Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Dipl.-Inf. Achim D. Brucker
Dr. Burkhart Wolff

Computer-supported Modeling and Reasoning

http://www.infsec.ethz.ch/education/permanent/csmr/
(rev. 16814)
Submission date: –

λ-Calculus

In this exercise, we will will use Isabelle as a prototype tool to describe calculi (including binding) and to perform computations in them by using tactics involving backtracking. This will also deepen our understanding of the unification procedures used by Isabelle.

We will also introduce the concept of (parametric) Polymorphism which can be used to encode object languages including their type system.

1. Isabelle

1.1. The Context of this Exercise

In lecture "The λ-Calculus", we defined the syntax of the untyped λ-calculus by the following grammar: untyped λ-calculus

$$e ::= x \mid c \mid (ee) \mid (\lambda x.\, e)$$

together with conventions of left-associativity and iterated λ's in order to avoid cluttering the notation. Later, we defined a substitution on this raw syntax, and congruence relations on λ-terms such as α-, β- and η congruences.

In this exercise, we will use a particular representation technique for the untyped λ-calculus called *shallow embedding*. It can be found in the- *shallow embedding* ory `http://www.infsec.ethz.ch/education/permanent/csmr/material/Lambda.thy` (which is based on FOL for purely technical reasons - the declaration part can be loaded even in `Pure`, the meta logic of Isabelle itself). Instead of e, we declare one universal type term — the presented calculus is thus untyped. The application is represented by the constant declaration "^" :: "[term, term] ⇒ term", consequently. Instead of defining an own substitution function, however, we define the abstraction as a constructor of a function; thus, it gets the type Abs :: "[term ⇒ term] ⇒ term" where ⇒ is the function space inherited from `Pure`. The notation lam x. P x is equivalent to Abs(λx . P x); recall that λ is the internal abstraction inherited from Isabelle/Pure. Thus, whenever we want to substitute a term into the body of an abstraction, we can just use the β-reduction provided by Isabelle/Pure (one also speaks of an "internalized" substitution provided by the shallow embedding of our language; or of using higher-order abstract syntax).

Lambda.thy

term

^

Abs

lam x. P x

higher-order abstract syntax

Our theory for the untyped λ-calculus also provides the β-reduction relation and the β-congruence by a set of axioms; note that we make no claims on the logical consistency of this exercise!

Further, it provides definitions for the standard combinators K,S and I and two versions of Y combinators.

In lecture it was said that the untyped λ-calculus is Turing-complete. We will show two core ingredients for such a proof: namely that data types (in particular: natural numbers) and fix-point combinators (enabling the presentation of recursive functional programs) can be represented inside the untyped λ-calculus.

1.2. Automated Proof Search Tactics

As mentioned in the lecture "Proof Search", Isabelle can organize proof-states in a tree-like fashion, which can therefore be searched according to depth-first or breadth-first strategies. The tactic command fast performs the former, according to introduction and elimination rules given to it. Introduction and Elimination rules are both subdivided into two classes:

safe rules

1. *safe rules*, which transform a proof state into an equivalent one,

unsafe rule

2. *unsafe rule*, which may transform a proof state into a logically weaker one.

Unsafe rules where tried in a limited way after safe rules did not succeed, and assumption is applied after no more unsafe rule applications are possible. Some syntactic variants for fast-commands are:

fast intro

fast elim

fast intro: *rules*
fast elim: *rules*

If the full context of assumptions should be included as well, one can append a ! to intro, elim, and dest, e.g.:

fast intro!: *rules*

2. Exercises

2.1. Exercise 18

As a warm-up, reduce the following terms to β-normal form in Isabelle.

1. SKK
2. SKS

Hint: Start with

lemma ex18_1: "S^K^K >−−> ?x"

In the end, the metavariable ?x should be instantiated to a term in β-normal form.

Hint: Do the proofs without using fast.

2.2. Exercise 19

Automate the proofs from Ex. 18 using fast and the ISAR control structures. Thanks to automation, you should be able to show also the following reductions using the identical "proof script":

1. $SKKISS$
2. $SKIKISS$

2.3. Exercise 20

Now show in Isabelle that for both Y-combinator versions enjoy a fix-point property, i.e. prove that:

1. $Y_T F >=< F(Y_T F)$ and
2. $Y_C F >=< F(Y_C F)$.

Is it possible to show $Y_T F$-->$F(Y_T F)$ and $Y_C F$-->$F(Y_C F)$?

