CS 565 May 9, 1997
Jens Palsberg

Final Exam + Qualifying Exam

Submit your solutions on separate paper (don't submit this handout). Write your name, id number, and department 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.

To those who are taking the qualifying exam: to pass the exam you must get an excellent score.

The Core of Java: cjava

Three of the questions on this final exam concerns a core subset of Java called cjava. You will notice that cjava is a subset of mjava.

$$
\begin{array}{lcl}
\mathit{Prog} & ::= & \mathit{Decl}^* \\
\mathit{Decl} & ::= & \texttt{class}\ \mathit{Id}\ \{\ \} \\
 & | & \texttt{class}\ \mathit{Id}\ \texttt{extends}\ \mathit{Id}\ \{\ \mathit{Meth}^*\} \\
\mathit{Meth} & ::= & \texttt{public}\ \mathit{Id}\ \texttt{()}\ \{\ \texttt{return}\ \mathit{Expr}\ \texttt{;}\} \\
\mathit{Expr} & ::= & \texttt{this} \\
 & | & \texttt{new}\ \mathit{Id}\ \texttt{()} \\
 & | & \mathit{Expr}\ \texttt{.}\ \mathit{Id}\ \texttt{()}
\end{array}
$$

Notice that methods have no arguments. There is no specification of return types, as in mjava. There is no overloading in cjava: in a class, all methods have different names.

1. [Interpreter] Assume that the above grammar is acceptable to `sllgen`, and that the lexical specification and the grammar have already been typed in. **Write in Scheme** the main parts of an interpreter for cjava. It is not important whether your Scheme code is completely syntactically correct.

2. [Subtyping] Consider the following language of nameless Java interfaces.

$$\begin{array}{lcl} \mathit{Type} & ::= & \texttt{interface}\ \{\ \mathit{Head}^*\} \\ \mathit{Head} & ::= & \mathit{Type}\ \mathit{Id}\ \texttt{();} \end{array}$$

The method names in an interface are different. Notice that all interfaces are finite. The smallest possible interface is `interface` $\{\}$.

Here is a structural subtype ordering $\leq$ on the above types:

$$\begin{array}{l} \texttt{interface}\ \{\ t_1\ \texttt{m}_1\ \texttt{();}\ \ldots t_n\ \texttt{m}_n\ \texttt{();}\ \ldots t_{n+k}\ \texttt{m}_{n+k}\ \texttt{();}\ \} \leq \\ \texttt{interface}\ \{\ t_1\ \texttt{m}_1\ \texttt{();}\ \ldots t_n\ \texttt{m}_n\ \texttt{();}\ \} \end{array}$$

The idea is that one interface J is a subtype of another interface I, written $J \leq I$, if J has at least the same methods as I, and for every common method, the return types are the same.

Define a way of representing such types in Java. **Write in Java** an algorithm which given representations of finite interfaces I, J decides if $I \leq J$. It is not important whether your Java code is completely syntactically correct.

Suppose we allow recursive types. The subtype ordering is the same as for the finite types. **Explain in detail** how you would modify your Java code from the first part of the question to handle recursive types.

3. [Type Rules] Consider the *finite* interfaces from Problem 2 together with the subtype ordering. Using that type language we can imagine assigning a type to some cjava expressions. We will also have to assign types to classes, with the idea that the type of a class is the same as the type of all its objects.

Give type rules for cjava such that if a program has a type derivation, then the error "message not understood" is not possible.

4. [Simply-Typed λ-calculus] Consider the following λ-term F.

$$F = (\lambda v.(((\lambda x.\lambda k.kv)v)(\lambda a.a)))(\lambda y.\lambda m.my)$$

I have used the standard convention that the body of a λ-abstraction extends "as far as possible." **Write down the constraint system** for F in the sense of Lecture note 10 (and homework 9). **Decide** if F has a simple type or not by doing unification of the constraints. If F has a simple type, then **give a type derivation.**