

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.

```

1 Grammar --> Symbols_List
2
3 Symbols_List --> Symbol Symbols_List
4           | Symbol
5
6 Symbol --> Identifier Right_Arrow Productions_List
7
8 Productions_List --> Production Productions_Separator Productions_List
9           | Production
10
11 Production --> Identifiers_List
12
13 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

```

1  Expr --> Expr plus Term
2      | Expr minus Term
3      | Term
4  Term --> Term times Factor
5      | Term divide Factor
6      | Factor
7  Factor --> lpar Expr rpar
8      | num
9      | id

```

JW_Pascal

```

1      program --> program_heading block full_stop
2
3      program_heading --> program_keyword identifier
4          left_parenthesis file_identifier_list
5          right_parenthesis semicolon
6
7      file_identifier_list --> file_identifier comma
8          file_identifier_list |
9          file_identifier
10
11     file_identifier --> identifier
12
13     block --> label_declaration_part constant_definition_part
14         type_definition_part variable_declaration_part
15         procedure_and_function_declaration_part
16         statement_part
17
18     label_declaration_part --> label_keyword label_list semicolon |
19
20
21     label_list --> label comma label_list |
22         label
23
24     label --> unsigned_integer
25
26     constant_definition_part --> const_keyword
27         constant_definition_list
28         semicolon |
29
30     constant_definition_list --> constant_definition semicolon
31         constant_definition_list |
32         constant_definition

```

```
33      constant_definition --> identifier equal_sign constant
34
35      constant --> unsigned_number |
36          sign unsigned_number |
37          constant_identifier |
38          sign constant_identifier |
39          string
40
41      unsigned_number --> unsigned_integer |
42          unsigned_real
43
44      constant_identifier --> identifier
45
46      type_definition_part --> type_keyword type_definition_list
47          semicolon |
48
49      type_definition_list --> type_definition semicolon
50          type_definition_list |
51          type_definition
52
53      type_definition --> identifier equal_sign type
54
55      type --> simple_type |
56          structured_type |
57          pointer_type
58
59      simple_type --> scalar_type |
60          subrange_type |
61          type_identifier
62
63      scalar_type --> left_parenthesis identifier_list
64          right_parenthesis
65
66      identifier_list --> identifier comma identifier_list |
67          identifier
68
69      subrange_type --> constant two_dots constant
70
71      type_identifier --> identifier
72
73      structured_type --> packed_keyword unpacked_structured_type |
74          unpacked_structured_type
75
76      unpacked_structured_type --> array_type |
77          record_type |
78          set_type |
79          file_type
80
81      array_type --> array_keyword left_bracket index_type_list
82          right_bracket of_keyword component_type
83
84      index_type_list --> index_type comma index_type_list |
85          index_type
86
87      index_type --> simple_type
88
89      component_type --> type
90
91      record_type --> record_keyword field_list end_keyword
92
93      field_list --> fixed_part semicolon variant_part |
```

```
95          fixed_part |
96          variant_part |
97
98      fixed_part --> record_section_list
99
100     record_section_list --> record_section semicolon
101             record_section_list |
102             record_section
103
104     record_section --> field_identifier_list colon type
105
106     field_identifier_list --> field_identifier comma
107             field_identifier_list |
108             field_identifier
109
110     field_identifier --> identifier
111
112     variant_part --> case_keyword tag_field type_identifier
113             of_keyword variant_list
114
115     variant_list --> variant semicolon variant_list |
116             variant
117
118     tag_field --> field_identifier colon |
119
120
121     variant --> case_label_list colon left_parenthesis
122             field_list right_parenthesis
123
124     case_label_list --> case_label comma case_label_list |
125             case_label
126
127     case_label --> constant
128
129     set_type --> set_keyword of_keyword base_type
130
131     base_type --> simple_type
132
133     file_type --> file_keyword of_keyword type
134
135     pointer_type --> up_arrow type_identifier
136
137     variable_declaration_part --> var_keyword
138             variable_declaration_list
139             semicolon |
140
141     variable_declaration_list --> variable_declaration semicolon
142             variable_declaration_list |
143             variable_declaration
144
145     variable_declaration --> identifier_list colon type
146
147     procedure_and_function_declaration_part -->
148             procedure_or_function_declaration_list
149
150     procedure_or_function_declaration_list -->
151             procedure_or_function_declaration semicolon
152             procedure_or_function_declaration_list |
153
154     procedure_or_function_declaration --> procedure_declaration |
155             function_declaration
156
```

