An implementation of Simply Typed Lambda Calculus in Coq

Tim Bruyn

July 1, 2018

Introduction

Simply Typed Lambda Calculus (STLC) is the most basic lambda calculus with a type system ...

For this project the standard STLC was implemented with the addition of conjunction or product type. The system implemented is described by the following grammar:

$$A, B ::= X | A \to B | A \times B \tag{1}$$
$$M, N ::= x | MN | \lambda x : A.M | (M, N) | fst\ M | snd\ M \tag{2}$$
$$\Gamma ::= \emptyset | \Gamma, x : A \tag{3}$$

Exercises

This project was implemented with in the Coq proof management system and with the use of the Coq IDE. The code can be found in the appendix and on the GitHub repository `https://github.com/TBruyn/Software-Verification`.

Definitions of types and terms

The inductive definition of type contains three different items: tvar for type variables, tarr for implications and tprod for products.

The six term types as described in the grammar in the introduction were implemented: var for the variable types, app for function application, lam for functions, prod for products and fst and snd to retrieve the first and second item of a product pair.

The context was implemented as a partial_map of Types as described in the map.v Coq file.

Examples of types and terms

To test the implementation of the type, term and context definitions and to gain practice with the way they are applied together, two types and the terms that inhabit them were implemented. The terms and types are described in table 1. The Coq implementations can be found under the header *Exercise 2.1* and

Table 1: Two types and the terms that inhabit them

Types	*Terms*
$X \to Y \to X$	$\lambda x : X.\lambda Y.x$
$(X \to X) \to (X \to X)$	$\lambda f : X \to X.\lambda x : X.f(fx)$

Exercise 2.2

The typing judgement

The typing judgment $\Gamma \vdash M : A$ was implemented as the inductively defined proposition *typed* of type $ctx \to term \to type \to Prop$. The definition can be found under the header *Exercise 2.3*

As a test for the validity of *typed*, it was proven that $\lambda f : X \to X.\lambda x : X.f(fx) : (X \to X) \to (X \to X)$.

The typechecker

To implement the type checker, a helper function *beq_type* $: A \to B \to bool$ for type equality was defined. It compares the strings of type variables and for implications and products the composing types were compared. It was proven that it is reflexive, i.e. $beq\_typeAA = true$, by relying on the proposition $beq\_string\_true\_iff : \forall xy : string, beq\_stringxy = true \rightleftarrows x = y$ and induction on the types A and B. Furthermore, it was proven that *beq_type* corresponds to true equality, i.e. $\forall AB, beq\_typeAB \to A = B$. This was done by again using *beq_string_true_iff* on the variable term and induction.

The type checker $typecheck(E : ctx)(t : term) : optiontype$ matches on the term t. When the type of term is determined (i.e. var, app, lam, etc.), it is checked whether the components match the prerequisites as described by the typing rules. If so, it will return the type given by the typing rule and if not it signal that the term cannot be typed. The function *beq_type* is used to type check the application rule $(A \to B) \to A \to B$, as it checks the type of $M : A \to B$ and $N : C$ to assert that $A = C$.

To test the type checker it was proven that for the terms of table 1, the types of table 1 were computed. Furthermore, it was proven that $\lambda x : X.xx$ and $\lambda f : X \to X.\lambda x : X.xf$ cannot be typed in STLC.

The difference between the typing judgment and the type checker

The typing judgment *typed* and the type checker *typecheck* are strongly related, but perform different functions. *typed* is the proposition that for a given context, a given term is of the given type. As it is a proposition, it is a statement that can be proven. The type checker is a function that for a given context and term deduces the associated type.

Naturally, if the type checker finds a type A for a term t, the typing judgment $\Gamma \vdash t : A$ should be true. Vice versa, if for a term s and a type B the typing judgment can be proven to be true, the type checker should produce B on input s. The first is known as completeness, which means the type system will never reject a term that is of a valid type, and the second is known as soundness, which means that if the typing system never accepts an invalid term. The proof of both properties is discussed in the next two sections.

Completeness

To prove completeness, the theorem
$typecheck\_complete : \forall E\ t\ a, typed\ E\ t\ a \rightarrow typecheck\ E\ t = Some\ A$ is proven.

Soundness

Experiences with Coq