CS 565, Jens Palsberg — Mar 30, 1999

Midterm Exam

Submit your solutions on separate paper (don't submit this handout). Write your name and id number on the top of the first page you submit. I will stable your pages together. This exam is intended to take you 2 hours. Budget your time wisely. There are 4 questions on the exam. Each question is worth 5 points, for a total of 20 points maximum. Write the solutions for different questions on different sheets of paper. Make clear what problem each solution is for. I have spare paper, just ask for it.

1. [**Beta-reduction**] Consider these Scheme definitions:

```
(define s
  (lambda (x) (lambda (y) (lambda (z) ((x z) (y z))))))

(define c
  (lambda (a) (lambda (b) (b a))))

(define i
  (((s c) s) c))
```

What is produced by following three expressions?

```
(i 2)
(i (lambda (x) x))
((i (lambda (x) x)) 2)
```

Justify your answer by showing a step-by-step computation.

2. [Type Soundness] Below is the grammar for an expression language with records of the form $\{l_1 : e_1, \ldots, l_n : e_n\}$ and integers.

$$
\begin{array}{rcl}
e & \in & \mathit{Expression} \\
e & ::= & \{l_1 : e_1, \ldots, l_n : e_n\} \mid e.l \mid e_1 \texttt{+} e_2 \mid c \\
l & \in & \mathit{Label} \\
c & \in & \mathit{IntegerConstant}
\end{array}
$$

A *value* is of the form:

$$
\begin{array}{rcl}
v & \in & \mathit{Value} \\
v & ::= & \{l_1 : v_1, \ldots, l_n : v_n\} \mid c
\end{array}
$$

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 $\rightarrow_V$:

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

$$(c_1 \texttt{+} c_2) \rightarrow_V c \quad (\mathit{where} \lceil c_1 \rceil + \lceil c_2 \rceil = \lceil c \rceil) \tag{1}$$

$$\frac{e_1 \rightarrow_V e_1'}{(e_1 \texttt{+} e_2) \rightarrow_V (e_1' \texttt{+} e_2)} \tag{2}$$

$$\frac{e_2 \rightarrow_V e_2'}{(v \texttt{+} e_2) \rightarrow_V (v \texttt{+} e_2')} \tag{3}$$

$$\frac{e \rightarrow_V e'}{e.l \rightarrow_V e'.l} \tag{4}$$

$$\{l_1 : v_1, \ldots, l_n : v_n\}.l_i \rightarrow_V v_i \;\; (1 \leq i \leq n) \tag{5}$$

$$\frac{e_i \rightarrow_V e_i'}{\{l_1 : v_1, \ldots, l_{i-1} : v_{i-1}, l_i : e_i, \ldots, l_n : e_n\} \rightarrow_V} \;\; (1 \leq i \leq n) \tag{6}$$
$$\{l_1 : v_1, \ldots, l_{i-1} : v_{i-1}, l_i : e_i', \ldots, l_n : e_n\}$$

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 a record type of the form $\{l_1 : t_1, \ldots, l_n : t_n\}$ or the base type int. Types are generated from the grammar:

$$\begin{array}{rcl} t & \in & \mathit{Type} \\ t & ::= & \{l_1 : t_1, \ldots, l_n : t_n\} \mid \mathsf{int} \end{array}$$

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:

$$\frac{\vdash e_1 : t_1 \quad \ldots \quad \vdash e_n : t_n}{\vdash \{l_1 : e_1, \ldots l_n : e_n\} : \{l_1 : t_1, \ldots l_n : t_n\}} \tag{7}$$

$$\frac{\vdash e : \{l_1 : t_1, \ldots, l_n : t_n\}}{e.l_i : t_i} \ (1 \leq i \leq n) \tag{8}$$

$$\frac{\vdash e_1 : \mathsf{int} \quad \vdash e_2 : \mathsf{int}}{\vdash e_1 \mathbin{\mathsf{+}} e_2 : \mathsf{int}} \tag{9}$$

$$\vdash c : \mathsf{int} \tag{10}$$

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.

3. [Structural Subtyping] Here are two types s, t:

$$\begin{array}{rcl} s & = & \mu\alpha.(\top \to (\alpha \to (\top \to (\mu\beta.(\alpha \to (\top \to \beta)))))) \\ t & = & \mu\gamma.(\gamma \to (\mu\delta.((\top \to \bot) \to (\gamma \to \delta)))) \end{array}$$

Is $s \leq t$ in the Amadio/Cardelli type ordering? Justify your answer.

4. [CPS] Below is a definition of a Scheme function `streak` which takes as arguments a flat list of integers `l` and an integer `n`. An example of a flat list of integers is `(1 2 3 4)` but not `(1 (2 3) 4)`. Here is an example of the behavior of `streak`:

```
> (streak '(1 2 3 4 7 8 9 0) 2)
(2 3 4)
```

Here is the definition of `streak`:

```
(define streak
  (lambda (l n)
    (if (null? l)
        l
        (if (eq? (car l) n)
            (cons n (streak (cdr l) (+ n 1)))
            (streak (cdr l) n)))))
```

Transform `streak` 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 possibly nested lists of integers, and
- all programmer-defined functions take no arguments.

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