An LALR(1) Grammar for (Revised) Ada

G. Persch, G. Winterstein
S. Drossopoulou, M. Dausmann

Universität Karlsruhe, D-7500 Karlsruhe 1

To speed up the development and to facilitate the validation of Ada compilers we propose to use a common LALR(1) grammar for bottom-up parsing. The emphasis in specifying such a grammar especially in removing the conflicts is to obtain the same (syntactic) language as described in the Reference Manual [R-Ada].

In transforming the initial grammar given in [R-ADA] first all obvious ambiguities have been removed. Most of them arise from the use of the nonterminal type_mark which is equivalent to name. The more severe LALR(1) conflicts are the identifier list in declarations, indexed names and similiar constructs, the sequencing in declarative parts [Cro], subprogram bodies and instantiations, and entry names in accept statements.

- Identifier lists are substituted by a single identifier or a list of at least two identifiers.

- Indexed names have been extended to allow also function calls, generic associations, slices, conversions, and type marks with index or discriminant constraints.

- The following idea has been used to resolve the sequencing in declarative parts. Assume the rules

  dp ::= {d} {r} {p}
  d ::= b | ...
  p ::= b | z

  Then if b can be reduced to d or p it is not clear whether {r} is empty. The following transformation overcomes this ambiguity:

  dp ::= {d} r {r} {p} | {d} | {d} z {p}

- The conflict concerning subprogram instantiation has been avoided by comprising the subprogram specification and the terminal 'is' into one rule.

- The conflict in analyzing the entry name in an accept statement is resolved by using semantic information about entry names. They can only consist of a sequence of selected components eventually followed by one index.

The (misprinted ?) rule for entry calls which requires two opening and closing parenthesis has been replaced by

entry_call ::= procedure_call

The grammar given in the remainder of this paper uses the same notation as in [R-Ada] except that terminals are enclosed by quotes. Sequences {, } and options [,] have been transformed to get pure BNF.

Our LALR(1) parser generator [PGS] determines 484 states for the grammar given below. The number of productions is 397, there are 93 (including the stop symbol) terminal symbols, and 181 nonterminals which can be reduced to 124 by elimination of chain productions. The size of the resulting tables including automatic error recovery is about 12 KB.

Acknowledgement

The work reported here is part of a project for the development of an Ada compiler financed by the Bundesamt für Wehrtechnik und Beschaffung, Vertrag-Nr.: E/F61D/90104/95031

References

[R-ADA] US Department of Defense
Reference Manual for the Ada Programming Language
July 1980

