A Higher Order Characterization of Probabilistic Polynomial Time

Ugo Dal Lago Paolo Parisen Toldin

Abstract

We present RSLR, an implicit higher order characterization of the class **PP** of those problems which can be decided in probabilistic polynomial time with error probability smaller than $1/2$. Analogously, we can get a characterization of the class **BPP**. RSLR is an extension of Hofmann's SLR with a probabilistic primitive. We prove that this system enjoys subject reduction. Polytime soundness is obtained by syntactical means, as opposed to the standard literature on SLR-derived systems, which use semantics in an essential way.

1 Introduction

Implicit Computational Complexity (ICC) combines computational complexity, logic, and formal systems to give a machine independent account of complexity phenomena. It has been successfully applied to the characterization of a variety of complexity classes, especially in the sequential and parallel modes of computation (e.g., **FP**, **PSPACE**, **LOGSPACE**, **NC**). Its techniques, however, may be applied also to non-standard paradigms, like quantum computation [5] and concurrency [4].

Randomized computation is central to several areas of theoretical computer science, including cryptography, analysis of computation dealing with uncertainty, incomplete knowledge agent systems. In the context of computational complexity, probabilistic complexity classes like **BPP** are nowadays considered as very closely corresponding to the informal notion of feasibility, since a solution to a problem in **BPP** can be computed in polynomial time up to any given degree of precision.

Probabilistic polynomial computations, seen as oracle computations, were showed to be amenable to implicit techniques since the early days of ICC, by a relativization of Bellantoni and Cook's safe technique [1]. They were then studied again in the context of formal systems for security, where probabilistic polynomial computations play a major role [6, 8]. These two systems build on Hofmann's work on SLR, adding a random choice operator to the calculus. The system in [6], however, lacks higher-order recursion, and in both papers the characterization of the probabilistic classes is obtained by semantical means. While this is fine for completeness, we think it is not completely satisfactory for soundness—we know from the semantics that for any term of a suitable type its normal form *may be* computed in the given bound, but no notion of evaluation is given for which computation time is guaranteed to be bounded.

In this paper we propose RSLR, another probabilistic variant of SLR, and we show that it characterizes the class **PP** of those problems which can be solved in polynomial time by a Turing Machine with error probability smaller than $1/2$. This requires proving that any term in the language can be reduced in polynomial time, but also that any problems in **PP** can be represented in RSLR. A similar result will be proved for **BPP**. Unlike [6], RSLR has higher order recursion. Unlike [6] and [8], the bound on reduction time is obtained by syntactical means, giving an explicit notion of reduction which may realize that bound.

1.1 Related Works

We discuss here in more details the relations of our system to the previous work we already cited.

Mitchell, Mitchell, and Scedrov [6] introduce OSLR, a type system that characterizes oracle polynomial time functionals. Even if inspired by SLR [3], OSLR does not admit primitive recursion on higher-order types, but only on ground types. The main theorem shows that terms of type $\Box\mathbf{N}^m \to \mathbf{N}^n \to \mathbf{N}$ define precisely the *oracle polynomial time functionals*, which constitutes a class related but different from the ones we are interested in here. Finally, inclusion in the polynomial time class is proved without studying reduction from an operational view, but only via semantics: it is not clear for *which* notion of evaluation, computation time is guaranteed to be bounded.

Zhang's [8] introduces a further system (CSLR) which builds on OSLR and allows higher-order recursion. The main interests of the paper are applications to the verification of security protocols. It is stated that CSLR defines exactly those functions that can be computed by probabilistic Turing machines in polynomial time, via a suitable variation of Hofmann's techniques as modified by Mitchell et al. This is again a purely semantical proof.

Finally, both works are derived from Hofmann's one, and as a consequence they both have potential problems with subject reduction. Indeed, as Hofmann showed in his work, subject reduction does not hold in SLR.

1.2 RSLR: An Informal Account

Our system is called RSLR, which stands for Random Safe Linear Recursion.

RSLR can be though of as the system obtained by endowing SLR with a random primitive. Some restrictions are needed to be able to prove some operational form of polynomial time soundness. The final result is indeed quite similar to (a probabilistic variation of) Bellantoni, Niggl and Schwictenberg calculus RA [2, 7].