```
157      procedure_declaration --> procedure_heading block
158
159      procedure_heading --> procedure_keyword identifier semicolon |
160                      procedure_keyword identifier
161                      left_parenthesis
162                      formal_parameter_section_list
163                      right_parenthesis semicolon
164
165      formal_parameter_section_list --> formal_parameter_section
166                      semicolon
167                      formal_parameter_section_list |
168                      formal_parameter_section
169
170      formal_parameter_section --> parameter_group |
171                      var_keyword parameter_group |
172                      function_keyword parameter_group |
173                      procedure_keyword
174                      identifier_list
175
176      parameter_group --> identifier_list colon type_identifier
177
178      function_declaration --> function_heading block
179
180      function_heading --> function_keyword identifier colon
181                      result_type semicolon |
182                      function_keyword identifier
183                      left_parenthesis
184                      formal_parameter_section_list
185                      right_parenthesis
186                      colon result_type semicolon
187
188      result_type --> type_identifier
189
190      statement_part --> compound_statement
191
192      statement --> label colon unlabelled_statement |
193                      unlabelled_statement
194
195      unlabelled_statement --> structured_statement |
196                      simple_statement
197
198      simple_statement --> assignment_statement |
199                      procedure_statement |
200                      go_to_statement |
201                      empty_statement
202
203      assignment_statement --> variable assignment_sign expression |
204                      function_identifier assignment_sign
205                      expression
206
207      variable --> entire_variable specifier_list |
208                      entire_variable
209
210      entire_variable --> identifier
211
212      specifier_list --> specifier specifier_list |
213                      specifier
214
215      specifier --> up_arrow |
216                      record_field |
217                      array_index
218
```

```
219     record_field --> full_stop field_identifier
220
221     array_index --> left_bracket expression_list right_bracket
222
223     expression_list --> expression comma expression_list |
224         expression
225
226     expression --> simple_expression relational_operator
227         simple_expression |
228         simple_expression
229
230     relational_operator --> equal_sign |
231         unequal_sign |
232         less_sign |
233         less_equal_sign |
234         greater_equal_sign |
235         greater_sign |
236         in_keyword
237
238     simple_expression --> term adding_operator
239         simple_expression |
240         term |
241         sign term
242
243     adding_operator --> plus_sign |
244         minus_sign |
245         or_keyword
246
247     term --> factor multiplying_operator term |
248         factor
249
250     multiplying_operator --> asterisk_sign |
251         slash_sign |
252         div_keyword |
253         mod_keyword |
254         and_keyword
255
256     factor --> unsigned_number |
257         string |
258         nil_keyword |
259         left_parenthesis expression right_parenthesis |
260         set |
261         not_keyword factor |
262         function_identifier left_parenthesis
263         actual_parameter_list right_parenthesis |
264         variable |
265         constant_identifier |
266         function_identifier
267
268     function_identifier --> identifier
269
270     set --> left_bracket element_list right_bracket
271
272     element_list --> element comma element_list |
273
274     element --> expression two_dots expression |
275         expression
276
277     procedure_statement --> procedure_identifier left_parenthesis
278         actual_parameter_list
279         right_parenthesis |
280         procedure_identifier
```

```
281      procedure_identifier --> identifier
282
283      actual_parameter_list --> actual_parameter comma
284                      actual_parameter_list |
285                      actual_parameter
286
287      actual_parameter --> expression |
288                      procedure_identifier |
289                      function_identifier
290
291      go_to_statement --> goto_keyword label
292
293      empty_statement -->
294
295      structured_statement --> compound_statement |
296                      conditional_statement |
297                      repetitive_statement |
298                      with_statement
299
300      compound_statement --> begin_keyword statement_list
301                      end_keyword
302
303      conditional_statement --> if_statement |
304                      case_statement
305
306      if_statement --> if_keyword expression then_keyword
307                      statement else_keyword statement |
308                      if_keyword expression then_keyword
309                      statement
310
311      case_statement --> case_keyword expression of_keyword
312                      case_list end_keyword
313
314      case_list --> case_list_element semicolon case_list |
315                      case_list_element
316
317      case_list_element --> case_label_list colon statement |
318
319      repetitive_statement --> while_statement |
320                      repeat_statement |
321                      for_statement
322
323      while_statement --> while_keyword expression do_keyword
324                      statement
325
326      repeat_statement --> repeat_keyword statement_list
327                      until_keyword expression
328
329      statement_list --> statement semicolon statement_list |
330                      statement
331
332      for_statement --> for_keyword control_variable
333                      assignment_sign for_list do_keyword
334                      statement
335
336      for_list --> initial_value to_keyword final_value |
337                      initial_value downto_keyword final_value
338
339      control_variable --> identifier
340
341      initial_value --> expression
```

```

343      final_value --> expression
344
345      with_statement --> with_keyword record_variable_list
346          do_keyword statement
347
348      record_variable_list --> variable comma
349          record_variable_list |
350          variable
351
352      sign --> plus_sign |
353          minus_sign

```

ModulaZ

```

1  Program
2      --> Module
3
4  Module
5      --> module identifier
6      semi BlockNoScope identifier dot
7
8  Block
9      -->
10     BlockNoScope
11
12 BlockNoScope
13     --> OptDecls begin Stmtts end
14
15 OptDecls
16     -->
17     | Decls
18
19 Decls
20     --> Declaration
21     | Decls Declaration
22
23 Declaration
24     --> const OptConstDecls
25     | type OptTypeDecls
26     | typerec RecTypeDecls
27     | var OptVarDecls
28     | ProcDecl
29
30 RecTypeDecls
31     --> TypeDecl
32     | RecTypeDecls and TypeDecl
33
34 OptConstDecls
35     -->
36     | ConstDecls
37
38 OptTypeDecls
39     -->
40     | TypeDecls
41
42 OptVarDecls
43     -->
44     | VarDecls
45
46 ConstDecls

```

