Appendix E

Error Recovery Examples

Erroneous Pascal and Ada programs were processed with the error recovery method described in this thesis. The first Pascal program was designed to demonstrate the effectiveness of the scope recovery. The second program was constructed from some interesting examples drawn from [20]. The Ada example is taken from [37]. The Pascal2 grammar of Appendix F and the Ada grammar of [31] were used exactly as is - no user-supplied scopes were used. Thus, the results presented here were obtained automatically from the specification of the input grammars with no intervention, whatsoever, from the user.

Pascal2 Example 1

```
 1. program test(input,output);
 2.     var i,j,k : integer;
 3.         l,m,n : integer;
 4.         x,y,z : real;
 5.         final : real;
 6.         a,b,c : array [1..10] of real;
 7. procedure sub(i,j,k:integer; x,y,z:real);
 8.     var v,w : integer;
 9.     begin
10.         a := (( 3 ]];
                   ^
                   ^
                     <>
***Error: ")" inserted to complete phrase
***Error: ")" inserted to complete phrase
***Error: Unexpected input discarded
11.         v := i + j + k;
12.         w := x + y + z;
                         ^
***Error: "END" inserted to complete phrase started at line 9, column 5

13.
14. function f(x,y:real) : result real;
                                    ^
```

```
***Error: Unexpected symbol ignored
    15.     var z : real;
    16.     begin
    17.         z := x + y
                         ^
                         ^
***Error: "END" inserted to complete phrase started at line 16, column 5
***Error: ";" inserted to complete phrase started at line 7, column 1
    18.
    19. begin
    20.     i := 1;
    21.     while i <= 10 do
    22.         begin
    23.         j := 1;
    24.         while j <= 10 do
    25.             begin
    26.             a[i] := a[i] + b[j] + c[j];
    27.             j := j + 1
    28.         end;
    29.         i := i + 1
                         ^
***Error: ; expected after this token
    30.     final := f(x,y,z)*2.0/i+j+345.9;
    31.     while (x = y) do begin
    32.         if y = z then begin
    33.             sub;
    34.             x := ((y + z
                                 ^
                                 ^
                                 ^
                                 ^
                                 ^
                                 ^
                                 ^
                                 ^
***Error: ")" inserted to complete phrase
***Error: ")" inserted to complete phrase
***Error: "END" inserted to complete phrase started at line 32, column 23
***Error: "ELSE" inserted to complete phrase started at line 32, column 9
***Error: "END" inserted to complete phrase started at line 31, column 22
***Error: "END" inserted to complete phrase started at line 22, column 9
***Error: "END" inserted to complete phrase started at line 19, column 1
***Error: . expected after this token
```

