Theoretical Computer Science 577 (2015) 1–24

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Term-generic logic

Andrei Popescu [a,*], Grigore Roşu [b]

[a] Department of Computer Science, School of Science and Technology, Middlesex University, UK
[b] Department of Computer Science, University of Illinois at Urbana-Champaign, United States

ARTICLE INFO

Article history:
Received 9 September 2009
Received in revised form 24 January 2015
Accepted 27 January 2015
Available online 4 February 2015
Communicated by D. Sannella

Keywords:
Term-generic logic
Substitution
λ-Calculus
π-Calculus
Semantics

ABSTRACT

We introduce *term-generic logic (TGL)*, a first-order logic parameterized with terms defined axiomatically (rather than constructively), by requiring terms to only provide *free variable* and *substitution* operators satisfying some reasonable axioms. TGL has a notion of model that generalizes both first-order models and Henkin models of the λ-calculus. The abstract notions of term syntax and model are shown to be sufficient for obtaining the completeness theorem of a Gentzen system generalizing that of first-order logic. Various systems featuring bindings and contextual reasoning, ranging from pure type systems to the π-calculus, are captured as theories inside TGL. For two particular, but rather typical instances—untyped λ-calculus and System F—the general-purpose TGL models are shown to be equivalent with standard ad hoc models.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

First-order logic (FOL) does not allow variables to be bound in terms (but only in formulas, via quantifiers), thus providing a straightforward notion of substitution in terms. On the other hand, most calculi and type systems used in programming languages are crucially based on the notion of binding of variables *in terms*: terms "export" only a subset of their variables, the free ones, that can be substituted. Because of their complex formulation for terms, these calculi cannot be naturally captured as FOL theories. Consequently, they need to define their own models and deduction rules, and to state their own theorems of completeness, not always easy to prove. In other words, they are presented as entirely new logics, as opposed to theories in an existing logic, thus incurring all the drawbacks (and boredom) of repeating definitions and proofs following generic, well-understood patterns, but facing new details.

In this paper we define *term-generic first-order logic*, or simply *term-generic logic (TGL)*, as a many-sorted first-order logic parameterized by any terms that come with abstract notions of *free variable* and *substitution*. More precisely, in TGL terms are elements in a generic set *Term* (including a subset *Var* whose elements are called variables) that comes with functions $FV : Term \to \mathcal{P}(Var)$ and $Subst : Term \times Term^{Var} \to Term$ satisfying some axioms, including compositionality for substitution. Terms are classified according to sorts and substitution is compatible with sorting. TGL models provide interpretations of terms, subject to axioms requiring smooth interaction with the syntactic operators (Section 2).

A main contribution of this paper is demonstrating that the above axiomatization is sufficient to develop the fundamental theory of first order-logic up to the completeness theorem. TGL admits a complete Gentzen deduction system, syntactically

* Corresponding author.
E-mail addresses: a.popescu@mdx.ac.uk (A. Popescu), grosu@illinois.edu (G. Roşu).

http://dx.doi.org/10.1016/j.tcs.2015.01.047
0304-3975/© 2015 Elsevier B.V. All rights reserved.

similar to that of FOL; its proof of completeness modifies the classic proof of completeness for FOL to use the generic notions of term, free variables and substitution (Section 3).

TGL can be instantiated to different kinds of terms, such as standard FOL terms or different categories of (typed or untyped) λ-terms. When instantiated to standard FOL terms, TGL becomes, as expected, precisely FOL. However, when instantiated to more complex terms, e.g., the terms of λ-calculus, TGL becomes a logic where a particular calculus is a particular theory.

We give several examples of calculi and type systems that can be described as TGL theories, including the λ-calculus and the π-calculus (Section 4). E.g., the β rule of λ-calculus is represented by the TGL axiom-schema in a language having the terms of λ-calculus and a binary relation symbol $\rightsquigarrow$:

$$(\lambda x.X)\, x \rightsquigarrow X \qquad (*)$$

assumed parameterized by the term X and having all its free variables quantified at the top by an implicit $\forall$. The variable x is bound in the left X by the term operator λ; the occurrences of x outside of $\lambda x.X$, that is, the second listed x and any occurrence of x in the right X, are bound by the top $\forall$ quantifier.

