Annals of Pure and Applied Logic 164 (2013) 676–701

Contents lists available at SciVerse ScienceDirect

Annals of Pure and Applied Logic

www.elsevier.com/locate/apal

The $\lambda\mu^{\mathbf{T}}$-calculus

Herman Geuvers [a,b], Robbert Krebbers [a,*], James McKinna [a]

[a] Institute for Computing and Information Science, Faculty of Science, Radboud University Nijmegen, The Netherlands
[b] Faculty of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands

ARTICLE INFO

Article history:
Available online 24 May 2012

MSC:
F.3.3
F.4.1

Keywords:
Lambda calculus with control
Confluence
Strong normalization

ABSTRACT

Calculi with control operators have been studied as extensions of simple type theory. Real programming languages contain datatypes, so to really understand control operators, one should also include these in the calculus. As a first step in that direction, we introduce $\lambda\mu^{\mathbf{T}}$, a combination of Parigot's $\lambda\mu$-calculus and Gödel's **T**, to extend a calculus with control operators with a datatype of natural numbers with a primitive recursor.

We consider the problem of confluence on raw terms, and that of strong normalization for the well-typed terms. Observing some problems with extending the proofs of Baba et al. and Parigot's original confluence proof, we provide new, and improved, proofs of confluence (by complete developments) and strong normalization (by reducibility and a postponement argument) for our system.

We conclude with some remarks about extensions, choices, and prospects for an improved presentation.

1. Introduction

In pursuit, on the one hand, of a satisfactory equational theory of call-by-value λ-calculus, and on the other, of a means to interpret the computational content of classical proofs, a variety of calculi with control operators have been proposed. Few of these systems address the problem of how to incorporate primitive datatypes in direct style, preferring instead to consider the usual Church encoding of datatypes or else to analyze computation over datatypes via CPS-translations.

In part this appears to arise because of the technical difficulty in getting standard results such as confluence or strong normalization, and their proof methods, either for classical calculi, or for simply-typed calculi with datatypes, to extend to their combination.

This paper introduces a new λ-calculus with control, $\lambda\mu^{\mathbf{T}}$, in which for example constructs for `catch` and `throw` may be represented, which moreover has a basic datatype of natural numbers with a primitive recursor, in the style of Gödel's **T**. We demonstrate that it is possible to achieve a synthesis of classical computation with datatypes with a conventional metatheory of typing and reduction. To show how the system can be used in programming, we give a simple example in 3.16, where we define a function that multiplies the first n values of $f : \mathtt{N} \to \mathtt{N}$ and throws an exception as soon as it encounters the value 0.

1.1. Our approach

Since Lafont's counterexample [19], it is well known that a calculus providing a general content to classical logic cannot be confluent. It only may become confluent if one adds an evaluation strategy (call-by-name or call-by-value). To define

* Corresponding author at: Institute for Computing and Information Science, Faculty of Science, Radboud University Nijmegen, The Netherlands.
E-mail addresses: herman@cs.ru.nl (H. Geuvers), mail@robbertkrebbers.nl (R. Krebbers), james.mckinna@cs.ru.nl (J. McKinna).

0168-0072/$ – see front matter
http://dx.doi.org/10.1016/j.apal.2012.05.005

a calculus with control operators and datatypes we have therefore observed a tension between the call-by-name features taken directly from Parigot's $\lambda\mu$-calculus, and the need to add certain call-by-value features to obtain a system that is confluent and satisfies a normal form theorem (each closed term of type $\mathtt{N}$ is convertible to a numeral). The $\lambda\mu^{\mathbf{T}}$-calculus is therefore a call-by-name system with strict evaluation on datatypes. To avoid losing a normal form theorem, we could not make it a full call-by-name system, and to avoid losing confluence we had to restrict the primitive recursor to only allow conversion when the numerical argument is a numeral.

Given these technical considerations, we were able to prove that $\lambda\mu^{\mathbf{T}}$ satisfies subject reduction, has a normal form theorem, is confluent and strongly normalizing. The last two proofs are non-trivial because various niceties are required to make the standard proof methods work.