Pascal2 Example 2

```
     1. PROGRAM P(INPUT,OUTPUT);
     2. BEGIN
     3.     FOR K1 := TO NOELEMS DO X:=1
                    ^
                                       ^
***Error: initial_value expected after this token
***Error: ; expected after this token
     4.     IF X=1 THEN
     5.        BEGIN
     6.            WRITELN('0*END OF SORT*')
                                             ^
***Error: "END" inserted to complete phrase started at line 5, column 8

     7.     ELSE
     8.        WRITELN('0***LOOP DETECTED IN INPUT ORDER RELATIONS***');
     9. END.
    10.
    11. PROGRAM HUNTER\INPUT,OUTPUT'?
                    ^
        <-------------------------->
***Error: ( expected instead of this token
***Error: program_heading expected instead
    12.     VAR Q:INTEGER;
    13. BEGIN
    14. END.
    15.
    16. PROGRAM P(INPUT,OUTPUT);
    17.     VAR  I,PRIME,CHECK,NUMB:REAL,A:ARRAY\1..6' OF REAL?
                                           ^
                                                 ^
                                                      ^
                                                           ^
***Error: ; expected instead of this token
***Error: [ expected instead of this token
***Error: ] expected instead of this token
***Error: ; expected instead of this token
    18. BEGIN
    19. END.
    20.
    21. PROGRAM P(INPUT,OUTPUT);
    22. BEGIN
    23.     FOR I  1 TO 6 DO X:=1
                ^
***Error: := expected after this token
    24. END.
    25.
    26. PROGRAM P(INPUT,OUTPUT);
    27. BEGIN
    28.     CHECK: 1?
```

```
           <------->
***Error: Unexpected input discarded
    29.     BEGIN
    30.        WHILE CHECK)' PRIME DO X:=1
                           <------>
***Error: Unexpected input discarded
    31.     END
    32. END.
    33.
    34. PROGRAM P(INPUT,OUTPUT);
    35.     VAR I:REAL;
    36.     CONST  A[1] 10;A[2] 15;A[3] 25;A[4] 3;?A[5] 50;A[6] 75;
            ^
                     ^
                             ^
                                     ^
                                             ^
                                                 ^
                                                     ^
                                                                 ^

***Error: BEGIN expected instead of this token
***Error: := expected after this token
***Error: := expected after this token
***Error: := expected after this token
***Error: := expected after this token
***Error: Unexpected symbol ignored
***Error: := expected after this token
***Error: := expected after this token
    37. BEGIN
    38. END.
          ^
***Error: "END" inserted to complete phrase started at line 36, column 5

    39.
    40. PROGRAM P(INPUT,OUTPUT);
    41.     BEGIN
    42.        IF X<=0 THEN FACT := 1 ENSE FACT := X*FACT(X-1)
                                          ^
***Error: ELSE expected instead of this token
    43.     END;
            ^
***Error: Unexpected symbol ignored
    44. BEGIN
    45. END.
          ^
***Error: "END" inserted to complete phrase started at line 41, column 5

    46.
    47. PROGRAM REID(INPUT,OUTPUT);
    48.     CONST A[1]=10;A[2]=15;A[3]=25;A[4]==35;A[5]=50;A[6]=75;
            ^
```

```
                    ^
                            ^
                                    ^
                                                    ^
                                                            ^
            <----------------------------------------------------->
***Error: BEGIN expected instead of this token
***Error: := expected instead of this token
***Error: := expected instead of this token
***Error: := expected instead of this token
***Error: := expected instead of this token
***Error: := expected instead of this token
***Error: Misplaced construct(s)
    49.     VAR Q:INTEGER;
    50. BEGIN
    51. END.
    52.
    53. PROGRAM P(INPUT,OUTPUT);
    54.     PROCEDURE PR;
    55.     BEGIN
    56.        X:=1;
    57.        BEGIN
    58.            X:=1
    59.        END ;
        ----------->
***Error: Misplaced construct(s) (from line 54, column 5 to ...)

    60.     VAR  FACT, FACT2: REAL ;
    61. BEGIN
    62. END.
    63.
    64. PROGRAM P(INPUT,OUTPUT);
    65. BEGIN;
        <--->
***Error: procedure_or_function_declaration_list expected instead
    66.     PROCEDURE FACTR(N: INTEGER  ; VAR FACTOR : INTEGER ) ;
    67.     BEGIN
    68.         X:=1
    69.     END ;
                ^
***Error: BEGIN expected after this token
    70.     X:=1
    71. END.
```

