Theoretical Computer Science 611 (2016) 50–86

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Termination of rewrite relations on λ-terms based on Girard's notion of reducibility

Frédéric Blanqui [1]

Institut National de Recherche en Informatique et Automatique (INRIA), France

ARTICLE INFO

Article history:
Received 13 December 2012
Received in revised form 24 February 2015
Accepted 25 July 2015
Available online 30 July 2015
Communicated by P.-A. Mellies

Keywords:
Termination
Rewriting
λ-calculus
Types
Girard's reducibility
Rewriting modulo
Matching modulo $\beta\eta$
Patterns *à la* Miller

ABSTRACT

In this paper, we show how to extend the notion of reducibility introduced by Girard for proving the termination of β-reduction in the polymorphic λ-calculus, to prove the termination of various kinds of rewrite relations on λ-terms, including rewriting modulo some equational theory and rewriting with matching modulo $\beta\eta$, by using the notion of *computability closure*. This provides a powerful termination criterion for various higher-order rewriting frameworks, including Klop's Combinatory Reductions Systems with simple types and Nipkow's Higher-order Rewrite Systems.

1. Introduction

This paper addresses the problem of checking the termination of various kinds of rewrite relations on simply typed λ-terms.

First-order rewriting [83,44] and λ-calculus [32,6] are two general (Turing-complete) computational frameworks with different strengths and limitations.

The λ-calculus is a language for expressing arbitrary functions based on a few primitives (abstraction over some variable and application of a function to an argument). Computation is done by repeatedly substituting formal arguments by actual ones (β-reduction) [32].

In first-order rewriting, one considers a fixed set of function symbols and a fixed set of term transformation rules. Computation is done by repeatedly substituting the left-hand side of a rule by the corresponding right-hand side [83].

Hence, in λ-calculus, there is only one computation rule and it is unconditional while, in rewriting, a computation step occurs only if a term *matches* a pattern (possibly modulo some equational theory).

But first-order rewriting cannot express in a simple way anonymous functions or patterns with bound variables. See for instance the works on Combinatory Logic [29], first-order definitions of a substitution operation compatible with α-equivalence [41,3,84] (to cite just a few, for the amount of publications on this subject is very important), or first-order encodings of higher-order rewriting [15].

E-mail address: frederic.blanqui@inria.fr.

[1] Hosted from July 2012 to August 2013 by the Institute of Software of the Chinese Academy of Sciences, Beijing, China.

http://dx.doi.org/10.1016/j.tcs.2015.07.045
0304-3975/

Rewriting on λ-terms, or higher-order rewriting, aims at unifying these two languages. Several approaches exist like Klop's Combinatory Reduction Systems (CRSs) [88,91], Khasidashvili's Expression Reduction Systems (ERSs) [85,59], Nipkow's Higher-order Rewrite Systems (HRSs) [101,99], or Jouannaud and Okada's higher-order algebraic specification languages (HALs) [77,78]. Van Oostrom and van Raamsdonk studied the relations between CRSs and HRSs [133] and developed a general framework (HORSs) that subsumes most of the previous approaches [132,134].

In another direction, some researchers introduced calculi where patterns are first-class citizens: van Oostrom's pattern calculus [131,90], Cirstea and Kirchner's ρ-calculus [35,36], Jay and Kesner's pattern calculus [72,75], or some extensions of ML or Haskell [49,125].

In this paper, I will consider HALs with curried symbols (i.e. all symbols are of arity 0), that is, arbitrary simply typed λ-terms with curried symbols defined by the combination of rewrite rules and β-reduction. But, as we will see in Section 6.5, our results easily apply to HRSs and simply typed CRSs as well.

My goal is to develop techniques for proving the termination of such a system, i.e. the combination of β-reduction and arbitrary user-defined rewrite rules.

For proving the termination of rewrite relations on λ-terms, one can try to extend to λ-calculus techniques developed for first-order rewriting (e.g. [94,129,117,73,51]) or, vice versa, adapt to rewriting techniques developed for λ-calculus (e.g. [77,18,24]).

Since β-reduction does not terminate in general, one usually restricts his attention to some strict subset of the set of all λ-terms, like the set of λ-terms typable in some type system [7] (types were first introduced by logicians as an alternative to the restriction of the comprehension axiom in set theory, and later found important applications in programming languages and compilers).

