Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Johannes Hölzl and Lars Hupel

Equational Logic
Summer Term 2015
Solutions to Exercise Sheet 11

Exercise 55 (Lists in λ-calculus)

Specify λ-Terms for nil, cons, hd, tl and null, that enocde lists in the λ-calculus. Show that your terms satisfy the following conditions:

$$\begin{array}{llll} \mathsf{null}\ \mathsf{nil} & \longrightarrow^* \mathsf{true} & \mathsf{hd}\ (\mathsf{cons}\ x\ l) & \longrightarrow^* x \\ \mathsf{null}\ (\mathsf{cons}\ x\ l) & \longrightarrow^* \mathsf{false} & \mathsf{tl}\ (\mathsf{cons}\ x\ l) & \longrightarrow^* l \end{array}$$

Hint: Use pairs.

Solution

A list is represented as a pair of its head and its tail. We define an encoding of pairs and booleans as below:

$$\begin{array}{lll} \mathsf{true} := \lambda x\ y.\ x & \mathsf{false} := \lambda x\ y.\ y & \\ \mathsf{pair} := \lambda a\ b\ f.\ f\ a\ b & \mathsf{fst} := \lambda p.\ p\ (\lambda x\ y.\ x) & \mathsf{snd} := \lambda p.\ p\ (\lambda x\ y.\ y) \end{array}$$

We only specify the solution without tagging here:

$$\begin{array}{rr} \mathsf{nil} := \mathsf{false} & \mathsf{cons} := \mathsf{pair} \\ \mathsf{hd} := \mathsf{fst} & \mathsf{tl} := \mathsf{snd} \\ \mathsf{null} := \lambda l.\ l\ (\lambda h\ t\ d.\ \mathsf{false})\ \mathsf{true} & \end{array}$$

We can now show the above conditions:

$$\begin{array}{rcl} \mathsf{null}\ \mathsf{nil} & = & (\lambda l.\ l\ (\lambda h\ t\ d.\ \mathsf{false})\ \mathsf{true})\ \mathsf{false} \\ & \longrightarrow^* & \mathsf{false}\ (\lambda h\ t\ d.\ \mathsf{false})\ \mathsf{true} \\ & = & (\lambda x\ y.\ y)\ (\lambda h\ t\ d.\ \mathsf{false})\ \mathsf{true} \\ & \longrightarrow^* & \mathsf{true} \\ \mathsf{null}\ (\mathsf{cons}\ x\ l) & = & (\lambda l.\ l\ (\lambda h\ t\ d.\ \mathsf{false})\ \mathsf{true})\ (\mathsf{pair}\ x\ l) \\ & =_\alpha & (\lambda l'.\ l'\ (\lambda h\ t\ d.\ \mathsf{false})\ \mathsf{true})\ (\mathsf{pair}\ x\ l) \\ & \longrightarrow^* & (\mathsf{pair}\ x\ l)\ (\lambda h\ t\ d.\ \mathsf{false})\ \mathsf{true} \\ & = & ((\lambda a\ b\ f.\ f\ a\ b)\ x\ l)\ (\lambda h\ t\ d.\ \mathsf{false})\ \mathsf{true} \\ & \longrightarrow^* & (\lambda f.\ f\ x\ l)\ (\lambda h\ t\ d.\ \mathsf{false})\ \mathsf{true} \\ & \longrightarrow^* & (\lambda h\ t\ d.\ \mathsf{false})\ x\ l\ \mathsf{true} \\ & \longrightarrow^* & \mathsf{false} \end{array}$$