Actually, the only needed restriction with respect to SLR deals with linearity: keeping the size of reducts under control during normalization is very difficult in presence of higher-order duplication. For this reason, the two function spaces $A \to B$ and $A \multimap B$ collapse to just one in RSLR. This allows us to get a reasonably simple system for which polytime soundness can be proved explicitly, by studying the combinatorics of reduction. More specifically, terms of RSLR are shown to be reducible in polynomial time using a mixed strategy, where arguments of ground types are handled as in call-by-value, while those of higher-order types are passed to functions without being evaluated.

2 The Syntax of RSLR

RSLR is a fairly standard Curry-style lambda calculus with constants for the natural numbers, branching and recursion. Its type system, on the other hand, is based on ideas coming from linear logic (some variables can appear at most once in terms) and by a distinction between modal and nonmodal variables.

Let us introduce the category of types first:

Definition 2.1 (Types)**.** The *types* of RSLR are generated by the following grammar:

$$A ::= \mathbf{N} \mid \Box A \to A \mid \blacksquare A \to A.$$

Types different from $\mathbf{N}$ are denoted with metavariables like H or G. $\mathbf{N}$ is the only *ground type*

There are two function spaces in our system. Programs which can be typed as $\blacksquare A \to B$ are such that the result in B can be computed in time somehow independent on the size of the argument (of type A). On the other hand, computing the result of functions in $\Box A \to B$ requires polynomial time in the size of their argument.

A notion of subtyping is used in RSLR to capture the intuition above by stipulating that the type $\blacksquare A \to B$ is a subtype of $\Box A \to B$.

Definition 2.2 (Aspects)**.** An *aspect* is either $\Box$ or $\blacksquare$. Aspects are partially ordered by a relation $<:$ such that $\Box <: \blacksquare$.

$$\frac{}{A <: A}\ (\text{S-Refl}) \qquad \frac{A <: B \qquad B <: C}{A <: C}\ (\text{S-Trans})$$

$$\frac{B <: A \qquad C <: D \qquad b <: a}{aA \to C <: bB \to D}\ (\text{S-Sub})$$

Figure 1: Subtyping rules.

Subtyping rules are in Figure 1.

Definition 2.3 (Terms)**.** Terms and constants are defined as follows:

$$t ::= x \mid c \mid ts \mid \lambda x : aA.t \mid \mathbf{case}_A t\ \mathbf{zero}\ s\ \mathbf{even}\ r\ \mathbf{odd}\ q \mid \mathbf{recursion}_A t\, s\, r;$$
$$c ::= n \mid \mathbf{S}_0 \mid \mathbf{S}_1 \mid \mathbf{P} \mid \mathbf{rand}.$$

where x ranges over a numerable set of variables and n ranges over the natural numbers seen as constants of base type. Every constant c has its naturally defined type $\mathit{type}(c)$. As an example, $\mathit{type}(n) = \mathbf{N}$ for every n, while $\mathit{type}(\mathbf{S}_0) = \blacksquare\mathbf{N} \to \mathbf{N}$. The size $|t|$ of any term t can be easily defined by induction on t.

As already mentioned, our notion of reduction is mixed: arguments of ground type are evaluated before being passed to functions, while arguments of an higher-order type are passed to functions possibly unevaluated, in a call-by-name fashion. Before doing that, let's define the one-step reduction relation:

Definition 2.4 (Reduction)**.** The *one-step reduction relation* $\to$ is a binary relation between terms and sequences of terms. It is defined by the axioms in Figure 2 and can be applied in any contexts, except in the second and third argument of a recursion. A term t is in *normal form* if t cannot appear as the left-hand side of a pair in $\to$. The set of terms in normal form is NF.

$$\begin{aligned}
\mathbf{case}_A 0\ \mathbf{zero}\ t\ \mathbf{even}\ s\ \mathbf{odd}\ r &\to t;\\
\mathbf{case}_A(\mathbf{S}_0 n)\ \mathbf{zero}\ t\ \mathbf{even}\ s\ \mathbf{odd}\ r &\to s;\\
\mathbf{case}_A(\mathbf{S}_1 n)\ \mathbf{zero}\ t\ \mathbf{even}\ s\ \mathbf{odd}\ r &\to r;\\
\mathbf{recursion}_\tau 0\, g\, f &\to g;\\
\mathbf{recursion}_\tau n\, g\, f &\to fn(\mathbf{recursion}_\tau \lfloor \frac{n}{2} \rfloor\, g\, f);\\
\mathbf{S}_0 n &\to 2 \cdot n;\\
\mathbf{S}_1 n &\to 2 \cdot n + 1;\\
\mathbf{P}0 &\to 0;\\
\mathbf{P}n &\to \lfloor \frac{n}{2} \rfloor;\\
(\lambda x : a\mathbf{N}.t)n &\to t[x/n];\\
(\lambda x : aH.t)s &\to t[x/s];\\
(\lambda x : aA.t)sr &\to (\lambda x : aA.tr)s;\\
\mathbf{rand} &\to 0, 1;
\end{aligned}$$