```
47      --> ConstDecl
48      | ConstDecls ConstDecl
49
50 TypeDecls
51      --> TypeDecl
52      | TypeDecls TypeDecl
53
54 VarDecls
55      --> VariableDecl
56      | VarDecls VariableDecl
57
58 ConstDecl
59      --> identifier colon Type equal Expr semi
60      | identifier           equal Expr semi
61
62 TypeDecl
63      --> identifier equal Type semi
64
65 VariableDecl
66      --> IDList colon Type semi
67      | IDList colon Type coloneq Expr semi
68      | IDList           coloneq Expr semi
69
70 ProcDecl
71      --> procedure identifier Signature
72          equal BlockNoScope identifier semi
73
74 Signature
75      --> lpar OptFormals rpar OptDeclType
76
77 OptDeclType
78      -->
79      | colon Type
80
81 OptFormals
82      -->
83      | Formals
84      | Formals semi
85
86 Formals
87      --> Formal
88      | Formals semi Formal
89
90 Formal
91      --> OptMode IDList colon Type
92
93 OptMode
94      -->
95      | value
96      | var
97
98 Stmtns
99      -->
100     | StmtList
101     | StmtList semi
102
103 StmtList
104     --> Stmt
105     | StmtList semi Stmt
106
107 Stmt
108     --> Block
```

```
109      | AssignStmt
110      | CallStmt
111      | ExitStmt
112      | EvalStmt
113      | ForStmt
114      | IfStmt
115      | LoopStmt
116      | ReadStmt
117      | RepeatStmt
118      | ReturnStmt
119      | WhileStmt
120      | WriteStmt
121
122 AssignStmt
123     --> Expr coloneq Expr
124
125 CallStmt
126     --> ProcCall
127
128 ExitStmt
129     --> exit
130
131 EvalStmt
132     --> eval Expr
133
134 ForStmt
135     --> for identifier coloneq Expr to Expr OptBy do
136         Stmts end
137
138 OptBy
139     -->
140     | by Expr
141
142 IfStmt
143     --> IfHead end
144     | IfHead else
145         Stmts end
146
147 IfHead
148     --> if Expr then
149         Stmts
150     | IfHead elsif
151         Expr then
152         Stmts
153
154 LoopStmt
155     --> loop
156         Stmts end
157
158 ReadStmt
159     --> read lpar text comma ReadList rpar
160
161 ReadList
162     --> Expr
163     | Expr comma ReadList
164
165 RepeatStmt
166     --> repeat
167         Stmts until Expr
168
169 ReturnStmt
170     --> return
```

```

171      | return Expr
172
173 WhileStmt
174     --> while
175     Expr do
176     Stmts end
177
178 WriteStmt
179     --> write lpar text WriteList rpar
180
181 WriteList
182     -->
183     | comma Expr WriteList
184
185 Type
186     --> ArrayType
187     | RecordType
188     | SubrangeType
189     | identifier
190     | lpar Type rpar
191     | ref Type
192
193 ArrayType
194     --> array OptTypeList of Type
195
196 OptTypeList
197     -->
198     | TypeList
199
200 TypeList
201     --> Type
202     | TypeList comma Type
203
204 RecordType
205     --> record OptFields end
206
207 OptFields
208     -->
209     | Fields
210     | Fields semi
211
212 Fields
213     --> IDList colon Type
214     | Fields semi IDList colon Type
215
216 SubrangeType
217     --> lbrack Expr dotdot Expr rbrack
218
219 Expr
220     --> Expr or Expr
221     | Expr and Expr
222     | not Expr
223     | Expr Relop Expr
224     | Expr Addop Expr
225     | Expr Mulop Expr
226     | Sign Expr
227     | Primary
228
229 Relop
230     --> equal
231     | nequal
232     | less

```

```

233      | greater
234      | le
235      | ge
236
237 Addop
238     --> plus
239     | minus
240
241 Mulop
242     --> star
243     | div
244     | mod
245
246 Sign
247     --> plus
248     | minus
249
250 Primary
251     --> identifier
252     | integer
253     | real
254     | text
255     | lpar Expr rpar
256     | Primary dot identifier
257     | ArrayRef
258     | ProcCall
259     | Primary caret
260     | new lpar Type rpar
261     | nil lpar Type rpar
262
263 ArrayRef
264     --> Primary lbrack rbrack
265     | ArrayHead rbrack
266
267 ArrayHead
268     --> Primary lbrack Expr
269     | ArrayHead comma Expr
270
271 ProcCall
272     --> Primary lpar rpar
273     | CallHead rpar
274
275 CallHead
276     --> Primary lpar Expr
277     | CallHead comma Expr
278
279 IDList
280     --> identifier
281     | IDList comma identifier

```

EBNF

See above.