CS 239, Jens Palsberg Mar 22, 2004

# Final 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 the solutions for different questions on different sheets of paper. Don't submit this handout.

**1. [Fast Control-Flow Analysis]** Consider the following lambda-term:

$$G = (\lambda^1 f.(f(f(\lambda^2 x.x))))(\lambda^3 y.y)$$

Show the graph for $G$ that is used by the Heintze/McAllester $O(n^2)$ time flow analysis algorithm. What is the label set for $G$ produced by the algorithm?

**2. [Flow Analysis]** Consider a subset of Java in which classes are defined by the following grammar:

| | | |
|---|---|---|
| *InterfaceDecl* | ::= | **interface** *id* { *Header** } |
| *Header* | ::= | **void** *id* ( *Type id* ); |
| *ClassDecl* | ::= | **class** *id* **implements** *id* { *VarDecl** *MethodDecl** } |
| *VarDecl* | ::= | *Type id* ; |
| *Type* | ::= | *id* \| **boolean** |
| *MethodDecl* | ::= | **public void** *id* ( *Type id* ) { *VarDecl** *Statement** } |
| *Statement* | ::= | **if** ( *Exp* ) { *Statement** } **else** { *Statement** } |
| | ::= | *id* = *Exp* ; |
| | ::= | *Exp* . *id* ( *Exp* ) |
| *Exp* | ::= | **true** \| **false** \| *id* \| **this** \| **new** *id* () |

We use the term *flow analysis* to denote any program analysis which for every expression $e$ in a program produces a set $S$ of class names such that if $e$ evaluates to a $C$-object, then $C \in S$. Define a flow analysis which is *flow sensitive*, that is, for a local variable `x` (local to some method), after the statements,

```
x = new A();
x = new B();
```

the flow set for `x` is preferably { `B` }, rather than { `A`, `B` }.

**3. [Recursive Types]** Here is a grammar for types:

$$t \;::=\; \alpha \;\mid\; t \to t \;\mid\; \mu\alpha.t$$

where $\alpha$ ranges over an infinite set of type variables, and the type $\mu\alpha.t$ is a recursive type. Consider the following type rules for $\lambda$-calculus expressions:

$$A \vdash x : t \quad (A(x) = t) \tag{1}$$

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

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

$$\frac{A \vdash e : \mu\alpha.t}{A \vdash e : t[\alpha := (\mu\alpha.t)]} \tag{4}$$

$$\frac{A \vdash e : t[\alpha := (\mu\alpha.t)]}{A \vdash e : \mu\alpha.t} \tag{5}$$

Prove in reasonable detail that for all closed $e$, we can derive $\emptyset \vdash e : \mu\alpha.(\alpha \to \alpha)$.

**4. [Stack Machine]** Here is a grammar for the action sequence of a stack machine:

$$\begin{array}{lrcl}
\text{(actions)} & a & ::= & \texttt{halt} \\
& & \mid & \texttt{incr; } a \\
& & \mid & \texttt{add; } a \\
& & \mid & \texttt{push } v\texttt{; } a \\
& & \mid & \texttt{pop;}a \\
& & \mid & \texttt{iftrue } v_1 \texttt{ else } v_2\texttt{; } a \\
\text{(values)} & v & ::= & c \mid \texttt{true} \mid \texttt{false}
\end{array}$$

where $c$ ranges over integer constants. A stack is a finite list of values. We use $s$ to range over stacks, and we use the notation $(v\ s)$ to denote a stack with top element $v$ and the rest of the stack being $s$. A state in the stack machine is either a pair $\langle s, a\rangle$ or a final state $v$. We use $\sigma$ to range over states. A small-step operational semantics for the language is given by the reflexive, transitive closure of the relation $\to$:

$$\langle (v\ s), \texttt{halt}\rangle \;\to\; v \tag{6}$$

$$\langle (c\ s), \texttt{incr; } a\rangle \;\to\; \langle (c'\ s), a\rangle \quad \text{where } c' = c + 1 \tag{7}$$

$$\langle (c_1\ (c_2\ s)), \texttt{add; } a\rangle \;\to\; \langle (c'\ s), a\rangle \quad \text{where } c' = c_1 + c_2 \tag{8}$$

$$\langle s, \texttt{push } v\ \texttt{; } a\rangle \;\to\; \langle (v\ s), a\rangle \tag{9}$$

$$\langle (v\ s), \texttt{pop; } a\rangle \;\to\; \langle s, a\rangle \tag{10}$$

$$\langle (\texttt{true}\ s), \texttt{iftrue } v_1 \texttt{ else } v_2\ \texttt{; } a\rangle \;\to\; \langle (v_1\ s), a\rangle \tag{11}$$

$$\langle (\texttt{false}\ s), \texttt{iftrue } v_1 \texttt{ else } v_2\ \texttt{; } a\rangle \;\to\; \langle (v_2\ s), a\rangle \tag{12}$$

A state $\sigma$ is *stuck* if it is not a final state and there is no state $\sigma'$ such that $\sigma \to \sigma'$. A state $\sigma$ *goes wrong* if $\exists \sigma' : \sigma \to^* \sigma'$ and $\sigma'$ is stuck.

Define a type system with the property that a well-typed state cannot go wrong. Argue informally why the property holds.