Figure 2: Reduction Steps: Axioms

A multistep reduction relation will not be defined by simply taking the transitive and reflective closure of $\rightarrow$, since a term can reduce in multiple steps to many terms with different probability. This implies we need the notion of a *distribution*, by which we mean here any function $\mathscr{D}$ from normal forms to $\mathbb{R}_{[0,1]}$ such that $\sum_t \mathscr{D}t = 1$. For every normal form t, the distribution $\mathscr{G}_t$ is simply the unique distribution such that $\mathscr{G}_t(t) = 1$.

Definition 2.5 (Multistep Reduction)**.** The binary relation $\rightsquigarrow$ between terms and probability distributions is defined in Figure 3.

$$\frac{t \rightarrow t_1, \ldots, t_n \qquad t_i \rightsquigarrow \mathscr{D}_i}{t \rightsquigarrow \sum_{i=1}^{n} \frac{1}{n}\mathscr{D}_i} \qquad \frac{t \in NF}{t \rightsquigarrow \mathscr{G}_t}$$

Figure 3: Multistep Reduction: Inference Rules

We are finally able to present the type system. Preliminary to that is the definition of a proper notion of a context.

Definition 2.6 (Contexts)**.** A *context* Γ is a finite set of assignment of types and aspects to variables, in the form $x : aA$. The union of two disjoint contexts Γ and Δ is denoted as Γ, Δ. In doing so, we implicitly assume that the variables in Γ and Δ are pairwise distinct. The union Γ, Δ is sometimes denoted as $\Gamma; \Delta$: this way we want to stress that all types appearing in Γ are ground. With the expression $\Gamma <: a$ we mean that any aspect b appearing in Γ is such that $b <: a$.

$$\frac{x \in dom(\Gamma)}{\Gamma \vdash x : \Gamma(x)}\ (\text{T-Var-Aff}) \qquad \frac{\Gamma \vdash t : A \qquad A <: B}{\Gamma \vdash t : B}\ (\text{T-Sub})$$

$$\frac{\Gamma, x : aA \vdash t : B}{\Gamma \vdash \lambda x A.t : aA \rightarrow B}\ (\text{T-Arr-I}) \qquad \frac{}{\Gamma \vdash c : type(c)}\ (\text{T-Const-Aff})$$

$$\frac{\begin{array}{cc}\Gamma; \Delta_1 \vdash t : \mathbf{N} & \Gamma; \Delta_3 \vdash r : A \\ \Gamma; \Delta_2 \vdash s : A & \Gamma; \Delta_4 \vdash q : A \qquad A \text{ is } \Box\text{-free}\end{array}}{\Gamma; \Delta_1, \Delta_2, \Delta_3, \Delta_4 \vdash \mathbf{case}_A t\ \mathbf{zero}\ s\ \mathbf{even}\ r\ \mathbf{odd}\ q : A}\ (\text{T-Case})$$

$$\frac{\begin{array}{cc}\Gamma_1; \Delta_1 \vdash t : \mathbf{N} & \\ \Gamma_1, \Gamma_2; \Delta_2 \vdash s : A & \Gamma_1; \Delta_1 <: \Box \\ \Gamma_1, \Gamma_2; \vdash r : \Box\mathbf{N} \rightarrow \blacksquare A \rightarrow A & A \text{ is } \Box\text{-free}\end{array}}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash \mathbf{recursion}_A t\, s\, r : A}\ (\text{T-Rec})$$

$$\frac{\Gamma; \Delta_1 \vdash t : aA \rightarrow B \qquad \Gamma; \Delta_2 \vdash s : A \qquad \Gamma, \Delta_2 <: a}{\Gamma, \Delta_1, \Delta_2 \vdash (ts) : B}\ (\text{T-Arr-E})$$

Figure 4: Type rules

2.1 Confluence

Theorem 2.1 (confluence multi-step)**.** *If $t \rightsquigarrow \mathscr{D}$ and $t \rightsquigarrow \mathscr{E}$ then $\mathscr{D} \equiv \mathscr{E}$.*

3 The Main Results

Definition 3.1. A *first-order term* of arity k is a closed, well typed term of type $a_1\mathbf{N} \rightarrow a_2\mathbf{N} \rightarrow \ldots a_k\mathbf{N} \rightarrow \mathbf{N}$ for some $a_1, \ldots, a_k$.

