CS 565, Jens Palsberg Apr 4, 2000

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. [**Type Inference**] Here is a grammar for a λ-calculus. We use c to range over integer constants, and we use d to range over double constants.

$$\begin{array}{rcl} e & ::= & x \mid \lambda x.e \mid e_1 e_2 \mid c \mid d \\ t & ::= & t \to t \mid \mathsf{int} \mid \mathsf{double} \mid \alpha. \end{array}$$

We use α to range over type variables. Consider the following type rules:

$$A[x : t] \vdash x : t \tag{1}$$

$$\frac{A[x : s] \vdash e : t}{A \vdash \lambda x.e : s \to t} \tag{2}$$

$$\frac{A \vdash e_1 : s \to t \quad A \vdash e_2 : s}{A \vdash e_1 e_2 : t} \tag{3}$$

$$A \vdash c : \mathsf{int} \tag{4}$$

$$A \vdash d : \mathsf{double} \tag{5}$$

$$\frac{A \vdash e : s \quad s \leq t}{A \vdash e : t} \tag{6}$$

An partial ordering $\leq$ of the types is defined by the four rules:

$$\mathsf{int} \leq \mathsf{int} \qquad \mathsf{int} \leq \mathsf{double} \qquad \mathsf{double} \leq \mathsf{double} \qquad \frac{s' \leq s \quad t \leq t'}{s \to t \leq s' \to t'}$$

A type constraint is of the form $s \leq t$, where s, t are types. Present an algorithm which given a closed expression e produces a set of type constraints C such that e is typable if and only if C is satisfiable. Argue informally for the correctness of the algorithm.

2. [Type Soundness] Here is the grammar for a typed assembly language. We use c to range over integer constants, and we use l to range over labels.

$$
\begin{array}{rcll}
v & ::= & c \mid l & \text{(values)} \\
t & ::= & \mathsf{int} \mid \Gamma & \text{(types)} \\
e & ::= & \text{mov } r_d, r_s; e \mid \text{set } r_d, v; e \mid \text{inc } r; e \mid \text{jmp } r & \text{(code)} \\
R & ::= & [r \mapsto v, \ldots] & \text{(register file)} \\
\Gamma & ::= & [r : t, \ldots] & \text{(register file type)} \\
H & ::= & [l \mapsto (\Gamma, e), \ldots] & \text{(code heap)} \\
\Psi & ::= & [l : \Gamma, \ldots] & \text{(heap type)} \\
\text{s} & ::= & (H, R, e) & \text{(program state)}
\end{array}
$$

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 $\rightarrow$:

$$\rightarrow \subseteq \text{ program state} \times \text{program state}$$

$$(H, R, \text{mov } r_d, r_s; e) \rightarrow (H, R[r_d \mapsto R(r_s)], e) \tag{7}$$

$$(H, R, \text{set } r_d, v; e) \rightarrow (H, R[r_d \mapsto v)], e) \tag{8}$$

$$(H, R, \text{inc } r; e) \rightarrow (H, R[r \mapsto c')], e) \tag{9}$$

where $\lceil c' \rceil = \lceil c \rceil + 1$, provided $R(r) = c$.

$$(H, R, \text{jmp } r) \rightarrow (H, R, e) \tag{10}$$

provided $R(r) = l$ and $H(l) = (\Gamma, e)$.

A program state s is *stuck* if there is no program state s' such that $s \rightarrow s'$. A program state s *goes wrong* if $\exists s' : s \rightarrow^* s'$ and s' is stuck.

We will use five forms of judgments and an ordering of register file types to describe what it means for a program state to be well typed.

Values:

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

$$\Psi \vdash l : \Gamma \quad (\Psi(l) = \Gamma) \tag{12}$$

Code:

$$\frac{\Psi, \Gamma[r_d : \Gamma(r_s)] \vdash e}{\Psi, \Gamma \vdash \text{mov } r_d, r_s; e} \tag{13}$$

$$\frac{\Psi \vdash v : t \quad \Psi, \Gamma[r_d : t] \vdash e}{\Psi, \Gamma \vdash \text{set } r_d, v; e} \tag{14}$$

$$\frac{\Gamma(r) = \mathsf{int} \quad \Psi, \Gamma \vdash e}{\Psi, \Gamma \vdash \text{inc } r; e} \tag{15}$$

$$\frac{\Gamma(r) = \Gamma' \quad \Gamma \leq \Gamma'}{\Psi, \Gamma \vdash \text{jmp } r} \tag{16}$$

Register files:

$$\frac{\Psi \vdash v : t \quad \ldots}{\Psi, \Gamma \vdash [r \mapsto v, \ldots]} \quad (\Gamma(r) = t, \ldots) \tag{17}$$

Ordering of register file types:

$$[r_1 : t_1, \ldots, r_n : t_n, \ldots, r_{n+m} : t_{n+m}] \leq [r_1 : t_1, \ldots, r_n : t_n] \tag{18}$$

where $m \geq 0$.

Code heaps:

$$\frac{\Psi, \Gamma \vdash e \quad \ldots}{\Psi \vdash [l \mapsto (\Gamma, e), \ldots]} \quad (\Psi(l) = \Gamma, \ldots) \tag{19}$$

Program states:

$$\frac{\Psi \vdash H \quad \Psi, \Gamma \vdash R \quad \Psi, \Gamma \vdash e}{\vdash (H, R, e)} \tag{20}$$

A program state s is *well typed* if and only if $\vdash s$.

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

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

$$\begin{array}{rcl} s & = & \mu\alpha.([\mu\beta.((\beta \to \alpha) \to (\bot \to \beta))] \to [\top \to (\mu\gamma.(\langle\mu\delta.((\gamma \to \delta) \to \top)\rangle \to \bot))]) \\ t & = & \mu u.([(\mu v.(\langle\top \to (\bot \to (\bot \to v))\rangle \to v)) \to u] \to [\mu w.(\bot \to (\bot \to w))]) \end{array}$$

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

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

```
(define f
  (lambda (n)
    (if (< n 1)
        2
        (if (< (f (- n 2)) 3)
            (- n 2)
            (- (f (- n 1)) (* 2 n))))))
```

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.