Ada

```
     1. procedure ATESTS IS
     2.     x: float := 2.1 +;
                            ^
***Error: term expected after this token
     3.     a: array INTEGER range 1..10 of integer;
                  ^
                                        ^
***Error: ( expected after this token
***Error: ) expected after this token
     4.     B: ARRAY [INTEGER RANGE 0..9] OF FLOAT;
                  ^
                                         ^
***Error: ( expected instead of this token
***Error: ) expected instead of this token
     5.     C : ARRAY (BOOLEAN);
                 <------------->
***Error: [CONSTANT]constrained_array_definition expected instead
     6.     type t is
     7.         RECORD
     8.             A B: CHARACTER;;
                      ^
                                    ^
***Error: Unexpected symbol ignored
***Error: Unexpected symbol ignored
     9.         END RECORD;
    10.     type b is INTEGER range 1..30;
                  ^
***Error: Unexpected symbol ignored
    11.     subtype c is range 1..30;
                       ^
***Error: type_mark expected after this token
    12.
    13.     procedure count is
    14.         use TEXT_IO;
    15.         x: integer;
                          ^
***Error: BEGIN expected after this token
    16.         GET(x);
    17.         PUT(x);         - - a bad comment
                                <--------------->
***Error: Unexpected input discarded
    18.     end count;
    19.
    20.     procedure q is seperate;
                           ^
***Error: SEPARATE expected instead of this token
    21.
    22.     procedure spell is
    23.       B: ARRAY OF FLOAT;
```

```
                    ^
***Error: index_constraint expected after this token
    24.       x: integer;
    25.       FOR I IN 1 .. 10 LOOP
              ^
***Error: BEGIN inserted before this token
    26.           b(i) := 0.0;
                          ^
***Error: "END LOOP ;" inserted to complete phrase started at line 25, column 7

    27.     end;
    28.
    29.     function DAYS_IN_MONTH(M: MONTH IS_LEAP: BOOLEAN) RETURN DAY IS
                                              ^
***Error: ; expected after this token
    30.     begin
    31.         case M of
                     ^
***Error: ( expected instead of this token
    32.                   FEB => RETURN 28;
        ---------------------->
                                           ^
***Error: Misplaced construct(s) (from line 31, column 9 to ...)

***Error: EXCEPTION expected after this token
    33.             when APR => 30;
                                ^
***Error: RETURN expected after this token
    34.             when SEP | APR | JUN | | NOV => RETURN 30;
                                             ^
***Error: Unexpected symbol ignored
    35.             when others => return 31;
    36.         end case;
                <------->
***Error: Unexpected input discarded
    37.         Z(Y - 5*J + K REM 7) > 0 LOOP
                <---------------------->
***Error: Unexpected input discarded
    38.             x := x + 1;
    39.             go to label;
                    <--->
***Error: Symbols merged to form GOTO
    40.         END LOOP;
    41.         x := x + + 1;
                         ^
***Error: Unexpected symbol ignored
    42.         y := (( 3 ;
                         ^
                         ^
***Error: ")" inserted to complete phrase
***Error: ")" inserted to complete phrase
```

```
    43.         return 28;
    44.         <<label
                      ^
***Error: >> expected after this token
    45.         return 29;
    46.     end of DAYS_IN_MONTH;
                ^
***Error: Unexpected symbol ignored
    47.
    48.     PROCEDURE P IS
    49.         x: integer := 2
                              ^
***Error: ; expected after this token
    50.     begin
    51.         loop
    52.             if x > 0 then y := 2;
    53.             if y < 0 then z := 3;
                                       ^
                                       ^
***Error: "END IF ;" inserted to complete phrase
***Error: "END IF ;" inserted to complete phrase started at line 52, column 13

    54.         end loop;
    55.     end p;
    56.
    57.     procedure test is
    58.         x: array(123.144) of real;
                         <-----^
***Error: ".. simple_expression" inserted to complete phrase
    59.         Y: INTEGER = 5;
                           ^
***Error: := expected instead of this token
    60.     begin
    61.         if x(1) := y then
                       ^
***Error: Invalid relational_operator
    62.             NULL;
    63.         ELSEIF X(2) > Y THEN
                ^
***Error: ELSIF expected instead of this token
    64.             NULL;
    65.         end if;
    66.     end atests;
                    ^
***Error: "BEGIN sequence_of_statements END ;" inserted to complete
          phrase started at line 1, column 1
```

Appendix F

Pascal2 Grammar

This grammar is an LALR(2) grammar for Pascal derived from the grammar in Appendix D of [11]. It is specified in standard BNF format with adjacent symbols separated by at least one blank. The "`%empty`" symbol denotes the empty string. All reserved words are written in upper case letters. The lexical categories integer_literal, string_literal, etc, are viewed as terminals. The "`--`" symbol is used to indicate that the rest of the line is a comment. Comments are included wherever there is a deviation from the syntax given in Appendix D.