The most difficult (and interesting!) result about RSLR is definitely polytime soundness: every (instance of) a first-order term can be reduced to its normal form in a polynomial number of steps using the previously introduced notion of reduction. Polytime soundness can be proved, following [2], by showing that:

- Any term of base type which does not contain any recursion can be reduced to its normal form with very low time complexity.
- Any term (non necessarily of base type!) can be reduced to another not containing any recursion in polynomial time.

By glueing these two results together, we obtain what we need, namely an effective and efficient procedure to compute the normal forms of terms.

Formally:

Theorem 3.1 (Soundness)**.** *Suppose t is a first order term of arity k. Then there is a Probabilistic Turing Machine M_t running in polytime such that M_t on input $n_1 \dots n_k$ returns n with probability exactly $\mathscr{E}(n)$, where $\mathscr{E}$ is a probability distribution such that $tn_1 \dots n_k \rightsquigarrow \mathscr{E}$.*

What about the converse? Clearly, we can proceed as in [6] and prove that the class of *probabilistic* functionals which are representable in RSLR equals the class of those probabilistic functionals which can be computed in polynomial time. Here, however, we are more interested in the correspondence with complexity classes, which are classes of subsets of the natural numbers.

Definition 3.2. Let t a first-order term of arity 1 and let $\alpha \in \mathbb{R}_{[0,1]}$. Then t is said to *α-represents* a language $L \subseteq \mathbb{N}$ iff:

1. If $n \in L$ and $tn \rightsquigarrow \mathscr{D}$, then $\mathscr{D}(0) \geq \alpha$
2. If $n \notin L$ and $tn \rightsquigarrow \mathscr{D}$, then $\sum_{m>0} \mathscr{D}(m) \geq \alpha$

Theorem 3.1, together with an encoding of probabilistic Turing Machines into RSLR allows us to conclude that:

Theorem 3.2 (Completeness for **PP**)**.** *The set of languages which can be α-represented in* RSLR *for some $1/2 < \alpha \leq 1$ equals* **PP**.

So, in a sense, RSLR is strongly linked with the complexity class **PP**.

But, interestingly, we can go beyond and capture a more interesting complexity class:

Theorem 3.3 (Completeness for **BPP**)**.** *The set of languages which can be α-represented in* RSLR *for some $2/3 \leq \alpha \leq 1$ equals* **BPP**.

References

[1] S Bellantoni. Predicative recursion and the polytime hierarchy. In P. Clote and J.B. Remmel, editors, *Feasible Mathematics II*, pages 15–29. Birkhauser, 1995.

[2] S.J. Bellantoni, K.H. Niggl, and H. Schwichtenberg. Higher type recursion, ramification and polynomial time. *Annals of Pure and Applied Logic*, 104(1-3):17–30, July 2000.

[3] Martin Hofmann. A mixed modal/linear lambda calculus with applications to bellantoni-cook safe recursion. In Mogens Nielsen and Wolfgang Thomas, editors, *CSL*, volume 1414 of *Lecture Notes in Computer Science*, pages 275–294. Springer, 1997.

[4] Ugo Dal Lago, Simone Martini, and Davide Sangiorgi. Light logics and higher-order processes. In Sibylle B. Fröschle and Frank D. Valencia, editors, *EXPRESS'10*, volume 41 of *EPTCS*, pages 46–60, 2010.

[5] Ugo Dal Lago, Andrea Masini, and Margherita Zorzi. Quantum implicit computational complexity. *Theor. Comput. Sci.*, 411(2):377–409, 2010.

[6] John C. Mitchell, Mark Mitchell, and Andre Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *FOCS*, pages 725–733, 1998.

[7] Helmut. Schwichtenberg and Steven Bellantoni. Feasible computation with higher types. In *Proof and System-Reliability*, pages 399–415. Kluwer Academic publisher, 2001.

[8] Yu Zhang. The computational slr: a logic for reasoning about computational indistinguishability. *Mathematical Structures in Computer Science*, 20(5):951–975, 2010.