To prove the termination of β-reduction in typed λ-calculi, there are essentially three techniques:

Direct proof. In the simply-typed λ-calculus, it is possible to prove the termination of β-reduction by induction on the size of the type of the substituted variable [112,128]. For instance, in the reduction sequence $(\lambda x^{A\Rightarrow B} xy)(\lambda y^A z) \to_\beta (\lambda y^A z)y \to_\beta z$, the type in the first reduction step of the substituted variable x is $A \Rightarrow B$ while, in the second reduction step (which is generated by the first one), the type of the substituted variable y is A.

But this technique extends neither to polymorphic types nor to rewriting since, in both cases, the type of the substituted variables may increase:

- With polymorphic types, consider the reduction sequence $(\lambda x^{(\forall\alpha)\alpha\Rightarrow B} xYy)(\Lambda\alpha\lambda y^\alpha z) \to_\beta (\Lambda\alpha\lambda y^\alpha z)Yy \to_\beta (\lambda y^Y z)y \to_\beta z$. In the first reduction step, the type of the substituted variable x is $(\forall\alpha)\alpha \Rightarrow B$ while, in the last reduction step, the type of the substituted variable y is the arbitrary type Y.
- With the rule $\mathsf{K}\ x\ \mathsf{a} \to_{\mathcal{R}} x$ where $\mathsf{K} : T \to \mathsf{A} \to T$, consider the reduction sequence $(\lambda z \mathsf{K} x z)\mathsf{a} \to_\beta \mathsf{K}x\mathsf{a} \to_{\mathcal{R}} x$. In the first reduction step, the type of the substituted variable z is A while, in the second reduction step which is generated by the first one, the type of the substituted variable x is the arbitrary type T.

Interpretation. For the simply-typed λ-calculus again, Gandy showed that λI-terms (λ-terms where, in every subterm $\lambda x t$, x has at least one free occurrence in t), can be interpreted by hereditarily monotone functionals on $\mathbb{N}$ [54]. Then, van de Pol showed that there is a transformation from λ-terms to λI-terms that strictly decreases when there is a β-reduction, and extended this to higher-order rewriting and other domains than $\mathbb{N}$ [129]. Finally, Hamana developed a categorical semantics for terms with bound variables [65] based on the work of Fiore, Plotkin and Turi [52], that is complete for termination (which is not the case of van de Pol's interpretations), and extended to higher-order terms the technique of semantic labeling [66] introduced for first-order terms by Zantema [137]. However, Roux showed that its application to β-reduction itself is not immediate since the interpretation of β-reduction is not β-reduction [25,111].

Computability. The last technique, not limited to simply-typed λ-calculus, is based on Tait and Girard's notions of computability[2] introduced by Tait [118] for the weak normalization of the simply-typed λ-calculus, and extended by Girard to polymorphic types [57] and strong normalization [58].

There are however relations between these techniques. For instance, van de Pol proved that his interpretations on $\mathbb{N}$ can be obtained from a computability proof by adding information on the length of reductions [129]. Conversely, the author and Roux proved that size-based termination [56,1,10,18], which is a refinement of computability, can to some extent be seen as an instance of Hamana's higher-order semantic labeling technique [25].

In this paper, we will consider a technique based on computability.

Computability has been first used for proving the termination of the combination of β-reduction, in the simply typed or polymorphic λ-calculus, together with a first-order rewrite system that is terminating on first-order terms, by Tannen and Gallier [26,27] and Okada [102] independently. It was noticed later by Dougherty that, with first-order rewriting, a proof can be given that is independent of the proof of termination of β-reduction [46,47], because first-order rewriting cannot

[2] In fact, Tait speaks of "convertibility" and Girard of "reducibility". To the best of my knowledge, the expression "computability" is due to Troelstra [124] although Troelstra himself invokes Tait. This notion of computability has to be distinguished from the one of Turing and Church [126]. However, given a Tait-computable λ-term $t : U \Rightarrow V$, the function that maps every Tait-computable λ-term $u : U$ to the normal form of tu is indeed Turing-computable.