$ is admissible, i.e. denotes a probabilistic state. Formula $|\psi\rangle\,\Box\,\overline{\sigma}$ stands for state formula $\overline{\sigma}$ holding in state $|\psi\rangle$. Finally, formula $\Box\,\overline{\sigma}$ stands for state formula $\overline{\sigma}$ holding in every state. Recall, as was said above, that vectors are linear combinations of base vectors. Due to space limitations we have refrained from presenting their full language here.

As would be expected these state terms and formulas are reducible to the language of the theory of real closed fields. Like program expressions, state terms are affine maps. We present below the relevant cases.

- $\mathsf{AM}(\int\sigma)$ for $\langle 1|\circ(\mathsf{Pr}\,\sigma)$
- $\mathsf{AM}(\oint\sigma)$ for $\langle 1|\circ(\mathsf{VP}\,\sigma)$
- $\mathsf{AM}(\delta\circ\theta)$ for $(\mathsf{AM}\,\delta)\circ(\mathsf{LT}\,\theta)$

The transformation for $\int\sigma$ when applied to a state, selects all the valuations of that state that satisfy σ, using the projector $\mathsf{Pr}\,\sigma$, and then add up the coefficients of those valuations in the resulting vector using the affine transformation $\langle 1|$. The transformation for $\oint\sigma$ is similar, except that it only considers the proper valuations by using $\mathsf{VP}\,\sigma$.

State formulas, like guards, are projectors. We present below just the relevant transformations, as some of these transformations are similar to the ones for program guards.

- $\mathsf{Pr}\,\sigma$ for $\mathsf{VP}\,\sigma+|\star\rangle\langle\star|$
- $\mathsf{VP}(\delta_1\leq\delta_2)$ for $\sum_{v\in\mathbb{K}^{\mathbb{M}}}\mathsf{leq}((\mathsf{AM}\,\delta_1)|v\rangle,(\mathsf{AM}\,\delta_2)|v\rangle)|v\rangle\langle v|$

The value of extended state formulas is presented in in the sequel.

- $\sigma(|\psi\rangle)=x\Leftrightarrow((\mathsf{Pr}\,\sigma|\psi\rangle=|\psi\rangle\wedge x=1)\vee(\neg(\mathsf{Pr}\,\sigma|\psi\rangle=|\psi\rangle)\wedge x=0))$
- $(\delta_1\leq\delta_2)(|\psi\rangle)=\mathsf{leq}((\mathsf{AM}\,\delta_1)|\psi\rangle,(\mathsf{AM}\,\delta_2)|\psi\rangle)$
- $(\neg\overline{\sigma})(|\psi\rangle)=1-\overline{\sigma}(|\psi\rangle)$
- $(\overline{\sigma}_1\wedge\overline{\sigma}_2)(|\psi\rangle)=\overline{\sigma}_1(|\psi\rangle)\,\overline{\sigma}_2(|\psi\rangle)$
- $(\forall\overline{\sigma})(|\psi\rangle)=x\Leftrightarrow(((\forall y_0\,\overline{\sigma}(|y_0\rangle)=1)\wedge x=1)\vee((\exists y_0\,\overline{\sigma}(|y_0\rangle)=0)\wedge x=0))$

The denotation of $\overline{\sigma}$ at a given state will be either 0 or 1, depending on whether $\overline{\sigma}$ holds or not at that state. In this case, the comparison between two terms δ_1 and δ_2 yields a global flavor: we first evaluate each of the terms in the state and only afterward compare the resulting values, which is different from the semantics of state formulas where the comparison is made in each component of the state.

Having all of the above, it is not very difficult to conclude that reasoning about probabilistic programs can be reduced to reasoning in the theory of real closed fields and linear transformations. In the sequel, we will write $\alpha_1,\ldots,\alpha_2\vdash_{\mathsf{PP}}\alpha$ to express that α can be derived from $\alpha_1,\ldots,\alpha_n$ in the theory of probabilistic programs that we just presented.

Let us now look at some examples. We may write $\oint\top=1$ stating that in the state at hand the proper valuations have mass 1, i.e. probability 1. Equivalently, we can also express that the state at hand has no improper component by writing $\int@\star=0$. Clearly, $\oint\top=1\Leftrightarrow\int@\star=0$ is a theorem.

Reasoning about programs can be done by rewriting the corresponding formulas in the primitive language of the theory of real closed fields (by expanding the abbreviations) and invoking a decision algorithm for that theory. Observe that the expansion step is not time polynomial but it is space polynomial. This degree of complexity is to be expected given the exponential nature of the state space in terms of the number of program registers. Clearly, this algorithmic method is not very convenient for proofs by hand. In the next section, we present a calculus for reasoning mass terms that can be used for that purpose. The axioms and the rules of the calculus are all derivable form the theory of real closed fields.

4 Mass calculus

We present in the sequel the axioms and rules of a calculus suitable for reasoning by hand about proper mass. Calculi for reasoning about total and improper mass can also be provided, but are omitted due to space limitations.

