

BU CAS CS 520 (FALL SEMESTER, 2005)
PRINCIPLES OF PROGRAMMING LANGUAGES

Assignment 3

Out: Friday, 14 October 2005
Due: Monday, 24 October 2005

Total: 200 points

The following is the syntax of the language STLC0 and its static and dynamic semantics.

Syntax

constants	$c ::= \text{true} \mid \text{false} \mid 0 \mid 1 \mid -1 \mid \dots$
operators	$op ::= + \mid - \mid * \mid / \mid \dots$
terms	$t ::= c \mid x \mid \text{if } t_0 \text{ then } t_1 \text{ else } t_2 \mid op(t) \mid \lambda x : T. t \mid t_1(t_2)$
values	$v ::= c \mid \lambda x : T. t$
types	$T ::= Bool \mid Int \mid T_1 \rightarrow T_2$
contexts	$\Gamma ::= \emptyset \mid \Gamma, x : T$

Static Semantics

$$\frac{c \in \{\text{true}, \text{false}\}}{\Gamma \vdash c : Bool} \text{ (T-Bool)}$$
$$\frac{c \in \{0, 1, -1, \dots\}}{\Gamma \vdash c : Int} \text{ (T-Int)}$$
$$\frac{\Gamma \vdash t_0 : Bool \quad \Gamma \vdash t_1 : T \quad \Gamma \vdash t_2 : T}{\Gamma \vdash \text{if } t_0 \text{ then } t_1 \text{ else } t_2 : T} \text{ (T-If)}$$
$$\frac{\Sigma(op) = T_1 \rightarrow T_2 \quad \Gamma \vdash t : T_1}{\Gamma \vdash op(t) : T_2} \text{ (T-Op)}$$
$$\frac{\Gamma(x) = T}{\Gamma \vdash x : T} \text{ (T-Var)}$$
$$\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \lambda x : T_1. t : T_1 \rightarrow T_2} \text{ (T-Abs)}$$
$$\frac{\Gamma \vdash t_1 : T_1 \rightarrow T_2 \quad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1(t_2) : T_2} \text{ (T-App)}$$

Dynamic Semantics

$$\begin{array}{c}
 \frac{t_0 \rightarrow t'_0}{\text{if } t_0 \text{ then } t_1 \text{ else } t_2 \rightarrow \text{if } t'_0 \text{ then } t_1 \text{ else } t_2} \text{ (E-If)} \\
 \frac{}{\text{if true then } t_1 \text{ else } t_2 \rightarrow t_1} \text{ (E-IfTrue)} \\
 \frac{}{\text{if false then } t_1 \text{ else } t_2 \rightarrow t_2} \text{ (E-IfFalse)} \\
 \frac{t \rightarrow t'}{\text{op}(t) \rightarrow \text{op}(t')} \text{ (E-Op)} \\
 \frac{\text{op}(v) = v'}{\text{op}(v) \rightarrow v'} \text{ (E-OpVal)} \\
 \frac{t_1 \rightarrow t'_1}{t_1(t_2) \rightarrow t'_1(t_2)} \text{ (E-App1)} \\
 \frac{t_2 \rightarrow t'_2}{v_1(t_2) \rightarrow v_1(t'_2)} \text{ (E-App2)} \\
 \frac{}{(\lambda x.t_1)(v_2) \mapsto [x \mapsto v_2]t_1} \text{ (E-AppAbs)}
 \end{array}$$

Theorem (Progress) Suppose that t is a closed and well-typed term in STLC0, that is, there exists a derivation $\mathcal{D} :: \emptyset \vdash t : T$. Then either t is a value or $t \rightarrow t'$ holds for some term t' .

Exercise 1 (50 pts) Please present a complete proof of the progress theorem by structural induction on the derivation $\mathcal{D} :: \emptyset \vdash t : T$. Note that you need to handle all possible cases in the structural induction.

Theorem (Subject Reduction) Suppose that we have $\mathcal{D} :: \Gamma \vdash t : T$ and $t \rightarrow t'$ in STLC0. Then $\Gamma \vdash t' : T$ is derivable.

Exercise 2 (50 pts) Please present a complete proof of the subject reduction theorem by structural induction on the derivation $\mathcal{D} :: \Gamma \vdash t : T$. Note that you need to handle all possible cases in the structural induction.

Exercise 3 (100 pts) Please first implement a function `erase` that translates a term in STLC0 into the corresponding untyped term in deBruijn notation, and then implement a function `evaluation` that evaluates such an untyped term with the call-by-value semantics. You need to handle the following primitive operators:

```

+ : int * int → int
- : int * int → int
* : int * int → int
/ : int * int → int
> : int * int → bool
>= : int * int → bool
< : int * int → bool
<= : int * int → bool
= : int * int → bool
<> : int * int → bool
print : string → unit

```

```

signature STLC = sig

  datatype stp =
    TpBase of string (* base type *)
  | TpFun of stp * stp (* function type *)
  | TpTup of stp list (* tuple type *)

  datatype ttm =
    TtmBool of bool (* boolean constant *)
  | TtmInt of int (* integer constant *)
  | TtmStr of string (* string constant *)
  | TtmVar of string (* variable *)
  | TtmIf of ttm * ttm * ttm (* if-then-else term *)
  | TtmOp of string * ttm list (* built-in operator *)
  | TtmLam of string * stp * ttm (* lambda abstraction *)
  | TtmApp of ttm * ttm (* application *)
  | TtmLet of string * ttm * ttm (* let-binding *)
  | TtmLetrec of string * stp * ttm * ttm (* letrec-binding *)
  | TtmTup of ttm list (* tuple *)
  | TtmPro of ttm * int (* projection *)
  | TtmFix of ttm (* fixed point *)
  | TtmAsc of ttm * stp (* ascription *)

  val parseString: string -> ttm

  val typeCheck: ttm -> bool

  datatype utm = (* for untyped term via deBruijn indices *)
    UtmBool of bool | UtmInt of int | UtmStr of string
  | UtmVar of int | UtmIf of utm * utm * utm | UtmOp of string * utm list
  | UtmLam of utm | UtmApp of utm * utm
  | UtmLet of utm * utm | UtmLetrec of utm * utm
  | UtmTup of utm list | UtmPro of utm * int
  | UtmFix of utm

  val erase: ttm -> utm

  val evaluate: utm -> utm

end

structure Stlc :> STLC = struct
  ... ..
end

```

Figure 1: The signature for Exercise 3

```

type0 ::= base_type | LPAREN type RPAREN

type1 ::= type0 star_type0_seq

star_type0_seq ::= /* empty */
                | STAR type0 star_type0_seq

type ::= type1 | type1 -> type

vartype ::= LPAREN var COLON type RPAREN

vartypes ::= /* empty */
           | vartype vartypes

prefixop ::= print | ~
infixop  ::= + | - | * | / | > | >= | < | <= | = | <>

term0 ::= bool | integer | string
       | var
       | LPAREN term RPAREN
       | LPAREN terms RPAREN
       | LPAREN term COLON type RPAREN (* type ascription *)

term1 ::= term0
       | term1 DOT integer
       | term1 term0

term ::= term1
      | prefixop term (* resolve the ambiguity! *)
      | term infixop term (* resolve the ambiguity! *)
      | IF term THEN term ELSE term
      | LET var EQ term IN term
      | LETREC var COLON type EQ term IN term
      | LAM vartypes => term (* e.g., lam (x: int -> int) (y: int) => x y *)
      | FIX term0

terms ::= /* empty */
       | term comma_terms

comma_terms ::= /* empty */
            | COMMA term comma_terms

```

Figure 2: A Grammar for Exercise 3