Mid-term examination — due Monday 15 October

This is a take-home examination. The goal is to learn the material on practical parsing, specifically LR(1) parsing, through the implementation of an LR(1) parser generator.

Preparation

Read all the material in Chapter 2 of Scott, and in Chapter 3 of Cooper and Torczon (handout). You may also consult Aho, Sethi, and Ullman or other sources listed in the course syllabus.

What should be built

Write a parser generator. The parser generator takes as input a formal description of a context-free grammar and produces as output a parser for the language defined by the grammar.

The formal description of a context-free grammar is written in extended BNF notation defined below.

The generated parser is an LR(1) parser. Its *action* and *goto* tables should be constructed according to the procedure outlined in Chapter 3 of Cooper and Torczon. The generated parser reads in a sequence of tokens. The generated parser produces the output *Yes* or *No*. It does not need to build an explicit parse tree.

The generated parser may be in any programming language, but you must be able to compile, run, and test it. The generated parser may be data-driven (a fixed skeleton parser consults explicit tables), or it may be code-driven (the tables are folded into the control flow of the parser).

In case it is possible to produce an LR(1) parser for the input grammar, the states (item sets) and the final *action* and *goto* tables should be printed in human-readable (symbolic) form.

In case it is not possible to produce an LR(1) parser for the input grammar, detailed error messages must also be printed to describe the reasons of failure.

The parser generator should be tested on at least the grammars given below (which will also be provided on-line).

The parser generator may be in any programming language, but you must be able to compile, run, and test it.

Scanning

You may assume that scanning is done in some standard way, and the parser sees tokens returned by the scanner. You can glue ad hoc scanners to the generated parsers, or you can even do the scanning by hand for your test inputs. If you wish, on the other hand, you can integrate the parser generator with a scanner generator, and have a common syntax description file that specifies both the context-free grammar and the several lexical classes of a language. Note that this may involve substantially more work.

Grammar specification language

The grammar of the language used to specify grammars is specified here in terms of itself.

```
Grammar -->  Symbols_List

Symbols_List --> Symbol  Symbols_List
               | Symbol

Symbol --> Identifier  Right_Arrow  Productions_List

Productions_List --> Production  Productions_Separator Productions_List
                   | Production

Production --> Identifiers_List

Identifiers_List --> Identifier  Identifiers_List |
```

The terminal symbols in the grammar specification language are identifiers, the right arrow, and the production separator. Identifiers are as in ModulaZ, or they can be an arbitrary string enclosed in a `<...>` pair. The right arrow is written `-->` or `::=`, and the production separator is written `|` or `/`.

Identifiers that appear on the left-hand side of right arrows are non-terminals. Any other identifiers are assumed to be terminals.

Analysis

Once you have built the parser generator, experiment with various grammars, including all that are given below. Report your experiences. If a grammar does not permit the construction of an LR(1) parser, show how to modify the grammar (preserving the same language!).

Example grammars

Expressions

```
Expr --> Expr plus Term
       | Expr minus Term
       | Term
Term --> Term times Factor
       | Term divide Factor
       | Factor
Factor --> lpar Expr rpar
         | num
         | id
```

