Information and Computation 248 (2016) 130–149

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

The role of polymorphism in the characterisation of complexity by soft types ☆

Jacek Chrząszcz *, Aleksy Schubert *

Institute of Informatics, University of Warsaw, ul. Banacha 2, 02-097 Warsaw, Poland

ARTICLE INFO

Article history:
Received 11 November 2013
Available online 6 January 2016

Keywords:
Implicit computational complexity
λ-Calculus
Polymorphism
Undecidability

ABSTRACT

Soft type assignment systems STA, STA_+, and STA_B characterise by means of reduction of terms computation in complexity classes PTIME, NP, and PSPACE, respectively. All these systems are inspired by linear logic and include polymorphism similar to the one of System F. We show that the presence of polymorphism gives the undecidability of typechecking and type inference. We also show that reductions in decidable monomorphic versions of these systems also capture the same complexity classes in a way sufficient for the traditional complexity theory. The translations we propose show in addition that the monomorphic systems to serve as a programming language require some metalanguage support since the program which operates on data has form and type which depend on the size of the input.

1. Introduction

One of the goals of Implicit Computational Complexity studies is the search of a programming language which captures a particular complexity class, e.g. PTIME. The research may lead not only to a different theoretical view on the class, but also to a programming language with strong guarantees on time or memory consumption.

The relation of the computational complexity to type systems was studied already in the Statman's characterisation of nonelementary computation by reductions in the simply typed lambda calculus [21] which was later simplified by Mairson [16] and, for a calculus with additional δ reduction rules, refined by Goerdt [10]. This was also studied in the pure simply typed calculus for types with low functional orders [19,23].

The current paper has its roots in the study of complexity classes by means of linear logic started by Girard in Light Linear Logic (LLL) and Elementary Linear Logic (ELL) [9] where polynomial and elementary time complexities respectively are considered and related to computation via cut-elimination in proof nets. Polynomial time is also characterised by Soft Linear Logic (SLL) of Lafont [13]. The line started by Lafont brought the starting point of the current paper, i.e. systems STA, STA_+, and STA_B of Gaboardi et al. [5–7] which, when appropriately tailored, capture the complexity classes of PTIME, NP, and PSPACE, respectively. As we explain it below, all the systems capture the classes in a specific way which extends the traditional reduction notion of the complexity theory. The papers provide for each Turing Machine in PTIME (NP, PSPACE respectively) a single term which when applied to the encoding of an input gives an answer which encodes the answer of the original machine. Since the term represents a function of the input, this can be generalised to showing that every

☆ Partly supported by NCN grant DEC-2012/07/B/ST6/01532.
* Corresponding authors.
E-mail addresses: chrzaszcz@mimuw.edu.pl (J. Chrząszcz), alx@mimuw.edu.pl (A. Schubert).

http://dx.doi.org/10.1016/j.ic.2015.12.011
0890-5401/

function from the FPTIME class is representable in STA by a term the reduction sequence of which for any given input has polynomial length.

We focus in the current paper on the question of what one can say on the systems STA, STA_+, $\mathrm{STA}_\mathbb{B}$ when we apply the notion of problem reduction as used in the traditional complexity theory. Using a particular problem of computation by terms we show that PTIME- and NP-completeness can be obtained even in monomorphic systems STA-mono, STA_+-mono, respectively. The terms we use to show hardness satisfy additional restrictions on the functional order of redexes. In case of PSPACE, the additional constraint can be lifted and we can obtain PSPACE-completeness for the monomorphic system $\mathrm{STA}_\mathbb{B}$-mono with the same restriction as in case of the polymorphic one. This is due to the presence of algebraic type of booleans. This observation leads us to the conclusion that the corresponding amendments to STA-mono, STA_+-mono lead to systems where the additional constraint on the order of redexes can be lifted. It is known since the results of Leivant and Marion [14,15] that appropriate restrictions on terms built with algebraic constructs lead to a characterisation of the polynomial complexity class even in the simply typed lambda calculus. In our paper we obtain a significantly more restricted set of constants that can give a sensible characterisation of the complexity classes. Still, there is a cost of this austere choice of means, namely, our characterisation does not give a single term to simulate a Turing Machine. This elucidates from a different perspective the point already mentioned by Mairson and Terui [17] that polymorphism is needed in the systems that characterise complexity classes to combine structurally similar definitions into a single term.

