Proof Systems for Retracts in Simply Typed Lambda Calculus

Colin Stirling

School of Informatics
University of Edinburgh
cps@inf.ed.ac.uk

Abstract. This paper concerns retracts in simply typed lambda calculus assuming $\beta\eta$-equality. We provide a simple tableau proof system which characterises when a type is a retract of another type and which leads to an exponential decision procedure.

1 Introduction

Type $\rho$ is a retract of type $\tau$ if there are functions $C : \rho \rightarrow \tau$ and $D : \tau \rightarrow \rho$ with $D \circ C = \lambda x.x$. This paper concerns retracts in the case of simply typed lambda calculus [1]. Various questions can be asked. The decision problem is: given $\rho$ and $\tau$, is $\rho$ a retract of $\tau$? Is there an independent characterisation of when $\rho$ is a retract of $\tau$? Is there an inductive method, such as a proof system, for deriving assertions of the form “$\rho$ is a retract of $\tau$”? If so, can one also construct (inductively) the witness functions $C$ and $D$?

Bruce and Longo [2] provide a simple proof system that solves when there are retracts in the case that $D \circ C = \beta \lambda x.x$. The problem is considerably more difficult if $\beta$-equality is replaced with $\beta\eta$-equality. De Liguoro, Piperno and Statman [3] show that the retract relation with respect to $\beta\eta$-equality coincides with the surjection relation: $\rho$ is a retract of $\tau$ iff for any model there is a surjection from $\tau$ to $\rho$. They also provide a proof system for the affine case (when each variable in $C$ and $D$ occurs at most once) assuming a single ground type. Regnier and Urzyczyn [9] extend this proof system to cover multiple ground types. The proof systems yield simple inductive nondeterministic algorithms belonging to NP for deciding whether $\rho$ is an affine retract of $\tau$. Schubert [10] shows that the problem of affine retraction is NP-complete and how to derive witnesses $C$ and $D$ from the proof system in [9]. Under the assumption of a single ground type, decidability of when $\rho$ is a retract of $\tau$ is shown by Padovani [8] by explicit witness construction (rather than by a proof system) of a special form.

More generally, decidability of the retract problem follows from decidability of higher-order matching in simply typed lambda calculus [13]: $\rho$ is a retract of $\tau$ iff the equation $\lambda z^\rho. x_1^\tau \rightarrow^\rho (x_2^{\rho \rightarrow \tau} z) = \beta\eta \lambda z^\rho. z$ has a solution (the witnesses $D$ and $C$ for $x_1$, $x_2$). Since the complexity of matching is non-elementary [15] this decidability result leaves open whether there is a better algorithm, or even a proof

1 For a full version see http://www.homepages.inf.ed.ac.uk/cps/ret.pdf
Proof Systems for Retracts in Simply Typed Lambda Calculus

system, for the problem. In the case of \( \beta \)-equality matching is no guide to solvability: the retract problem is simply solvable whereas \( \beta \)-matching is undecidable \( [4] \).

In this paper we provide an independent solution to the retract problem. We show it is decidable by exhibiting sound and complete tableau proof systems. We develop two proof systems for retracts, one for the (slightly easier) case when there is a single ground type and the other for when there are multiple ground types. Both proof systems appeal to paths in terms. Their correctness depend on properties of such paths. We appeal to a dialogue game between witnesses of a retract to prove such properties: a similar game-theoretic characterisation of \( \beta \)-reduction underlies decidability of matching.

In Section 2 we introduce retracts in simply typed lambda calculus and fix some notation for terms as trees and for their paths. The two tableau proof systems for retracts are presented in Section 3 where we also briefly examine how they generate a decision procedure for the retract problem. In Section 4 we sketch the proof of soundness of the tableau proof systems (and completeness and further details are provided in the full version).

2 Preliminaries

Simple types are generated from ground types using the binary function operator \( \to \). We let \( a, b, o, \ldots \) range over ground types and \( \rho, \sigma, \tau, \ldots \) range over simple types. Assuming \( \to \) associates to the right, so \( \rho \to \sigma \to \tau \) is \( \rho \to (\sigma \to \tau) \), if a type \( \rho \) is not a ground type then it has the form \( \rho_1 \to \ldots \to \rho_n \to a \). We say that \( a \) is the target type of \( a \) and of any type \( \rho_1 \to \ldots \to \rho_n \to a \).

Simply typed terms in Church style are generated from a countable set of typed variables \( x^\sigma \) using lambda abstraction and function application \( [1] \). We write \( S^\tau \), or sometimes \( S : \sigma \), to mean term \( S \) has type \( \sigma \). The usual typing rules hold: if \( S^\tau \) then \( \lambda x^\sigma.S^\tau : \sigma \to \tau \); if \( S^\sigma \to \tau \) and \( U^\sigma \) then \( (S^\sigma \to \tau U^\sigma) : \tau \). In a sequence of unparenthesised applications we assume that application associates to the left, so \( SU_1 \ldots U_k \) is \( (((S)U_1)\ldots)U_k \). Another abbreviation is \( \lambda z_1 \ldots z_m \) for \( \lambda z_1 \ldots \lambda z_m \). Usual definitions of when a variable occurrence is free or bound and when a term is closed are assumed.

We also assume the usual dynamics of \( \beta \) and \( \eta \)-reductions and the consequent \( \beta\eta \)-equivalence between terms (as well as \( \alpha \)-equivalence). Confluence and strong normalisation ensure that terms reduce to (unique) normal forms. Moreover, we assume the standard notion of \( \eta \)-long \( \beta \)-normal form (a term in normal form which is not an \( \eta \)-reduct of some other term) which we abbreviate to lnf. The syntax of such terms reflects their type: a lnf of type \( a \) is a variable \( x^a \), or \( x U_1 \ldots U_k \) where \( x^\rho_1 \to \ldots \to x^\rho_n \to a \) and each \( U_i^\rho_i \) is a lnf; a lnf of type \( \rho_1 \to \ldots \to \rho_n \to a \) has the form \( \lambda x_1^a \ldots x_n^a .S \), where \( S^a \) is a lnf.

The following definition introduces retracts between types \( [2,3] \).

**Definition 1.** Type \( \rho \) is a retract of type \( \tau \), written \( \models \rho \preceq \tau \), if there are terms \( C : \rho \to \tau \) and \( D : \tau \to \rho \) such that \( D \circ C \equiv_{\beta\eta} \lambda x^\rho .x \).