$$
\begin{array}{rcl}
\mathsf{hd}\ (\mathsf{cons}\ x\ l) & = & \mathsf{fst}\ (\mathsf{pair}\ x\ l) \\
& = & (\lambda p.\ p\ (\lambda x\ y.\ x))\ ((\lambda a\ b\ f.\ f\ a\ b)\ x\ l) \\
& \longrightarrow^* & ((\lambda a\ b\ f.\ f\ a\ b)\ x\ l)\ (\lambda x\ y.\ x) \\
& \longrightarrow^* & (\lambda f.\ f\ x\ l)\ (\lambda x\ y.\ x) \\
& \longrightarrow^* & (\lambda x\ y.\ x)\ x\ l \\
& \longrightarrow^* & x \\
\mathsf{tl}\ (\mathsf{cons}\ x\ l) & = & \mathsf{snd}\ (\mathsf{pair}\ x\ l) \\
& = & (\lambda p.\ p\ (\lambda x\ y.\ y))\ ((\lambda a\ b\ f.\ f\ a\ b)\ x\ l) \\
& \longrightarrow^* & ((\lambda a\ b\ f.\ f\ a\ b)\ x\ l)\ (\lambda x\ y.\ y) \\
& \longrightarrow^* & (\lambda f.\ f\ x\ l)\ (\lambda x\ y.\ y) \\
& \longrightarrow^* & (\lambda x\ y.\ y)\ x\ l \\
& \longrightarrow^* & l
\end{array}
$$

Exercise 56 (Fixed-point Combinator)

Use a fixed-point combinator to compute the length of lists.

Solution

We use the Y-combinator:

$$\mathsf{y} := \lambda f.\ (\lambda x.\ f\ (x\ x))\ (\lambda x.\ f\ (x\ x))$$

The Y-combinator satisfies the property $\mathsf{y}\ f \longrightarrow^* f\ (\mathsf{y}\ f)$.

Recall how the Church numerals are implemented:

$$\mathsf{zero} := \lambda f\ x.\ x \qquad\qquad \mathsf{succ} := \lambda n\ f\ x.\ f\ (n\ x)$$

In a programming language with recursion, length would be implemented as follows:

```
len x = if null x then 0 else Succ (len (tl x))
```

We obtain the following definition:

$$\mathsf{len} := \mathsf{y}\ (\lambda l\ x.\ (\mathsf{null}\ x)\ \mathsf{zero}\ (\mathsf{succ}\ (l\ (\mathsf{tl}\ x))))$$

Exercise 57 (β-reduction on de Bruijn preserves substitution)

Prove Lemma 1.2.5 for de-Bruijn terms:

$$s \to_\beta s' \implies s[u/x] \to_\beta s'[u/x]$$

Solution

See `Ex57.thy`.

Homework 58 (Trees in λ-calculus)

Encode a datatype of binary trees in lambda calculus. Specify the operations tip and node that construct trees, as well as isTip, left, right, and value. Each tip should carry a value, whereas each node should consist of two subtrees.

Show that the following holds:

$$\begin{aligned}
\mathsf{isTip}\ (\mathsf{tip}\ a) &\longrightarrow^* \mathsf{true} \\
\mathsf{isTip}\ (\mathsf{node}\ x\ y) &\longrightarrow^* \mathsf{false} \\
\mathsf{value}\ (\mathsf{tip}\ a) &\longrightarrow^* a \\
\mathsf{left}\ (\mathsf{node}\ x\ y) &\longrightarrow^* x \\
\mathsf{right}\ (\mathsf{node}\ x\ y) &\longrightarrow^* y
\end{aligned}$$

Homework 59 (β-reduction preserves types)

A type system has the *subject reduction property* if evaluating an expression preserves its type. Prove that the simply typed λ-calculus ($\lambda^{\to}$) has the subject reduction property:

$$\Gamma \vdash t : \tau \wedge t \longrightarrow_\beta t' \implies \Gamma \vdash t' : \tau$$

Hints: Use induction over the inductive definition of $\longrightarrow_\beta$ (Def. 1.2.2). State your inductive hypotheses precisely – it may help to introduce a binary predicate $P(t, t')$ to express the property you are proving by induction. Also note that the proof will require *rule inversion*: Given $\Gamma \vdash t : \tau$, the shape of t (variable, application, or λ-abstraction) may determine which typing rule must have been used to derive the typing judgment.

Within your proof, you are free to use the following lemma about substitution:

$$\Gamma \vdash u : \tau_0 \wedge \Gamma[x : \tau_0] \vdash t : \tau \implies \Gamma \vdash t[u/x] : \tau \tag{1}$$