CS 565, Jens Palsberg — Nov 11, 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. [**Operational Semantics**] Below is a grammar for a subset of Java called Flyweight Java. Method overloading is not allowed in Flyweight Java.

$$
\begin{array}{rcl}
P & \in & \mathit{Program} \\
P & ::= & \texttt{class}\ \mathit{Main}\ \{ \\
 & & \quad \texttt{public static void main(String [] } x\texttt{)\{} \\
 & & \quad\quad \texttt{new}\ C().m(e) \\
 & & \quad \}\} \\
 & & D^* \\
D & \in & \mathit{ClassDeclaration} \\
D & ::= & \texttt{class}\ C\ \texttt{extends}\ C\{M^*\} \\
M & \in & \mathit{Method} \\
M & ::= & C\ m(C\ x)\{\texttt{return}\ e;\} \\
e & \in & \mathit{Expression} \\
e & ::= & x \mid \mathit{this} \mid e.m(e) \mid \texttt{new}\ C() \mid (C)e \\
C, m, x & \in & \mathit{Identifier}
\end{array}
$$

Give a small-step operational semantics for Flyweight Java.

2. [Type Soundness] Below is the grammar for an expression language with integers and random-access arrays of the form $\mathsf{array}(e_1, \ldots, e_n)$.

$$
\begin{array}{rcl}
e & \in & Expression \\
e & ::= & e \texttt{ + } 1 \mid c \mid \mathsf{array}(e_1, \ldots, e_n) \mid e_1[e_2] \mid \mathsf{error} \\
c & \in & IntegerConstant
\end{array}
$$

A *value* is of the form:

$$
\begin{array}{rcl}
v & \in & Value \\
v & ::= & c \mid \mathsf{array}(e_1, \ldots, e_n) \mid \mathsf{error}.
\end{array}
$$

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 $\rightarrow_V$:

$$\rightarrow_V \subseteq Expression \times Expression$$

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

$$(\mathsf{error} \texttt{ + } 1) \rightarrow_V \mathsf{error} \tag{2}$$

$$\frac{e \rightarrow_V e'}{(e \texttt{ + } 1) \rightarrow_V (e' \texttt{ + } 1)} \tag{3}$$

$$\mathsf{array}(e_1, \ldots, e_n)[c] \rightarrow_V e_i \quad (\text{where } i = \lceil c \rceil \text{ and } 1 \leq i \leq n) \tag{4}$$

$$\mathsf{array}(e_1, \ldots, e_n)[c] \rightarrow_V \mathsf{error} \quad (\text{where } i = \lceil c \rceil \text{ and } (i \leq 0 \text{ or } n < i)) \tag{5}$$

$$\mathsf{error}[e] \rightarrow_V \mathsf{error} \tag{6}$$

$$v[\mathsf{error}] \rightarrow_V \mathsf{error} \quad (v \neq \mathsf{error}) \tag{7}$$

$$\frac{e_1 \rightarrow_V e_1'}{e_1[e_2] \rightarrow_V e_1'[e_2]} \tag{8}$$

$$\frac{e_2 \rightarrow_V e_2'}{v[e_2] \rightarrow_V v[e_2']} \quad (v \neq \mathsf{error}) \tag{9}$$

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 an array type of the form $t[\ ]$ or the base type int. Types are generated from the grammar:

$$\begin{array}{rcl} t & \in & \mathit{Type} \\ t & ::= & t\,[\ ] \mid \mathsf{int} \end{array}$$

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 five rules:

$$\frac{\vdash e : \mathsf{int}}{\vdash e \mathbin{\mathsf{+}} 1 : \mathsf{int}} \tag{10}$$

$$\vdash c : \mathsf{int} \tag{11}$$

$$\frac{\vdash e_1 : t \quad \dots \quad \vdash e_n : t}{\vdash \mathsf{array}(e_1, \dots, e_n) : t\,[\ ]} \tag{12}$$

$$\frac{\vdash e_1 : t\,[\ ] \qquad \vdash e_2 : \mathsf{int}}{\vdash e_1[e_2] : t} \tag{13}$$

$$\vdash \mathsf{error} : t \tag{14}$$

Notice that error has all types. 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:

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

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

4. [CPS] Below is a definition of a Scheme function `f` which takes as argument an integer `n`:

```
(define f
  (lambda (n)
    (if (> n 100)
        n
        (+ (f (+ n 1)) (f (+ n 2))))))
```

Transform `f` 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.