2.4. Exercise 21

Following a proposal by Alonzo Church, natural numbers n were encoded as the term

$$\lambda f x.\ \underbrace{f(f \ldots (f}_{n \text{ times}}\, x) \cdots)\ ,$$

which we abbreviate by writing $\lambda f x.\ f^n x$. The successor function and addition are given by the λ-terms:

$$\begin{aligned} succ &\equiv \lambda u f x.\ f(u f x) \\ add &\equiv \lambda u v f x.\ u f(v f x) \end{aligned}$$

Write a theory of the Church-Numerals with constants for $C0$,$C1$,$C2$ and *succ* and *add*.

Convince yourself that *succ* and *add* are indeed the successor and addition function, by evaluating them symbolically (i.e, on "terms" $\lambda f x.\ f^n x$ and $\lambda f x.\ f^m x$) under a suitable assumption.

2.5. Exercise 22

Reduce the following terms:

1. *succ* C_0
2. *add* C_3 C_2

2.6. Exercise 23 (*optional*)

When applying a rule, Isabelle uses a process that is called *higher-order unification* for finding instantiations for meta-variables. Consider the unification problem

$$\texttt{?P(?b)} \ =_{\alpha\beta\eta}\ \texttt{y} = \texttt{x}$$

which has the solutions:

$$\begin{aligned} &[\texttt{?P} \leftarrow (\lambda \texttt{z}.\ \texttt{z} = \texttt{x}),\ \texttt{?b} \leftarrow \texttt{y}] \\ &[\texttt{?P} \leftarrow (\lambda \texttt{z}.\ \texttt{y} = \texttt{z}),\ \texttt{?b} \leftarrow \texttt{x}] \\ &[\texttt{?P} \leftarrow (\lambda \texttt{z}.\ \texttt{y} = \texttt{x}),\ \texttt{?b} \leftarrow t] \qquad \text{(for any } t\text{)} \end{aligned}$$

We can simulate higher-order unification inside `Lambda.thy` on the basis of ?P ^ ?x >=< add ^ C3 ^ C4.

1. Synthesize at least two solutions. You may use local substitutions or back.
2. Try to unify lam x. add ^ ?P ^ C4 >=< lam x. add ^ x ^ C4 and lam x. add ^ (?P ^ x) ^ C4 >=< lam x. add ^ x ^ C4

A. Encoding the untyped λ-calculus in in Isabelle

```

theory Lambda = FOL:

(* common definition for both calculi *)
typedecl
  "term"

arities
  "term" :: logic

consts
  Abs     :: "[term ⇒ term] ⇒ term"     (binder "lam " 10)
  "^"     :: "[term, term] ⇒ term"      (infixl 20)

  K       :: "term"
  I       :: "term"
  S       :: "term"

  B       :: "term"

  YC      :: "term"
  YT      :: "term"

defs
  K_def:  "K ≡lam x. (lam y. x)"
  I_def:  "I ≡lam x. x"
  S_def:  "S ≡lam x. (lam y. (lam z. x^z^(y^z)))"

  B_def:  "B ≡S^(K^S)^K"

  YC_def: "YC ≡lam f. ((lam x. f^(x^x))^(lam x. f^(x^x)))"
  YT_def: "YT ≡(lam z. lam x. x^(z^z^x))^(lam z. lam x. x^(z^z^x))"


  (* reduction λ−calculus *)
consts
  Red     :: "[term, term] ⇒ prop"      ("(_ >−−> _)")

axioms
  beta:   "(lam x. f(x))^a >−−> f(a)"
  refl:   "M >−−> M"
  trans:  "⟦ M >−−> N; N >−−> L ⟧⟹M >−−> L"
  appr:   "M >−−> N ⟹M^Z >−−> N^Z"
  appl:   "M >−−> N ⟹Z^M >−−> Z^N"
  epsi:   "⟦ !!x. M(x) >−−> N(x) ⟧⟹(lam x. M(x)) >−−> (lam x. N(x))"

  (* equational λ−calculus *)
consts
  Conv :: "[term, term] ⇒ prop"      ("(_ >=< _)")

axioms
  beta_sym:   "(lam x. f(x))^a >=< f(a)"
  refl_sym:   "M >=< M"
  symm_sym:   "M >=< N ⟹N >=< M"
  trans_sym:  "⟦ M >=< N; N >=< L ⟧⟹M >=< L"
  appr_sym:   "M >=< N ⟹M^Z >=< N^Z"
  appl_sym:   "M >=< N ⟹Z^M >=< Z^N"
  epsi_sym:   "⟦!!x. M(x) >=< N(x) ⟧⟹lam x. M(x) >=< lam x. N(x)"

  (* syntax setup *)
syntax (symbols)
  "lam "        :: "[idts, term] ⇒ term"          ("(3λ_./ _)" [0, 10] 10)

end
```