[PGS] P. Dencker
Ein neues LALR-System
Diplomarbeit, Universität Karlsrune

[Cro] M. T. Cronin
Ambiguity in Ada
Sigplan Notices 15-1-80, pp. 6-7

```
pragma ::= 'pragma' identifier ';'
  | 'pragma' identifier '(' argument_list ')' ';'

argument_list ::= argument | argument_list ',' argument

argument ::= expression | identifier '=>' expression


declaration ::=
    object_declaration        | number_declaration
  | type_declaration          | subtype_declaration
  | subprogram_declaration    | package_declaration
  | task_declaration          | exception_declaration
  | renaming_declaration
  | pragma

object_declaration ::= component_declaration
  | identifier       ':' 'constant' subtype_indication
                                    initialization_option ';'
  | identifier       ':' 'constant' array_type_definition
                                    initialization_option ';'
  | identifier_list2':' 'constant' subtype_indication
                                    initialization_option ';'
  | identifier_list2':' 'constant' array_type_definition
                                    initialization_option ';'

initialization_option ::=  | ':=' expression

number_declaration ::=
    identifier       ':' 'constant' ':=' expression ';'
  | identifier_list2':' 'constant' ':=' expression ';'

identifier_list2 ::= identifier ',' identifier
  | identifier_list2',' identifier

type_declaration ::=
   'type' identifier discriminant_part_option 'is' type_defi-
nition ';'
  | incomplete_type_declaration

discriminant_part_option ::= | discriminant_part

type_definition ::=
    enumeration_type_definition | integer_type_definition
  | real_type_definition        | array_type_definition
  | record_type_definition      | access_type_definition
  | derived_type_definition     | private_type_definition

subtype_declaration ::=
   'subtype' identifier 'is' subtype_indication ';'

subtype_indication ::= name
  | subtype_indication_with_constraint
```

```
pragma ::= 'pragma' identifier ';'
  | 'pragma' identifier '(' argument_list ')' ';'

argument_list ::= argument | argument_list ',' argument

argument ::= expression | identifier '=>' expression


declaration ::=
   object_declaration       | number_declaration
  | type_declaration         | subtype_declaration
  | subprogram_declaration   | package_declaration
  | task_declaration         | exception_declaration
  | renaming_declaration
  | pragma

object_declaration ::= component_declaration
  | identifier      ':' 'constant' subtype_indication
                                   initialization_option ';'
  | identifier      ':' 'constant' array_type_definition
                                   initialization_option ';'
  | identifier_list2':' 'constant' subtype_indication
                                   initialization_option ';'
  | identifier_list2':' 'constant' array_type_definition
                                   initialization_option ';'

initialization_option ::=  | ':=' expression

number_declaration ::=
   identifier      ':' 'constant' ':=' expression ';'
  | identifier_list2':' 'constant' ':=' expression ';'

identifier_list2 ::= identifier ',' identifier
  | identifier_list2',' identifier

type_declaration ::=
  'type' identifier discriminant_part_option 'is' type_defi-
nition ';'
  | incomplete_type_declaration

discriminant_part_option ::= | discriminant_part

type_definition ::=
   enumeration_type_definition | integer_type_definition
  | real_type_definition       | array_type_definition
  | record_type_definition     | access_type_definition
  | derived_type_definition    | private_type_definition

subtype_declaration ::=
   'subtype' identifier 'is' subtype_indication ';'

subtype_indication ::= name
  | subtype_indication_with_constraint
```

```
subtype_indication_with_constraint ::=
   name range_constraint
 | name accuracy_constraint

derived_type_definition ::= 'new' subtype_indication

range_constraint ::= 'range' range

range ::= simple_expression '..' simple_expression

enumeration_type_definition ::=
   '(' enumeration_literal_list  ')'

enumeration_literal_list ::= enumeration_literal
 | enumeration_literal_list ',' enumeration_literal

enumeration_literal ::= identifier | character_literal

integer_type_definition ::= range_constraint

real_type_definition ::= accuracy_constraint

accuracy_constraint ::=
   floating_point_constraint | fixed_point_constraint

floating_point_constraint ::=
   'digits' simple_expression range_constraint_option

range_constraint_option ::=  | range_constraint

fixed_point_constraint  ::=
   'delta' simple_expression range_constraint_option

array_type_definition ::=
   'array' '(' index_list    ')' 'of' subtype_indication