JW_Pascal

```
      program --> program_heading block full_stop

      program_heading --> program_keyword identifier
                          left_parenthesis file_identifier_list
                          right_parenthesis semicolon

      file_identifier_list --> file_identifier comma
                                  file_identifier_list |
                               file_identifier

      file_identifier --> identifier

      block --> label_declaration_part   constant_definition_part
                type_definition_part    variable_declaration_part
                procedure_and_function_declaration_part
                statement_part

      label_declaration_part --> label_keyword label_list semicolon |


      label_list --> label comma label_list |
                     label

      label --> unsigned_integer

      constant_definition_part --> const_keyword
                                   constant_definition_list
                                   semicolon |

      constant_definition_list --> constant_definition semicolon
                                      constant_definition_list |
                                   constant_definition
```

```
constant_definition --> identifier equal_sign constant

constant --> unsigned_number |
             sign unsigned_number |
             constant_identifier |
             sign constant_identifier |
             string

unsigned_number --> unsigned_integer |
                    unsigned_real

constant_identifier --> identifier

type_definition_part --> type_keyword type_definition_list
                         semicolon |

type_definition_list --> type_definition semicolon
                           type_definition_list |
                         type_definition

type_definition --> identifier equal_sign type

type --> simple_type |
         structured_type |
         pointer_type

simple_type --> scalar_type |
                subrange_type |
                type_identifier

scalar_type --> left_parenthesis identifier_list
                right_parenthesis

identifier_list --> identifier comma identifier_list |
                    identifier

subrange_type --> constant two_dots constant

type_identifier --> identifier

structured_type --> packed_keyword unpacked_structured_type |
                    unpacked_structured_type

unpacked_structured_type --> array_type |
                             record_type |
                             set_type |
                             file_type

array_type --> array_keyword left_bracket index_type_list
               right_bracket of_keyword component_type

index_type_list --> index_type comma index_type_list |
                    index_type

index_type --> simple_type

component_type --> type

record_type --> record_keyword field_list end_keyword

field_list --> fixed_part semicolon variant_part |
```

```
                        fixed_part |
                        variant_part |

        fixed_part --> record_section_list

        record_section_list --> record_section semicolon
                                record_section_list |
                                record_section

        record_section --> field_identifier_list colon type

        field_identifier_list --> field_identifier comma
                                  field_identifier_list |
                                  field_identifier

        field_identifier --> identifier

        variant_part --> case_keyword tag_field type_identifier
                         of_keyword variant_list

        variant_list --> variant semicolon variant_list |
                         variant

        tag_field --> field_identifier colon |


        variant --> case_label_list colon left_parenthesis
                    field_list right_parenthesis

        case_label_list --> case_label comma case_label_list |
                            case_label

        case_label --> constant

        set_type --> set_keyword of_keyword base_type

        base_type --> simple_type

        file_type --> file_keyword of_keyword type

        pointer_type --> up_arrow type_identifier

        variable_declaration_part --> var_keyword
                                      variable_declaration_list
                                      semicolon |

        variable_declaration_list --> variable_declaration semicolon
                                      variable_declaration_list |
                                      variable_declaration

        variable_declaration --> identifier_list colon type

        procedure_and_function_declaration_part -->
                        procedure_or_function_declaration_list

        procedure_or_function_declaration_list -->
                        procedure_or_function_declaration semicolon
                         procedure_or_function_declaration_list |

        procedure_or_function_declaration --> procedure_declaration |
                                              function_declaration

