CS 565, Jens Palsberg   Mar 25, 2002

Midterm Exam

Each of the four questions is worth five points, for a total of 20 points. Write your name and id number at the top of the first page you submit. Write solutions for different questions on different sheets of paper. Don't submit this handout.

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

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

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

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

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

3. [**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
        (+ 2 (f (f (- n 8)))))))
```

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.

4. [Type Soundness] Below is the grammar for an expression language with integers, strings, and an overloaded + operation.

$$
\begin{array}{rcl}
e & \in & \mathit{Expression} \\
e & ::= & c \mid s \mid e_1 \texttt{ + } e_2 \\
c & \in & \mathit{IntegerConstant} \\
s & \in & \mathit{StringConstant}
\end{array}
$$

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

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

$$\frac{e_1 \rightarrow_V e_1'}{e_1 \texttt{ + } e_2 \rightarrow_V e_1' \texttt{ + } e_2} \tag{1}$$

$$\frac{e_2 \rightarrow_V e_2'}{v \texttt{ + } e_2 \rightarrow_V v \texttt{ + } e_2'} \tag{2}$$

$$(c_1 \texttt{ + } c_2) \rightarrow_V c \qquad (\lceil c_1 \rceil + \lceil c_2 \rceil = \lceil c \rceil) \tag{3}$$

$$(s_1 \texttt{ + } s_2) \rightarrow_V s \qquad (\lceil s \rceil \text{ is the concatenation of } \lceil s_1 \rceil \text{ and } \lceil s_2 \rceil) \tag{4}$$

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 String. 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:

$$\vdash c : \mathsf{int} \tag{5}$$

$$\vdash s : \mathsf{String} \tag{6}$$

$$\frac{\vdash e_1 : \mathsf{int} \qquad \vdash e_2 : \mathsf{int}}{\vdash e_1 \texttt{ + } e_2 : \mathsf{int}} \tag{7}$$

$$\frac{\vdash e_1 : \mathsf{String} \qquad \vdash e_2 : \mathsf{String}}{\vdash e_1 \texttt{ + } e_2 : \mathsf{String}} \tag{8}$$

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.