**Indian Institute of Technology Delhi**
**Department of Computer Science and Engineering**

CSL302 Programming Languages Minor 1

February 14, 2009 16:00–17:00 Maximum Marks: 60

Open notes. Write your name, entry number and group at the top of each sheet in the blanks provided. Answer all questions in the space provided, in blue or black ink (no pencils, no red pens). Budget your time according to the marks. Do rough work on separate sheets.

Q1. (1+1 marks) **Lexical and syntactic analysis.** Hopefully, you have been programming in ML. Write an example each of incorrect ML code that will produce

1. a lexical error detected by the scanner:

2. a syntax error, detected by the parser:

Q2. (8 marks) **Compilers and Interpreters**. Recall that *byte code* is the code that is interpreted on a virtual machine, whereas *native code* is machine code for a physical machine, and that source code is code written in a high-level programming language. Suppose I have a *macos* machine and a *Wintel* machine, as well as the following software objects:

1. The source code, written in language *j*, of a compiler *jc*; *jc*, when executed, takes programs in language *j*, and produces native code for a *macos*-machine;
2. A compiler *jbc* already compiled into *byte code* language $B$; *jbc*, when executed, takes programs in language *j* and produces code in byte code $B$;
3. A byte-code $B$-interpreter $BI_{Wintel}$ (which is in native code), which runs on a *Wintel*-machine.

Using the notation $X(Y)$ to mean running the software entity $X$ on input $Y$, and in particular $BI_{Wintel}(X)$ to denote running the byte-code $B$ program called $X$ on the byte-code $B$-interpreter $BI_{Wintel}$ (or by drawing a small diagram) indicate the steps by which, following a small variant on boot-strapping, I can produce a native code compiler for language *j* programs, which runs on *macos*-machines (and produces native *macos*-machine code).

Q3. (16 marks) **Grammar for a simple language**. Let's design the syntax for a simple language Plog.

1. A Plog *program* consists of a possibly empty *sequence* of *clauses*, with each clause terminated by a full-stop (".").
2. A *clause* can be of two forms, either (i) just a single *atom*, or (ii) an *atom* followed by the keyword "**if**" and then a non-empty *sequence of atoms*, separated by semicolons (";").
3. Each *atom* consists of either (i) a single *identifier* (called the predicate symbol), or (ii) the identifier (predicate symbol) followed by a *tuple* of *terms*.
4. A *tuple of terms* begins with a left parenthesis "(", then a non-empty sequence of *terms* separated by commas (",") and finally a right parenthesis ")".
5. A *term* consists of either (i) a single *identifier* (a constructor) or (ii) an identifier (the constructor) followed by a *tuple of terms*.
6. You may assume that *identifiers* have been already specified.

Write the rules of an unambiguous context-free concrete grammar $\mathcal{G}$ for the language Plog (Don't write Yacc code):

Q4. (8 marks) **Representation of abstract syntax.** Write an ML data type definition for abstractly representing Plog programs, assuming that identifiers can be represented by the predefined ML type `string`.

**datatype** *program* =

Q5. (3+3 marks) **Auxiliary functions.** With respect to your representational data types in Q4, specify recursive functions *preds* and *consts*, which return the set of predicate symbols and constructors respectively of a given Plog program (use mathematical notation; you don't need to write programs).

Q5. (4x2 marks) **Unification**. For each of the following pairs of terms, if the *most general unifier* exists, present it as a *simultaneous substitution*, or else state why it does not exist:

1. $f(h(a,b),X)$ and $f(Y,h(X,b))$.

2. $f(h(a,Y),X)$ and $f(X,h(b,Y))$.

3. $f(h(X,a),Y)$ and $f(Z,h(b,X))$.

4. $f(h(a,X),h(X,b))$ and $f(h(a,b),h(Z,X))$.

Q6. (4+4+4 marks) **Prolog Programming**. Suppose we represent sets as Prolog lists having no duplicates. Write Prolog programs to express the following operations:

1. `member`$(X,A)$ – succeeds if element $X$ is in set $A$, fails otherwise.

2. `union`$(A,B,C)$ – union of $A$ and $B$ yields $C$.

3. `intersection`$(A,B,C)$ – intersection of $A$ and $B$ yields $C$.