```

```
procedure_declaration --> procedure_heading block

procedure_heading --> procedure_keyword identifier semicolon |
                       procedure_keyword identifier
                       left_parenthesis
                       formal_parameter_section_list
                       right_parenthesis semicolon

formal_parameter_section_list --> formal_parameter_section
                                  semicolon
                                  formal_parameter_section_list |
                                   formal_parameter_section

formal_parameter_section --> parameter_group |
                              var_keyword parameter_group |
                              function_keyword parameter_group |
                              procedure_keyword
                              identifier_list

parameter_group --> identifier_list colon type_identifier

function_declaration --> function_heading block

function_heading --> function_keyword identifier colon
                      result_type semicolon |
                      function_keyword identifier
                      left_parenthesis
                      formal_parameter_section_list
                      right_parenthesis
                      colon result_type semicolon

result_type --> type_identifier

statement_part --> compound_statement

statement --> label colon unlabelled_statement |
              unlabelled_statement

unlabelled_statement --> structured_statement |
                         simple_statement

simple_statement --> assignment_statement |
                      procedure_statement |
                      go_to_statement |
                      empty_statement

assignment_statement --> variable assignment_sign expression |
                         function_identifier assignment_sign
                        expression

variable --> entire_variable specifier_list |
             entire_variable

entire_variable --> identifier

specifier_list --> specifier specifier_list |
                   specifier

specifier --> up_arrow |
              record_field |
              array_index
```

```
record_field --> full_stop field_identifier

array_index --> left_bracket expression_list right_bracket

expression_list --> expression comma expression_list |
                    expression

expression --> simple_expression relational_operator
               simple_expression |
               simple_expression

relational_operator --> equal_sign |
                        unequal_sign |
                        less_sign |
                        less_equal_sign |
                        greater_equal_sign |
                        greater_sign |
                        in_keyword

simple_expression --> term adding_operator
                      simple_expression |
                      term |
                      sign term

adding_operator --> plus_sign |
                    minus_sign |
                    or_keyword

term --> factor multiplying_operator term |
         factor

multiplying_operator --> asterisk_sign |
                         slash_sign |
                         div_keyword |
                         mod_keyword |
                         and_keyword

factor --> unsigned_number |
           string |
           nil_keyword |
           left_parenthesis expression right_parenthesis |
           set |
           not_keyword factor |
           function_identifier left_parenthesis
           actual_parameter_list right_parenthesis |
           variable |
           constant_identifier |
           function_identifier

function_identifier --> identifier

set --> left_bracket element_list right_bracket

element_list --> element comma element_list |

element --> expression two_dots expression |
            expression

procedure_statement --> procedure_identifier left_parenthesis
                        actual_parameter_list
                        right_parenthesis |
                        procedure_identifier
```

```
procedure_identifier --> identifier

actual_parameter_list --> actual_parameter comma
                           actual_parameter_list |
                           actual_parameter

actual_parameter --> expression |
                      procedure_identifier |
                      function_identifier

go_to_statement --> goto_keyword label

empty_statement -->

structured_statement --> compound_statement |
                         conditional_statement |
                         repetitive_statement |
                         with_statement

compound_statement --> begin_keyword statement_list
                           end_keyword

conditional_statement --> if_statement |
                           case_statement

if_statement --> if_keyword expression then_keyword
                 statement else_keyword statement |
                 if_keyword expression then_keyword
                 statement

case_statement --> case_keyword expression of_keyword
                   case_list end_keyword

case_list --> case_list_element semicolon case_list |
              case_list_element

case_list_element --> case_label_list colon statement |

repetitive_statement --> while_statement |
                         repeat_statement |
                         for_statement

while_statement --> while_keyword expression do_keyword
                    statement

repeat_statement --> repeat_keyword statement_list
                     until_keyword expression

statement_list --> statement semicolon statement_list |
                   statement

for_statement --> for_keyword control_variable
                  assignment_sign for_list do_keyword
                  statement

for_list --> initial_value to_keyword final_value |
             initial_value downto_keyword final_value

control_variable --> identifier

initial_value --> expression
```

```

      final_value --> expression

      with_statement --> with_keyword record_variable_list
                           do_keyword statement

      record_variable_list --> variable comma
                                 record_variable_list |
                                 variable

      sign --> plus_sign |
                minus_sign
```

