CS 239, Jens Palsberg Mar 8, 2004

Midterm Exam

Each of the three questions is worth five points, for a total of 15 points. Write your name and id number at the top of the first page you submit. Write the solutions for different questions on different sheets of paper. Don't submit this handout.

1. [Type Inference] Consider the following λ-term:

$$E = \lambda f.\lambda x.((f\ x)\ (f\ x\ 5))$$

Write down a constraint system for E which captures typability with simple types. Decide if E has a simple type or not by doing unification of the constraints. In case E has a simple type, then show its principal type.

2. [CPS] Below is a definition of a Scheme function `f` which takes as argument two lists of integers `l` and `m`:

```
(define merge
  (lambda (l m)
    (if (null? l) m
        (if (null? m) l
            (if (<= (car l) (car m))
                (cons (car l) (merge (cdr l) m))
                (cons (car m) (merge l (cdr m))))))))
```

Transform `merge` to a program with these four properties:

- all calls to programmer-defined functions are in tail position,
- all occurrences of `lambda` are in definitions at the top level,
- all data are either integers or lists of integers, and
- all programmer-defined functions take no arguments.

Justify your answer by showing a step-by-step transformation process.

3. [Type Soundness] Below is the grammar for an expression language with integers, booleans and calls to a particular library function called f. The library function f takes two arguments; the first argument must be a boolean and the second argument must be an integer. A call to the library function f always returns an integer.

$$\begin{array}{rcl} e & \in & \mathit{Expression} \\ e & ::= & c \mid b \mid e \texttt{ + } 1 \mid f(e_1, e_2) \\ c & \in & \mathit{IntegerConstant} \\ b & \in & \mathit{BooleanConstant} \end{array}$$

A *value* is of the form c or of the form b. We use v to range over values. We use $\lceil c \rceil$ to denote the integer represented by an integer constant c. A small-step operational semantics for the language is given by the reflexive, transitive closure of the relation $\rightarrow_V$:

$$\rightarrow_V \subseteq \mathit{Expression} \times \mathit{Expression}$$

$$\frac{e \rightarrow_V e'}{e \texttt{ + } 1 \rightarrow_V e' \texttt{ + } 1} \tag{1}$$

$$(c \texttt{ + } 1) \rightarrow_V c' \quad (\lceil c \rceil + 1 = \lceil c' \rceil) \tag{2}$$

$$\frac{e_1 \rightarrow_V e_1'}{f(e_1, e_2) \rightarrow_V f(e_1', e_2)} \tag{3}$$

$$\frac{e_2 \rightarrow_V e_2'}{f(v, e_2) \rightarrow_V f(v, e_2')} \tag{4}$$

$$f(b, c) \rightarrow_V c' \qquad \lceil c' \rceil \text{ is the result of calling } f \text{ with arguments } b \text{ and } c \tag{5}$$

An expression e is *stuck* if it is not a value and there is no expression e' such that $e \rightarrow_V e'$. An expression e *goes wrong* if $\exists e' : e \rightarrow_V^* e'$ and e' is stuck.

A type is either int or $\mathsf{boolean}$. If e is an expression, and t is a type, then the judgment

$$\vdash e : t$$

means that e has the type t. Formally, this holds when the judgment is derivable by a finite derivation tree using the following four rules:

$$\vdash c : \mathsf{int} \tag{6}$$

$$\vdash b : \mathsf{boolean} \tag{7}$$

$$\frac{\vdash e : \mathsf{int}}{\vdash e \texttt{ + } 1 : \mathsf{int}} \tag{8}$$

$$\frac{\vdash e_1 : \mathsf{boolean} \quad \vdash e_2 : \mathsf{int}}{\vdash f(e_1, e_2) : \mathsf{int}} \tag{9}$$

An expression e is *well typed* if and only if there exists a type t such that $\vdash e : t$.

Prove in reasonable detail that a well-typed expression cannot go wrong.