The λ-calculus mechanisms are managed in TGL by FOL-like reasoning mechanisms. Consider the reduction

$$(\lambda x.\, x\, x)\, (\lambda x.\, x) \rightsquigarrow (\lambda x.\, x)\, (\lambda x.\, x)$$

It is deduced in TGL from $(*)$ taking X to be $x\,x$ and instantiating the $\forall$-bound x with $\lambda x.\, x$. As customary in first-order logic, $\forall$-instantiation takes place by substitution, only now one employs the λ-calculus syntax for terms. After instantiation, $\lambda x.X$ stays the same (since $(\lambda x.x\,x)[(\lambda x.\, x)/x] = \lambda x.x\,x$), the second listed x becomes $\lambda x.x$ (since $x[(\lambda x.\, x)/x] = \lambda x.x$), and the right X becomes $(\lambda x.\, x)\, (\lambda x.\, x)$ (since $(x\,x)[(\lambda x.\, x)/x] = (\lambda x.\, x)\, (\lambda x.\, x)$). In general, instantiating in $(*)$ the $\forall$-bound x with arbitrary terms Y, we obtain the more familiar schema used in λ-calculus, $(\lambda x.X)\, Y \rightsquigarrow X[Y/x]$.

Type systems can also be represented in TGL. For example, the following rule for typing abstractions in typed λ-calculi,

$$\frac{\Gamma, x : T \rhd X : T'}{\Gamma \rhd \lambda x{:}T.\, X : T \to T'}[x \text{ fresh for } \Gamma]$$

is represented by the following TGL axiom-schema written in a language with 2 sorts, one for data and one for types, and a binary relation symbol $tpOf$ of arity $data \times type$:

$$(\forall x.\, tpOf(x, t) \Rightarrow tpOf(X, t')) \Rightarrow tpOf(\lambda x{:}t.\, X,\ t \to t')$$

In this schema, x and t, t' are data and type variables, respectively, X is an arbitrary data term (a parameter of the schema), and $\Rightarrow$ is logical implication. The term X may contain the free variable x, which is bound by TGL's $\forall$ in the left of the outermost implication, and by the term syntax's λ in the right.

As illustrated by the above examples, the TGL descriptions are more compact and more high-level than the original presentations of the rules: for β there is no explicit substitution and for typing there is no side-condition or explicit context—all these details are managed by the underlying mechanism of TGL.

A description of a calculus as a TGL theory automatically explains a notion of model for that calculus: the TGL models satisfying the theory. Models for λ-calculi have received much attention in the literature, as witnessed in the monographs [7,25,29]. Apart from their intrinsic interest as the dual of syntax, models are often used to get a better insight into a calculus or a type system, which helps developing its metatheory. For example, Reynold's abstraction theorem (later called parametricity) [43], as well as Wadler's free theorems [52], make essential use of models. More generally, the method of logical relations [46] can be developed systematically using models [29].

A natural question is how insightful are these general-purpose TGL models for the described calculi. Due to the large palette of systems describable in TGL and the bewildering variety of methods and models developed for existing systems, a complete answer to this question cannot be attempted. However, we can identify some characteristics of the TGL models: they offer a loose, set-theoretic semantics to a calculus. We look at two well-known particular cases, considering standard ad hoc models for the λ-calculus and System F falling in this category (Section 5). We show that, in these cases, the free TGL models are equivalent to the ad hoc models.

While it captures various systems with bindings as particular theories, TGL does not address the mechanical representations of the syntax of these systems, suitable for reasoning in a theorem prover. Indeed, TGL does not indicate means to represent syntax with bindings, but is parameterized by such a syntax. Nor does it address the mechanical representation of its axiom-schemas. On the other hand, higher order abstract syntax (HOAS) is a methodology specialized in precisely these two aspects: syntax with bindings and schematic judgments. We discuss the relationship between TGL and HOAS and the potential benefit of combining the two. Most proofs are delegated to Appendices B–D.

This paper extends the WADT'08 conference paper [41] with the following: an analysis of the use of the generic-syntax axioms in the proof of completeness (Section 3); a TGL representation for the π-calculus (Subsection 4.3); a comparison between TGL models and ad hoc models (Section 5); proofs of the facts stated in the paper. One aspect not included in this paper, but discussed in the technical report [40], is a methodology for establishing adequacy of TGL specifications, based on a fragment of the logic called HORN^2 (Horn squared).

Download English Version:

https://daneshyari.com/en/article/6876053

Download Persian Version:

https://daneshyari.com/article/6876053

Daneshyari.com