ModulaZ

```
Program
    --> Module

Module
    --> module identifier
      semi BlockNoScope identifier dot

Block
    -->
      BlockNoScope

BlockNoScope
    --> OptDecls begin Stmts end

OptDecls
    -->
    | Decls

Decls
    --> Declaration
    | Decls Declaration

Declaration
    --> const  OptConstDecls
    | type   OptTypeDecls
    | typerec   RecTypeDecls
    | var    OptVarDecls
    | ProcDecl

RecTypeDecls
    --> TypeDecl
    | RecTypeDecls and TypeDecl

OptConstDecls
    -->
    | ConstDecls

OptTypeDecls
    -->
    | TypeDecls

OptVarDecls
    -->
    | VarDecls

ConstDecls
```

```
    --> ConstDecl
    | ConstDecls ConstDecl

TypeDecls
    --> TypeDecl
    | TypeDecls TypeDecl

VarDecls
    --> VariableDecl
    | VarDecls VariableDecl

ConstDecl
    --> identifier colon Type equal Expr semi
    | identifier          equal Expr semi

TypeDecl
    --> identifier equal Type semi

VariableDecl
    --> IDList colon Type semi
    | IDList colon Type coloneq Expr semi
    | IDList          coloneq Expr semi

ProcDecl
    --> procedure identifier Signature
      equal BlockNoScope identifier semi

Signature
    --> lpar OptFormals rpar OptDeclType

OptDeclType
    -->
    | colon Type

OptFormals
    -->
    | Formals
    | Formals semi

Formals
    --> Formal
    | Formals semi Formal

Formal
    --> OptMode IDList colon Type

OptMode
    -->
    | value
    | var

Stmts
    -->
    | StmtList
    | StmtList semi

StmtList
    --> Stmt
    | StmtList semi Stmt

Stmt
    --> Block
```

```
     | AssignStmt
     | CallStmt
     | ExitStmt
     | EvalStmt
     | ForStmt
     | IfStmt
     | LoopStmt
     | ReadStmt
     | RepeatStmt
     | ReturnStmt
     | WhileStmt
     | WriteStmt

AssignStmt
    --> Expr coloneq Expr

CallStmt
    --> ProcCall

ExitStmt
    --> exit

EvalStmt
    --> eval Expr

ForStmt
    --> for identifier coloneq Expr to Expr OptBy do
      Stmts end

OptBy
    -->
    | by Expr

IfStmt
    --> IfHead end
    | IfHead else
      Stmts end

IfHead
    --> if Expr then
      Stmts
    | IfHead elsif
      Expr then
      Stmts

LoopStmt
    --> loop
      Stmts end

ReadStmt
    --> read lpar text comma ReadList rpar

ReadList
    --> Expr
    | Expr comma ReadList

RepeatStmt
    --> repeat
      Stmts until Expr

ReturnStmt
    --> return
```

```
    | return Expr

WhileStmt
    --> while
      Expr do
      Stmts end

WriteStmt
    --> write lpar text WriteList rpar

WriteList
    -->
    | comma Expr WriteList

Type
    --> ArrayType
    | RecordType
    | SubrangeType
    | identifier
    | lpar Type rpar
    | ref Type

ArrayType
    --> array OptTypeList of Type

OptTypeList
    -->
    | TypeList

TypeList
    --> Type
    | TypeList comma Type

RecordType
    --> record OptFields end

OptFields
    -->
    | Fields
    | Fields semi

Fields
    --> IDList colon Type
    | Fields semi IDList colon Type

SubrangeType
    --> lbrack Expr dotdot Expr rbrack

Expr
    --> Expr or Expr
    | Expr and Expr
    | not Expr
    | Expr Relop Expr
    | Expr Addop Expr
    | Expr Mulop Expr
    | Sign Expr
    | Primary

Relop
    --> equal
    | nequal
    | less
```

```
     | greater
     | le
     | ge

Addop
     --> plus
     | minus

Mulop
     --> star
     | div
     | mod

Sign
     --> plus
     | minus

Primary
     --> identifier
     | integer
     | real
     | text
     | lpar Expr rpar
     | Primary dot identifier
     | ArrayRef
     | ProcCall
     | Primary caret
     | new lpar Type rpar
     | nil lpar Type rpar

ArrayRef
     --> Primary lbrack rbrack
     | ArrayHead rbrack

ArrayHead
     --> Primary lbrack Expr
     | ArrayHead comma Expr

ProcCall
     --> Primary lpar rpar
     | CallHead rpar

CallHead
     --> Primary lpar Expr
     | CallHead comma Expr

IDList
     --> identifier
     | IDList comma identifier
```

EBNF

See above.