- $\oint(\sigma_1\vee\sigma_2)=\oint\sigma_1+\oint\sigma_2-\oint(\sigma_1\wedge\sigma_2)$
- $\oint\bot=0$
- $\oint\sigma\geq 0$
- $\oint\sigma\circ\mathsf{skip}=\oint\sigma$
- $\oint\sigma\circ\mathsf{abort}=0$
- $\oint\sigma\circ(\gamma!)=\oint(\sigma\wedge\gamma\wedge(\gamma\downarrow))$
- $\oint\sigma\circ(\gamma?)=\oint(\sigma\wedge\gamma\wedge(\gamma\downarrow))$
- $\oint\sigma\circ\mathsf{r}_m\!\leftarrow\!\eta=\oint(\sigma[\![^{\mathsf{r}_m}_{\eta}\wedge\eta\downarrow)$
- $\oint\sigma\circ(\theta_1;\theta_2)=\oint\sigma\circ\theta_2\circ\theta_1$

- $\oint \sigma \circ (\theta_1\, n_1 | n_2\, \theta_2) = \begin{cases} \frac{1}{n_1+n_2}(n_1 \oint \sigma \circ \theta_1 + n_2 \oint \sigma \circ \theta_2) & \text{if } n_1 > 0 \wedge n_2 > 0 \wedge (n_1+n_2)\downarrow \\ 0 & \text{otherwise} \end{cases}$
- $\oint \sigma \circ (\mathsf{if}\ \gamma\ \mathsf{then}\ \theta_1\ \mathsf{else}\ \theta_2) = \oint \sigma \circ \theta_1 \circ (\gamma?) + \oint \sigma \circ \theta_2 \circ ((\neg\gamma)?)$
- $\oint \sigma \circ (\mathsf{while}\ \gamma\ \mathsf{do}\ \theta) = \oint \sigma \circ \sum_{k=0}^{K} ((\neg\gamma)?) \circ (\theta \circ (\gamma?))^k$
- $\dfrac{\forall (\sigma_1 \leq \sigma_2)}{\oint \sigma_1 \circ \theta \leq \oint \sigma_2 \circ \theta}$

Most of these axioms are what would be expected. We briefly discuss some of them. The axiom for abort imposes that after that command the proper mass is 0. The axiom for the test states that the proper mass of σ after γ! corresponds to σ and γ holding together with the computation termination of γ. We will write $\overline{\sigma}_1, \ldots, \overline{\sigma}_n \vdash_{\mathsf{M}} \overline{\sigma}$ to say that $\overline{\sigma}$ was obtained from $\overline{\sigma}_1, \ldots, \overline{\sigma}_n$ using the above axioms and rules. As would be expected we have the following *soundness* result:

$$\text{If } \overline{\sigma}_1, \ldots, \overline{\sigma}_n \vdash_{\mathsf{M}} \overline{\sigma} \text{ then } \vdash_{\mathsf{PP}} \Box((\bigwedge_{j=1}^{n} \overline{\sigma}_j) \Rightarrow \overline{\sigma}).$$

5 Concluding remarks

We have provided a method for deciding the correctness of bounded resources probabilistic programs by considering a program to be a linear transformation over a vector space. One of the key ingredients was to look at probabilistic states as vectors in the linear space generated from the classical states.

Reasoning about programs can be reduced to reasoning in a real closed field, known to be decidable. However, we have also provided a mass calculus suitable for reasoning by hand about probabilistic programs. This allows us to envisage computer aided proof systems where certain parts of the proof can be done manually, using the mass calculus, and other parts can be handed to a machine, based on decision algorithms for the theory of real closed fields.

It should be stressed that nothing is lost or mistreated by replacing the field $\mathbb{R}$ of real numbers by an arbitrary real closed field. Indeed, for the specific language of the theory of real closed fields a formula is a theorem of the theory if and only if it holds in $\mathbb{R}$ (see for instance [1]). Hence, our theory of probabilistic programs and the derived mass calculus does talk about probabilities in $\mathbb{R}$.

The techniques presented here are easily extended to classical non deterministic programs, as well as quantum programs. Work in those directions is already under way [9, 6], in the latter case looking at states not as vectors but as density operators in order to deal with the probabilistic and quantum nature of states of quantum programs. The same idea could be followed in order to deal with non determinism and randomness at the same time.

References

[1] S. Basu, R. Pollack, and R. Marie-Françoise. *Algorithms in Real Algebraic Geometry*. Springer Verlag, 2003.

[2] R. Fagin, J.Y. Halpern, and N. Megiddo. A logic for reasoning about probabilities. *Information and Computation*, 87(1–2):78–128, 1990.

[3] D. Kozen. Semantics of probabilistic programs. *Journal of Computer System Science*, 22:328–350, 1981.

[4] D. Kozen. A probabilistic PDL. *Journal of Computer System Science*, 30:162–178, 1985.

[5] P. Mateus and A. Sernadas. Weakly complete axiomatization of exogenous quantum propositional logic. *Information and Computation*, 204(5):771–794, 2006. ArXiv math.LO/0503453.

[6] P. Mateus, A. Sernadas, and J. Ramos. Deciding the correctness of imperative programs with bounded resources II: Quantum procedures. Technical report, SQIG - IT, 2008. In preparation.

[7] P. Mateus, A. Sernadas, and C. Sernadas. Exogenous semantics approach to enriching logics. In G. Sica, editor, *Essays on the Foundations of Mathematics and Logic*.

[8] N.J. Nilsson. Probabilistic logic revisited. *Artificial Intelligence*, 59(1–2):39–42, 1993.

[9] A. Sernadas, J. Ramos, and P. Mateus. Deciding the correctness of imperative programs with bounded resources I: Classical and probabilistic procedures. Technical report, SQIG - IT, 2008. In preparation.

[10] A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, 1951.