 | 'array' index_constraint      'of' subtype_indication

index_list ::= index | index_list ',' index

index ::= name 'range' '<>'

index_constraint ::= '('discrete_range_list  ')'

discrete_range_list ::= discrete_range
 | discrete_range_list ',' discrete_range

discrete_range ::= name range_constraint_option | range
record_type_definition :
   'record'
      component_list
   'end' 'record'

component_list ::=
   component_declaration_listØ variant_part_option | 'null
' ';'
```

```
component_declaration_list0 ::=
 | component_declaration_list0  component_declaration

variant_part_option ::=  | variant_part

component_declaration ::=
   discriminant_declaration ';'
 | identifier       ':' array_type_definition
                                initialization_option ';'
 | identifier_list2':' array_type_definition
                                initialization_option ';'
discriminant_part :
   '(' discriminant_declaration_list  ')'

discriminant_declaration_list ::= discriminant_declaration
 | discriminant_declaration_list ';' discriminant_declara-
tion

discriminant_declaration ::=
   identifier       ':' subtype_indication
                                initialization_option
 | identifier_list2':' subtype_indication
                                initialization_option

variant_part ::=
   'case' name 'is'
      variant_list0
   'end' 'case' ';'

variant_list0 ::=
 | variant_list0 'when' choice_list '=>' component_list

choice ::= simple_expression        | 'others'
         | name range_constraint | range


choice_list ::= choice | choice_list '|' choice

access_type_definition ::= 'access' subtype_indication

incomplete_type_declaration ::=
   'type' identifier discriminant_part_option ';'

declarative_part ::=
   declarative_item_list0
 | declarative_item_list0  representation_spec_list1
   program_component_list0
 | declarative_item_list0
   body_or_stub
   program_component_list0

declarative_item_list0 ::=
 | declarative_item_list0  declaration
 | declarative_item_list0  use_clause
```

```
representation_spec_list0 ::= | representation_spec_list1

representation_spec_list1 ::=
   representation_specification
 | representation_spec_list1  representation_specification

body_or_stub ::= body | body_stub

program_component_list0 ::=
 | program_component_list0 program_component

program_component ::=   body
 | package_declaration | task_declaration | body_stub

body ::= subprogram_body | package_body | task_body

name ::= identifier
 | indexed_component
 | selected_component | attribute
 | function_call      | operator_symbol

indexed_component ::= name '(' generalized_expression_list
 ')'

generalized_expression_list ::= generalized_expression
 | generalized_expression_list ',' generalized_expression

generalized_expression ::=
   expression | range | subtype_indication_with_constraint
 | choice_list '=>' expression
 | choice_list '=>' subtype_indication_with_constraint

selected_component ::=
   name '.' identifier      | name '.' 'all'
 | name '.' operator_symbol

attribute ::= name ''' identifier

literal  ::=
   numeric_literal | character_literal
 | 'null'

aggregate ::=  '(' component_association_list2 ')'
           | '(' choice_list '=>' expression ')'

component_association_list2 ::=
   component_association ',' component_association
 | component_association_list2 ',' component_association

component_association ::= expression
     | choice_list '=>' expression

expression ::= relation
 | and_expression | or_expression | xor_expression
```

```
 | andthen_expression | orelse_expression

and_expression ::= relation 'and' relation
 |         and_expression 'and' relation

or_expression ::= relation 'or' relation
 |         or_expression 'or' relation

xor_expression ::= relation 'xor' relation
 |         xor_expression 'xor' relation

andthen_expression ::= relation 'and' 'then' relation
 |         andthen_expression 'and' 'then' relation

orelse_expression ::= relation 'or' 'else' relation
 |         orelse_expression 'or' 'else' relation

relation ::= simple_expression
 | simple_expression relational_operator simple_expression
 | simple_expression membership_operator range
 | simple_expression membership_operator subtype_indication

membership_operator ::= 'in' | 'not' 'in'

simple_expression ::= term_list
  |  unary_operator term_list

term_list ::= term | term_list adding_operator term

term ::= factor | term multiplying_operator factor

factor ::= primary | primary '**' primary
primary :
   literal | aggregate | name | allocator
                    | qualified_expression | '('expression')'

relational_operator ::= '=' | '/=' | '<' | '<=' | '>' | '>='

adding_operator ::= '+' | '-' | '&'

unary_operator ::= '+' | '-' | 'not'

multiplying_operator ::= '*' | '/' | 'mod' | 'rem'

