CPSC 510: Fall Final Midway Exam

Robin Cockett

January 2, 2001

This exam is worth 40% of the examination credit of the course. This is a take home exam which is due on the first class in 2001. You must complete all except for the last question of the exam BY HAND (that is in your own hand writing).

1. (15 points) For each of these grammars you are expected to calculate the vital statistics (that is nullable, endable, first, and follow properties) and to give the item automata (Note: the grammars are written in CFG format with nonterminals in lower case, the productions for a nonterminal terminated by a ".", and the first nonterminal being the start symbol).

   (a) (5 points) Show that the following grammar is neither LL(1) nor LR(1):

   ```
   a -> b A
   b -> b A
      |.
   ```

   What language does this grammar recognize? Is this grammar ambiguous?

   (b) (5 points) Show that the following grammar is LR(1) but neither LL(1) nor LALR(1):

   ```
   s -> A a D | B b D | c.
   c -> A b E | B a E.
   a -> C.
   b -> C.
   ```

Is this grammar ambiguous?

(c) (5 points) Show that the following grammar is LL(1) but not SLR(1):

```
s -> a A a B | b B b A.
a -> .
b -> .
```

Is this grammar LR(1)? Is this grammar LALR(1)?

2. (15 points) Consider the attribute grammar below and overleaf:

   (a) Draw the plumbing diagrams for this attribute system,

   (b) Show that this system is an attribute system. In particular, identify the inherited and synthesized attributes, identify the dependencies at the interfaces, and show that the system is a strongly non-circular system.

   (c) Generate mutually recursive pseudo-code to calculate the value of `exp.ir` and `exp.type`.

```
exp -> OP LPAR exp COMMA exp RPAR
     { exp_1.type = type_match(exp_2.type,exp_3.type,OP.name);
       exp_1.ir = if exp_2.ir = ERROR or exp_3.ir = ERROR
                     or exp_1.type = ERROR then ERROR
                  else i_APP(OP.name,exp_2.ir,exp_3.ir);
       exp_2.symtable = exp_1.symtable;
       exp_3.symtable = exp_1.symtable; }
     | LET decls stmts IN exp END
     { decls.in_symtable = exp_1.symtable;
       exp_2.symtable = decls.out_symtable;
       stmts.symtable = decls.out_symtable;
       exp_1.ir = if stmts.ir = ERROR or exp_2.ir = ERROR
                  then ERROR else i_ESEQ(stmts.ir,exp_2.ir);
       exp_1.type = exp_2.type; }
     | VAR
     { exp.type = findtype(VAR.name,exp.symtable);
       exp.ir = if exp.type = ERROR then ERROR
                else i_VAR(offsets(VAR.name,exp.symtable)); }.
```

```
stmt -> VAR ASSIGN exp
      { exp.symtable = stmt.symtable;
        var_type = findtype(VAR.name,stmt.symtable);
        var_ir = if var_type = ERROR then ERROR
                 else i_VAR(offsets(VAR.name,stmt.symtable));
        stmt.ir = if var_type = ERROR
                     or not(type_equal(var_type, exp.type))
                     or exp.ir = ERROR then ERROR
                  else i_ASS(var_ir,exp.ir); }.

stmts -> stmt SEMICOLON stmts
       { stmt.symtable = stmts_1.symtable;
         stmts_2.symtable = stmts_1.symtable;
         stmts_1.ir = if stmt.ir = ERROR
                         or stmts_2.ir = ERROR then ERROR
                      else i_SEQ(stmt.ir,stmts_2.ir); }
       |
       { stmts.ir = i_NULL; }.

decl -> VAR COLON TYPE
       { decl.out_symtable = addsym(decl.in_symtable
                                   , VAR.name, TYPE.name); }.

decls -> decl SEMICOLON decls
       { decl.in_symtable = decls_1.in_symtable;
         decls_2.in_symtable = decl.out_symtable;
         decls_1.out_symtable = decls_2.out_symtable; }
       |
       { decls.out_symtable = decls.in_symtable }.
```

3. (10 points) This question need not be answered by hand (i.e. you permitted to answer this question with the aid of a computer):

   Propose an extension to the language `m+` which includes a new basic type `char` of characters and arrays of the three basic types (`bool`, `int`, and `char`). Explain any additional instructions you would expect to have to add to the existing `am+` stack machine in order to implement this language.

   You are expected to propose a grammar for the extended language (the proposed language must be a superset of the existing language) and to discuss its semantics. Pay careful attention to the following issues:

   (a) The grammar should be implementable in YACC!

   (b) Can the size of a local array in a declaration be determined by a variable/argument? If so what are the implementation issues?

   (c) Should arrays be passed by value or by reference. Discuss the implementation issues.

   (d) You should assume that array bound checking is a must. How would this be implemented?

   (e) What basic character handling primitives should the language have?

   Please note you are not expected to solve all these implementation issues in detail but you are expected to understand the issues (hint: many of these issues are discussed in the dragon book!)