The addition of polymorphism makes it possible to uniformly represent data and functions that operate on them. This can be captured by a notion of uniformity introduced by Dal Lago [3] and further studied in the context of implicative intuitionistic light affine logic (ILAL) by Dal Lago and Baillot [4]. In the current paper we supplement the discussion there by showing a characterisation of a number of term classes where deterministic polynomial computations can be realised, but have too little expressivity to represent this computational power in a fashion where a single machine is encoded by a single term.

A big drawback of the polymorphic systems is that the typechecking and type inference problems are, as shown here, undecidable. This can be mitigated by a system with a limited polymorphism. In particular the monomorphic version of STA, where there is no polymorphism, enjoys decidable typechecking and type inference problems [8]. It is now interesting to look for systems with minimal amount of polymorphism sufficient to obtain decidable type inference, but strong enough to combine necessary definitions into programs which work for all inputs.

The paper is organised as follows. Section 2 presents the soft type assignment systems. In Section 3 we show that the polymorphic systems have undecidable typechecking and type inference problems. The characterisation of complexity classes by the monomorphic systems is given in Section 4.

This article is an extended version of the paper presented at the MFCS conference [2]. We enhance the discussion there by presentation of omitted proofs and illustrative examples.

2. Presentation of systems

The main focus of the paper is on a presentation of the distinction between the polymorphic systems STA, STA_+, $\mathrm{STA}_\mathbb{B}$ and their monomorphic versions STA-mono, STA_+-mono, $\mathrm{STA}_\mathbb{B}$-mono. Let us present all the six systems. Their types are built using the following pair of grammars.

Polymorphic types	Monomorphic types	
$A ::= \alpha \mid \sigma \multimap A \mid \forall\alpha.A$	$A ::= \alpha \mid \sigma \multimap A$	(Linear Types)
$\sigma ::= A \mid !\sigma$	$\sigma ::= A \mid !\sigma$	(Bang Types)

The polymorphic systems have types generated in the column *Polymorphic types* while the monomorphic ones in the column *Monomorphic types*. The systems $\mathrm{STA}_\mathbb{B}$ and $\mathrm{STA}_\mathbb{B}$-mono, in addition, use a fixed type variable $\mathbb{B}$ which is a domain for boolean values $\mathbf{0}$ and $\mathbf{1}$. The quantifier $\forall$ is a binding operator and the variable α in $\forall\alpha.A$ is bound. We identify types that differ only in the names of bound type variables. The set of free type variables in A is written $\mathrm{FTV}(A)$. We can substitute type B for a free type variable α in type A so that no free variable is captured during the operation. The operation is written $A[B/\alpha]$. A precise definition of substitution can be found in the book by Sørensen and Urzyczyn [20, Section 11.1]. We write $\forall.A$ to express type where all free type variables in A are bound by a series of quantifiers, e.g. $\forall.\alpha \multimap \beta = \forall\alpha\beta.\alpha \multimap \beta$. We also use a shorthand $A \overset{i}{\multimap} B$ to mean the type $A \multimap \cdots \multimap A \multimap B$ with i occurrences of A, in particular $A \overset{1}{\multimap} B$ is $A \multimap B$ and $A \overset{0}{\multimap} B$ is B.

The terms of the systems are defined using the following grammars:

$$
\begin{array}{ll}
M ::= x \mid \lambda x.M \mid M_1 M_2 & \text{terms of STA,} \\
M ::= x \mid \lambda x.M \mid M_1 M_2 \mid M_1 + M_2 & \text{terms of } \mathrm{STA}_+, \\
M ::= x \mid \lambda x.M \mid M_1 M_2 \mid \mathbf{0} \mid \mathbf{1} \mid \mathtt{if}\ M_1\ \mathtt{then}\ M_2\ \mathtt{else}\ M_3 & \text{terms of } \mathrm{STA}_\mathbb{B}.
\end{array}
$$

The terms of the monomorphic systems are the same as of their respective polymorphic counterparts. As usual, the λ operator is a binding operator and the variable x is bound in the term $\lambda x.M$. We consider terms up to α-conversion, i.e. two