qualified_expression :
   name'''' '('expression')' | name''''aggregate

allocator ::= 'new' name


sequence_of_statements ::=  statement
 | sequence_of_statements statement

statement ::=
   label_list simple_statement
```

```
 | label_list compound_statement
 | pragma

label_list ::= | label_list label

simple_statement ::= null_statement
 | assignment_statement | exit_statement
 | return_statement     | goto_statement
 | procedure_call
 | delay_statement      | abort_statement
 | raise_statement      | code_statement

compound_statement ::=
   if_statement         | case_statement
 | loop_statement       | block
 | accept_statement     | select_statement

label ::= '<<' identifier '>>'

null_statement ::= 'null' ';'

assignment_statement ::= name ':=' expression ';'

if_statement ::=
     'if' condition 'then'
        sequence_of_statements
     elsif_list0
     else_option
     'end' 'if' ';'

elsif_list0 ::=
 | elsif_list0
   'elsif' condition 'then' sequence_of_statements

else_option ::= | 'else' sequence_of_statements

condition ::= expression

case_statement ::=
   'case' expression 'is'
      alternative_list0
   'end' 'case' ';'

alternative_list0 ::=
 | alternative_list0
   'when' choice_list '=>' sequence_of_statements

loop_statement ::=
   iteration_clause_option basic_loop ';'
 | identifier ':' iteration_clause_option
                  basic_loop identifier';'

basic_loop ::=
   'loop'
      sequence_of_statements
```

```
   'end' 'loop'

iteration_clause_option ::=
 | 'for' identifier 'in'           discrete_range
 | 'for' identifier 'in' 'reverse' discrete_range
 | 'while' condition

block ::=
     declare_part_option
    'begin'
       sequence_of_statements
     exception_option
    'end' ';'
 | identifier ':'
     declare_part_option
    'begin'
       sequence_of_statements
     exception_option
    'end' identifier ';'

declare_part_option ::=
 | 'declare' declarative_part

exception_option ::=
 | 'exception' exception_handler_list0

exception_handler_list0 ::=
 | exception_handler_list0   exception_handler

exit_statement ::=
   'exit' name_option when_option ';'

name_option ::= | name

when_option ::= | 'when' condition

return_statement ::= 'return' ';'
                   | 'return' expression ';'

goto_statement ::= 'goto' name ';'

subprogram_declaration ::= subprogram_specification ';'
 | generic_subprogram_declaration
 | generic_subprogram_instantiation

subprogram_specification ::=
   'procedure' identifier formal_part_option
 | 'function'  designator formal_part_option
                          'return' subtype_indication

subprogram_specification_is ::=
   'procedure' identifier                        'is'
 | 'procedure' identifier formal_part            'is'
 | 'function'  designator formal_part_option
                         'return' subtype_indication 'is'
```

```
designator ::= identifier | operator_symbol

operator_symbol ::= character_string

formal_part ::= '(' parameter_declaration_list  ')'

formal_part_option ::= | formal_part

parameter_declaration_list ::= parameter_declaration
 | parameter_declaration_list ';' parameter_declaration

parameter_declaration ::=
   identifier      ':' mode subtype_indication
                                  initialization_option
 | identifier_list2':' mode subtype_indication
                                  initialization_option

mode ::= | 'in' | 'out' | 'in' 'out'

subprogram_body ::=
   subprogram_specification_is
      declaration_part
  'begin'
      sequence_of_statements
   exception_option
  'end' designator_option ';'

designator_option ::= | designator

procedure_call ::= name ';'


function_call ::= name '(' ')'


package_declaration ::= package_specification ';'
 | generic_package_declaration
 | generic_package_instantiation

package_specification ::=
   'package' identifier 'is'
       declarative_item_list0
    private_part_option
   'end' identifier_option

private_part_option ::=
 | 'private'
       declarative_item_list0
       representation_spec_list0

identifier_option ::= | identifier

package_body ::=
   'package' 'body' identifier 'is'
```

```
      declarative_part
    statements_option
   'end' identifier_option ';'

statements_option ::=
 | 'begin' sequence_of_statements exception_option

private_type_definition ::= 'limited' 'private'
                          | 'private'

use_clause ::= 'use' name_list ';'

name_list  ::= name | name_list ',' name

renaming_declaration ::=
   identifier ':' name         'renames' name ';'
 | identifier ':' 'exception' 'renames' name ';'
 | 'package' identifier        'renames' name ';'
 | 'task'    identifier        'renames' name ';'
 | subprogram_specification    'renames' name ';'