Our confluence proof uses the notion of parallel reduction and defines a complete development for each term. Surprisingly, it was difficult to find a confluence proof for the original untyped $\lambda\mu$-calculus. Baba, Hirokawa and Fujita [3] have given a confluence proof for $\lambda\mu$ without the $\to_{\mu\eta}$-rule ($\mu\alpha.[\alpha]t \to t$ provided that $\alpha \notin \mathrm{FCV}(t)$). Although they suggest how to extend parallel reduction for the $\to_{\mu\eta}$-rule, they do not provide a formal definition of the complete development nor a proof. Nakazawa [25] has successfully carried out their suggestion for a call-by-value variant of $\lambda\mu$, but does not use the notion of complete development. Walter Py's PhD thesis [31] was the only place where we have found a complete proof of confluence for $\lambda\mu$. It uses Aczel's generalization of parallel reduction [1] and a number of postponement arguments. In the present paper we extend the methodology of [3] to the case of $\lambda\mu^{\mathbf{T}}$, which also includes the $\to_{\mu\eta}$-rule.

Our strong normalization proof proceeds by defining relations $\to_A$ and $\to_B$ such that $\to = \to_{AB} := \to_A \cup \to_B$. First we prove that $\to_A$ is strongly normalizing by the reducibility method. Secondly, we prove that $\to_B$ is strongly normalizing and that both reductions commute in a way that we can obtain strong normalization for $\to_{AB}$. The first phase is inspired by Parigot's proof of strong normalization for the $\lambda\mu$-calculus [29].

1.2. Related work

The extension of simply typed lambda calculus with control operators and the observation that these operators can be typed using the rules of classical logic is originally due to Griffin [20] and has lead to a lot of research [27,28,14,32,7,10,4,2,37], by considering variations on the control operators, the underlying calculus or the computation rules, or by studying concrete examples of the computational content of proofs in classical logic. The $\lambda\mu$-calculus of Parigot [27] has become a central starting point for much research in this area.

The extension with datatypes, to make the calculus into a real programming language with control operators, has not received so much attention. We briefly summarize the research done in this direction and compare it with our work.

Murthy has defined a system with control operators, arithmetic, products and sums in his PhD thesis [24]. His system uses the control operators $\mathcal{C}$ and $\mathcal{A}$ (originally due to [20]) and the semantics of these operators is specified by evaluation contexts rather than local reduction rules, as we do. So his system does not really describe a *calculus* for datatypes and control. Furthermore, Murthy mainly considers CPS-translations to give an operational semantics of his system and did not prove properties like confluence or strong normalization.

Crolard and Polonowski have considered a version of Gödel's **T** with products and `call/cc` [12]. As with Murthy, the semantics is presented by CPS-translations instead of a direct specification via a calculus. Therefore properties like confluence and strong normalization are trivial because they hold for the target system already.

Barthe and Uustalu have worked on CPS-translations for inductive and coinductive types [5]. Their work includes a system with a primitive for iteration over the natural numbers and the control operator Δ. Unfortunately only some properties of CPS-translations are proven.

Rehof and Sørensen have described an extension of the λ_Δ-calculus with basic constants and functions [32]. Unfortunately their extension is quite limited. For example the primitive recursor `nrec` takes terms, rather than basic constants, as its arguments. Their extension does not allow this, making it impossible to define `nrec`.

Parigot has described a second-order variant of his $\lambda\mu$-calculus [27]. This system is very powerful, because it includes all the well-known second-order representable datatypes. However, it suffers from the same weakness as System **F**, namely poor computational efficiency (for example, an $O(n)$-predecessor function). Also, as observed in [27,28], this system does not ensure *unique representation of datatypes*. For example, there is no one-to-one correspondence between natural numbers and closed normal forms of the type of Church numerals.

There have been various investigations into concrete examples of computational content of classical proofs. Coquand gives an overview in his notes [10]. An earlier example is [7], where a binpacking problem is analyzed using proof transformations. More recent work is by Makarov [23], who takes Griffin's calculus and adds various rules to optimize the extracted program.

If we look in particular at Gödel's **T**, Berger, Buchholz and Schwichtenberg have described a form of program extraction from classical proofs [6]. Their method extracts a term from a classical proof in which all computationally irrelevant parts are removed. To prove the correctness of their approach they give a realizability interpretation. However, since their target language is Gödel's **T**, extracted programs do not contain control mechanisms.

Caldwell, Gent and Underwood have considered program extraction from classical proofs in the proof assistant NuPrl [8]. In their work they extend NuPrl with a proof rule for Peirce's law and they associate `call/cc` to the extraction of Peirce's law. Now, program extraction indeed results in a program with control. The main focus of their work is on using program