Information and Computation 248 (2016) 215–226

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Bounded Combinatory Logic and lower complexity

Brian F. Redmond

Department of Computing, Mathematics and Statistical Sciences, Grande Prairie Regional College, 10726-106 Avenue, Grande Prairie, AB T8V 4C4, Canada

ARTICLE INFO

Article history:
Received 21 October 2014
Available online 6 January 2016

Keywords:
Combinatory Logic
Stratification
Implicit computational complexity
Polynomial time and space
Soft linear logic

ABSTRACT

We introduce a stratified version of Combinatory Logic[1] in which there are two classes of terms called player and opponent such that the class of player terms is strictly contained in the class of opponent terms. We show that the system characterizes Polynomial Time in a similar way to Soft Linear Logic. With the addition of explicit "lazy" products to the player terms and various notions of distributivity, we obtain a characterization of Polynomial Space. This paper is an expanded version of the abstract presented at DICE 2013.

1. Introduction

Combinatory Logic (CL), introduced by Moses Schönfinkel and independently by Haskell Curry, is an extremely simple rewrite system with universal computational power [13]. Terms are constructed from variables and the two basic combinators S and K via application. Their corresponding rewrite rules are given in Table 1. An equivalent presentation of CL is obtained by replacing the combinator S by the three combinators B, C, W together with their corresponding rewrite rules in Table 1. In this paper, CL will refer to Combinatory Logic based on the combinators B, C, K and W.

There is an important subsystem of CL, called BCK-logic, obtained by removing the duplication combinator W altogether. As this is the only combinator responsible for duplication, it is straightforward to show that terms in BCK-logic strongly normalize in a number of steps bounded by their size. The drawback is that BCK-logic is computationally very weak.

Abramsky proposed – in the spirit of linear logic – to add to BCK-logic a unary operator, denoted ! (read 'bang'), on terms and a handful of new basic combinators governing its use [1,2]. The full computational power of CL is then recovered by a suitable interpretation of standard application and the standard combinators in CL. This extra structure, however, allows one to investigate systems of intermediate complexity simply by dropping one or more of these new combinators. This leads to CL (hence type-free) versions of the so-called "light" linear logics – subsystems of linear logic that characterize some important complexity classes (see, for example, [11,6,16]).

In this paper we take a different approach that is more in line with the work of Leivant and Marion [18–20] and Bellantoni and Cook [3] and stratify terms into two classes called player and opponent such that the class of player terms is strictly contained in the class of opponent terms. Rather than calling these sets safe and normal as in [3], we prefer to use the game-theoretic terminology of [4,5]. The player world consists of BCK-terms and is computationally very weak,

E-mail address: bredmond@GPRC.ab.ca.

[1] Strictly speaking, Combinatory Logic is not a logic, but a rewrite system. At the risk of offending the reader, this abuse of terminology will continue to be used in this paper as it is entrenched in the literature. Note, however, that the rewrite systems introduced in this paper do have an intended type system/logic (and semantics) that will be presented elsewhere.

http://dx.doi.org/10.1016/j.ic.2015.12.013
0890-5401/

Table 1
Basic combinators and their rewrite rules.

Combinator	Redex	Contractum
S	SXYZ	$XZ(YZ)$
B	BXYZ	$X(YZ)$
C	CXYZ	XZY
K	KXY	X
W	WXY	XYY

while the opponent world has, in addition, a duplication combinator W that can duplicate player terms. We shall show that this surprisingly simple system, called Bounded Combinatory Logic (BCL),[2] characterizes Polynomial Time computation. This result, of course, is not too surprising given similar characterizations in the typed lambda calculus (see [10,24,15,12]). However, our results do not depend on types and are based on much simpler underlying principles and techniques.

Nondeterministic computation is modeled in BCL using explicit "lazy" products together with a player distribution combinator D. The explicit product structure allows for the representation of other lazy data structures, in particular the binary trees used in the encoding of alternating Turing machines (see Section 3.2). Given a path (i.e. a choice of projections) from the root of such a tree to a leaf, we show that reduction can be simulated in Polynomial Time, provided it is done lazily. This will lead to a characterization of Polynomial Space (PSPACE). A second characterization of PSPACE is obtained with additional rewrite rules that have the same computational power as the "peek" rule of Pola [21]. The present work is not the first implicit computational complexity characterization of PSPACE; see for example [19] and [24]. More recently, Gaboardi et al. [7,8] obtained a characterization using a non-affine (or "additive") conditional "if_then_else_" construct. Essentially the same construct appears earlier in [14] by Hofmann, and is analogous to the peek rule in Pola.

The purpose of this paper is to establish the above claimed complexity results in the untyped setting. The interesting issues of semantics and types will be addressed elsewhere.

2. Bounded Combinatory Logic

The class of BCL terms is defined inductively as follows. We assume an countably infinite set Var of variables and define:

$$F ::= x \mid \mathsf{B} \mid \mathsf{C} \mid \mathsf{K} \mid FF$$

$$O ::= F \mid \mathsf{W} \mid OO$$

where $x \in \text{Var}$. F-terms are called **player**, or **affine**, and O-terms are called **opponent**. Notice that the set of player terms[3] forms a proper subset of the set of opponent terms. All terms associate to the left and we omit unnecessary brackets. The letters X, Y, Z, etc., will be used to denote arbitrary terms, but the letters F and O will be reserved for player and opponent terms, respectively. The one-step reduction rules in BCL are:

$$\begin{array}{lll} \mathsf{B}XYZ & \to_1 & X(YZ) \\ \mathsf{C}XYZ & \to_1 & XZY \\ \mathsf{K}XY & \to_1 & X \\ \mathsf{W}XF & \to_1 & XFF \end{array}$$

Note that the W combinator can only duplicate player terms. This ensures that reduction terminates (Theorem 2).

Of course, terms may also reduce in context:

$$\frac{X \to_1 X'}{XY \to_1 X'Y} \qquad \frac{Y \to_1 Y'}{XY \to_1 XY'}$$

The **size** of a BCL term X, denoted $s(X)$, is defined to be the number of basic combinators in the term. Note that player terms always reduce to player terms in a number of steps bounded by their size, while opponent terms may reduce to either player or opponent terms. We use the symbol $\equiv$ to denote the syntactic identity of terms. By **combinator** we mean a term containing no variables. We write $\to$ for the reflexive, transitive closure of the one-step reduction relation $\to_1$.

Our first result is a straightforward generalization of the corresponding result in CL (see [13], for example).

Theorem 1. *BCL is confluent.*

[2] In [22] we called the system Stratified Combinatory Logic (SCL), but the name Bounded Combinatory Logic (BCL) now seems more appropriate in light of recent work [9].

[3] For those more accustomed to the lambda calculus, player terms correspond to lambda terms in which each *bound* variable occurs at most once up to α-congruence, but free variables may occur any number of times.