task_declaration ::= task_specification

task_specification ::= 'task' identifier task_specifier
 | 'task' 'type' identifier task_specifier

task_specifier ::=  ';'
 | 'is'
        entry_declaration_list0
        representation_spec_list0
   'end' identifier_option ';'

entry_declaration_list0 ::=
 | entry_declaration_list0  entry_declaration

task_body ::=
   'task' 'body' identifier 'is'
      declarative_part
   'begin'
      sequence_of_statements
    exception_option
   'end' identifier_option ';'

entry_declaration ::=
   'entry' identifier formal_part_option';'
 | 'entry' identifier '('discrete_range')'
                              formal_part_option';'

entry_call ::= procedure_call

accept_statement ::=
   'accept' entry_name formal_part_option ';'
 | 'accept' entry_name formal_part_option 'do'
      sequence_of_statements
   'end' identifier_option ';'
```

```
entry_name ::= identifier | operator_symbol
             | entry_name '.' identifier
             | entry_name '.' operator_symbol
             | entry_name '(' expression ')'


delay_statement ::= 'delay' simple_expression';'

select_statement ::= selective_wait
 | conditional_entry_call | timed_entry_call

selective_wait ::=
   'select'
       condition_option
          select_alternative
    select_alternative_list0
    else_option
   'end' 'select' ';'

select_alternative_list0 ::=
 | select_alternative_list0
   'or' condition_option select_alternative

condition_option ::= | 'when' condition '=>'

select_alternative ::=
   accept_statement sequence_of_statements_option
 | delay_statement  sequence_of_statements_option
 | 'terminate' ';'

sequence_of_statements_option ::=
 | sequence_of_statements

conditional_entry_call ::=
   'select'
       entry_call sequence_of_statements_option
   'else'
       sequence_of_statements
   'end' 'select' ';'

timed_entry_call ::=
   'select'
       entry_call sequence_of_statements_option
   'or'
       delay_statement sequence_of_statements_option
   'end' 'select' ';'

abort_statement ::= 'abort' name_list';'

compilation ::= compilation_list

compilation_list ::=
   pragma_list0  compilation_unit
 | compilation_list  pragma_list0  compilation_unit
```

```
pragma_list0 ::= | pragma_list0  pragma

compilation_unit ::=
   context_specification subprogram_declaration
 | context_specification subprogram_body
 | context_specification package_declaration
 | context_specification package_body
 | context_specification subunit

context_specification ::=
 | context_specification  with_clause use_clause_option

use_clause_option ::= | use_clause

with_clause ::= 'with' name_list';'

subunit ::=
   'separate' '(' name')' body

body_stub ::=
   subprogram_specification_is      'separate' ';'
 | 'package' 'body' identifier 'is' 'separate' ';'
 | 'task' 'body' identifier    'is' 'separate' ';'

exception_declaration ::= identifier      ':' 'exception
' ';'


exception_handler ::=
   'when' exception_choice_list '=>'
       sequence_of_statements

exception_choice_list ::= exception_choice
 | exception_choice_list '|' exception_choice

exception_choice ::= name | 'others'

raise_statement ::= 'raise' name_option ';'
generic_subprogram_declaration :
   generic_part subprogram_specification';'
generic_package_declaration :
   generic_part package_specification';'

generic_part ::= 'generic'
 | generic_part  generic_formal_parameter

generic_formal_parameter ::=
   parameter_declaration';'
 | 'type' identifier discriminant_part_option 'is'
                                    generic_type_definition
';'
 | 'with' subprogram_specification ';'
 | 'with' subprogram_specification_is name ';'
 | 'with' subprogram_specification_is '<>' ';'
```

```
generic_type_definition ::=
   '(' '<>' ')' | 'range' '<>' | 'delta' '<>' | 'digits
' '<>'
 | array_type_definition        | access_type_definition
 | private_type_definition

generic_subprogram_instantiation ::=
   'procedure' identifier 'is' generic_instantiation';'
 | 'function' designator  'is' generic_instantiation';'

generic_package_instantiation ::=
   'package' identifier 'is' generic_instantiation ';'

generic_instantiation ::= 'new' name

representation_specification ::=
   length_specification
 | record_type_representation | address_specification

length_specification ::= 'for' name 'use' expression';'

record_type_representation ::=
   'for' name 'use'
      'record' alignment_clause_option
         location_list0
      'end' 'record' ';'

location_list0 ::=
 | location_list0
   name 'at' simple_expression 'range' range ';'

alignment_clause_option ::= | 'at' 'mod' simple_expression
';'

address_specification ::= 'for' name 'use' 'at' simple_ex-
pression';'

code_statement ::= qualified_expression';'
```