Homework 5
CS420/KAIST Spring 1999
이 광근
Due: April 22 (목요일), 24:00

1999년 4월 14일

Exercise 1 "Substitution"

In implementing the beta reduction for the Lambda Calculus, the key operation is to apply a substitution to a Lambda expression.

In general, substitution S is written as

$$S = \{e_1/x_1, \cdots e_n/x_n\}.$$

Applying the substitution S to a term e, written as $S(e)$ replaces x_i's in e by e_i's. Note that for a substitution to be well-defined, all the variables (x_i's) have to be distinct. We write $\mathit{supp}(S)$ (the support set of S) for the substitutee variables $\{x_1, \cdots, x_n\}$. Composition SR of two substitutions S and R is defined as

$$SR = \{S(R(x))/x | x \in \mathit{supp}(R)\} \cup \{S(x)/x | x \in \mathit{supp}(S) \setminus \mathit{supp}(R)\}.$$

Now we define the substitution operation for Lambda Calculus, as we discussed in class:

$$\begin{array}{lcl}
\texttt{subst:}\ \ \texttt{substitution * lexpr -> lexpr} & & \\
\texttt{subst}(S, x) & = & S(x) \\
\texttt{subst}(S, \lambda x.e) & = & \lambda y.(S\{y/x\}(e)) \\
 & & \text{where } y \notin \mathit{vars}(S) \\
\texttt{subst}(S, e_1\, e_2) & = & (\texttt{subst}(S, e_1))\, (\texttt{subst}(S, e_2))
\end{array}$$

The $\mathit{vars}(S)$ is

$$\mathit{vars}\{e_1/x_1, \cdots, e_n/x_n\} = \{x_1, \cdots, x_n\} \cup \mathrm{fv}(e_1) \cup \cdots \cup \mathrm{fv}(e_n).$$

The new name y in the case of $\texttt{subst}(S, \lambda x.e)$ is in order to avoid the situation where a free variable x of e' in $e'/y \in S$ becomes bound after y in $\lambda x.e$ is replaced by e'. That is,

$$\texttt{subst}(\lambda y.xy, \{y/x\}) = \lambda z.yz \quad \text{instead of } \lambda y.yy.$$

You have to implement the above `subst` function. The types of the Lambda expression and the subtitution are:

```
datatype lexpr = VAR of string
               | LAM of string * lexpr
               | APP of lexpr * lexpr
type substitution = (string * lexpr) list
```

□

Exercise 2 "Normal-order reduction"

Now using the `subst`, we can define the normal-order reduction:

```
nored:  lexpr -> lexpr
```

Remind that normal-order reduction is the reduction rule that always reduces the left-most outer-most redex. Remind also that a redex is an application expression whose left-hand expression is a lambda expression:

$$(\lambda x.e)e' \quad \text{reduces to} \quad \texttt{subst}(\{e'/x\}, e).$$

Using the above `nored`, define the `reduceit` function

```
reduceit:  string -> unit
```

that `reduceit(''in'')` parses a Lambda expression in file `in`, normal-order reduces it, and pretty-prints the result.

□

Exercise 3 "De Bruijn index"

In Exercise 1 you implemented a substitution that has to find new names during beta-reduction in order to avoid the boring problem of name clashes. De Bruijn (read "brooin") invented an elegant encoding scheme that lets us free from worrying about the name clashes. He observed that the only important information about a variable is its *binding height*, i.e. the number of λ's between it and (not including) the binding one. Then the variable names are replaced by this number. For example

$$\lambda y.(\lambda x.\lambda y.x)y \quad \text{is} \quad \lambda.(\lambda.\lambda.1)0.$$

De Bruijn showed that this gives rise to an elegant treatment of beta-reduction.

Define the `bruijnize` function:

```
bruijnize:  lexpr -> bexpr
```

where the the given Lambda expression is closed. The datatype for Bruijn expression is:

```
datatype bexpr = I of int | LAM of bexpr | APP of bexpr * bexpr
```

□

Exercise 4 "DeBruijn's normal-order reduction"

Now you can define the beta reduction rules for Lambda expressions in the De Bruijn forms. You will enjoy a simpler `subst` function

```
subst:  substitution * bexpr -> bexpr
```

that has no extra book-keeping to avoid the name clashes. Using the above simpler substitution function you can define the reduction `noredBruijn` function for De Bruijn forms:

```
noredBruijn:  bexpr -> bexpr.
```

Now define the `reduceitBruijn` function:

```
reduceitBruijn:  string -> unit
```

which is analogous to the `reduceit` function of Exercise 2. □

Challenge 1 "functorize!"

You might have observed that you could do the exercises 2 and 4 by defining and using a functor, say, `NormalReduce`, which is parameterized with things *including* the types for `lexpr` and `substitution` and the `subst` function. The functor body has the definitions of the `nored` and `reduceit` functions. For Exercise 4 this functor can be applied to a structure that handles lambda expressions encoded as De Bruijn forms.

Define the functor `NormalReduce` and show that exercises 2 and 4 are done with this functor.

This challenge exercise worths 50 points. □