

Programming Languages — 604, spring 2008

Assignment 3

February 22, 2009

General rules

You can return your answers via *CSNet*. Your answers to exercises 1, 3, 4, 5, and 6 you can also, or alternatively, bring to class. Hand-written answers are fine, answers typeset with, say, \LaTeX are even finer. This is a perfect excuse to learn \LaTeX if you do not know it already. If you submit via *CSNet*, answers to exercises 1, 3, 4, 5, and 6 should be in one file `answers.pdf`. Late penalty is calculated in the same manner than for Assignment 1; deadline is announced in class and/or on the class web pages.

Enjoy!

Assignment

1. Consider the language \mathcal{NB} we used in class:

t	::=	v $\text{if } t_1 \text{ then } t_2 \text{ else } t_3$ $\text{succ } t$ $\text{pred } t$ $\text{iszero } t$	<i>terms:</i> <i>value</i> <i>conditional</i> <i>successor</i> <i>predecessor</i> <i>test for zero</i>
v	::=	true false nv	<i>values:</i> <i>constant true</i> <i>constant false</i> <i>numeric value</i>
nv	::=	0 $\text{succ } nv$	<i>numeric values:</i> <i>zero value</i> <i>successor value</i>
T	::=	Bool Nat	<i>types:</i> <i>the Boolean type</i> <i>the type of numeric values</i>

and its small step evaluation relation:

$$\begin{array}{c}
\text{E-ISZERO} \\
\frac{t \rightarrow t'}{\text{iszero } t \rightarrow \text{iszero } t'} \\
\text{E-ISZEROZERO} \quad \text{iszero } 0 \rightarrow \text{true} \\
\text{E-ISZEROSUCC} \quad \text{iszero } (\text{succ } nv) \rightarrow \text{false} \\
\text{E-SUCC} \\
\frac{t \rightarrow t'}{\text{succ } t \rightarrow \text{succ } t'} \\
\text{E-PRED} \\
\frac{t \rightarrow t'}{\text{pred } t \rightarrow \text{pred } t'} \\
\text{E-PREDZERO} \quad \text{pred } 0 \rightarrow 0 \\
\text{E-PREDSUCC} \quad \text{pred } (\text{succ } nv) \rightarrow nv \\
\text{E-IFTRUE} \quad \text{if true then } t_2 \text{ else } t_3 \rightarrow t_2 \\
\text{E-IFFALSE} \quad \text{if false then } t_2 \text{ else } t_3 \rightarrow t_3 \\
\text{E-IF} \\
\frac{t_1 \rightarrow t'_1}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \rightarrow \text{if } t'_1 \text{ then } t_2 \text{ else } t_3}
\end{array}$$

and the typing rules:

$$\begin{array}{c}
\text{T-TRUE} \quad \text{true} : \text{Bool} \\
\text{T-FALSE} \quad \text{false} : \text{Bool} \\
\text{T-IF} \\
\frac{t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \\
\text{T-ZERO} \quad 0 : \text{Nat} \\
\text{T-SUCC} \\
\frac{t : \text{Nat}}{\text{succ } t : \text{Nat}} \\
\text{T-PRED} \\
\frac{t : \text{Nat}}{\text{pred } t : \text{Nat}} \\
\text{T-ISZERO} \\
\frac{t : \text{Nat}}{\text{iszero } t : \text{Bool}}
\end{array}$$

Extend (and change) the language as follows:

- add a binary addition operator $+$ for Nats.
- add the operators \wedge , \vee , and \neg for logical *and*, *or*, and *not*, respectively. The \wedge and \vee operators should only evaluate their right-hand argument if necessary—that is, they should be “short-circuited”.
- add the value **error**. An attempt to apply the `pred` operator to `0` should result in **error** (note that this is different from how we define the language above, so you must change the existing rules some). The **error** value as an argument of any other operator should result in **error**. Think carefully how you incorporate the **error** term in the type system.

Examples:

```

if (pred 0) then true else true → error
if true then 0 else (pred 0) → 0
if true then false else (pred 0) // type error

```

Please show the definitions of the grammar, the evaluation relation, and the typing relation.

- Implement a type checker and an evaluator for the extended (and changed) language that you defined in the previous exercise. You can use the type checker and evaluator for \mathcal{NB} provided in a separate file as the starting point. Name the type constructors of your new terms as: `TAdd`, `TAnd`, `TOr`, `TNot`, and `TError`.
- What are the normal forms of these lambda expressions in untyped lambda calculus? Use the evaluation order defined by the reduction rules given in the appendix. If there is no normal form, write “no normal form”.

- $(\lambda y. (\lambda z. x y)) (\lambda x. z)$

- (b) $(\lambda y. (\lambda z. z z) y) (\lambda x. x x)$
- (c) $(\lambda x. (\lambda x. x x) (\lambda x. x x))$

What are the *free* variable occurrences in these terms:

- (a) $(\lambda y. (\lambda z. x y)) (\lambda x. z)$
- (b) $(\lambda x. (\lambda y. y x) (\lambda x. x x))$

4. Reduce the lambda expression: plus $c_2 c_3$ and convince yourself that the result is c_5 . Definitions:

$$\mathbf{plus} = \lambda m. \lambda n. \lambda s. \lambda z. m s (n s z)$$

$$c_2 = \lambda s. \lambda z. s (s z)$$

$$c_3 = \lambda s. \lambda z. s (s (s z))$$

...

$$c_5 = \lambda s. \lambda z. s (s (s (s (s z))))$$

- 5. Show that $Y_2 = (\lambda x y. y (x x y))(\lambda x y. y (x x y))$ is a fixed point operator.
- 6. A substitution $[N/x]M$ is called *valid* if no free variable in N become bound in $[N/x]M$. Argue that if substitutions in α or β -conversions are allowed to be invalid, any two λ -expressions can be shown to be equal. Hint: an invalid α -conversion can make $\lambda x. \lambda y. x$ and $\lambda y. \lambda y. y$ equal.

Untyped lambda-calculus

Syntax

$t ::=$
 x
 $\lambda x. t$
 $t t$

$v ::=$
 $\lambda x. t$

Evaluation

$$\frac{t_1 \longrightarrow t'_1}{t_1 t_2 \longrightarrow t'_1 t_2} \quad \text{(E-APP1)}$$
$$\frac{t_2 \longrightarrow t'_2}{v_1 t_2 \longrightarrow v_1 t'_2} \quad \text{(E-APP2)}$$
$$(\lambda x. t_{12}) v_2 \longrightarrow [x \mapsto v_2] t_{12} \quad \text{(E-APPABS)}$$

terms

variable
abstraction
application

values

abstraction value

$t \longrightarrow t'$