The information associated with this grammar is broken down into four sections. The first section is a "Terminals" section in which the terminals of the grammar are listed. The second section is a "Rules" section where the rules of the grammar are specified. The third section is a "Names" section where symbols of the grammar are mapped into a different name to be used in issuing error diagnosis. The fourth and final section is a "Scopes" section where the scopes that were computed automatically from the grammar are given. Each scope is written in the form of a BNF rule with a "." in its right-hand side to indicate where the prefix of the scope ends and the suffix begins. Note that a scope does not necessarily correspond to a rule of the grammar. In particular, if the suffix of a scoped rule contains nullable nonterminals in its suffix, these nullable symbols are removed from the scope suffix.

Terminals

```
PROGRAM BEGIN END FUNCTION PROCEDURE VAR TYPE CONST LABEL GOTO WITH
DO FOR TO DOWNTO REPEAT UNTIL WHILE CASE OF IF THEN ELSE RECORD SET
FILE_tok ARRAY PACKED OR IN DIV MOD AND NOT NIL DIRECTIVE
IDENTIFIER      STRING_LITERAL     INTEGER_LITERAL     REAL_LITERAL
+  -  *  /  =  <>  <  <=  >  >=  .  ..  ,  :  ;  :=  (  )  [  ]  ^
```

Rules

```
program_list ::= %empty  |  program_list program
program ::= program_heading block .
program_heading ::= PROGRAM IDENTIFIER ( file_identifier_list ) ;
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 ::= %empty  |  LABEL label_list ;
label ::= integer_literal
constant_definition_part  ::= %empty  |  CONST constant_definition_list ;
constant_definition ::= IDENTIFIER = constant
constant ::= unsigned_number      |  sign unsigned_number
           | constant_identifier  |  sign constant_identifier
           | string_literal
unsigned_number ::= integer_literal  |  real_literal
sign ::= +  |  -
constant_identifier ::= IDENTIFIER
type_definition_part ::= %empty  |  TYPE type_definition_list ;
type_definition ::= IDENTIFIER = type
type  ::= simple_type  |  structured_type  |  pointer_type
simple_type ::= scalar_type  |  subrange_type  |  type_identifier
scalar_type ::= ( identifier_list )
subrange_type ::= constant .. constant
type_identifier ::= IDENTIFIER
structured_type ::= unpacked_structured_type
                  |  PACKED unpacked_structured_type
unpacked_structured_type ::= array_type  |  record_type
                           | set_type    |  file_type
array_type ::= ARRAY [ index_type_list ] OF component_type
index_type ::= simple_type
component_type ::= type
record_type ::= RECORD field_list END
field_list ::= fixed_part | variant_part  |  fixed_part ; variant_part
fixed_part ::= record_section  |  fixed_part ; record_section
record_section ::= %empty  |  field_identifier_list : type
variant_part ::= CASE tag_field type_identifier OF
                     variant_list
tag_field ::= %empty  |  field_identifier :
field_identifier ::= IDENTIFIER
variant ::= %empty  |  case_label_list : ( field_list )
case_label ::= constant
set_type ::= SET OF base_type
base_type ::= simple_type
file_type ::= FILE OF type
pointer_type ::= ^ type_identifier
variable_declaration_part ::= %empty  |  VAR variable_declaration_list ;
variable_declaration ::= identifier_list : type
procedure_and_function_declaration_part ::= %empty
             | procedure_or_function_declaration_list ;
procedure_or_function_declaration ::= procedure_declaration
                                    | function_declaration
procedure_declaration ::= procedure_heading block
                        | procedure_heading DIRECTIVE
procedure_heading ::= PROCEDURE IDENTIFIER ;
                    | PROCEDURE IDENTIFIER
```

```
                          ( formal_parameter_section_list ) ;
    formal_parameter_section ::= parameter_group  |  VAR parameter_group
                           | FUNCTION parameter_group
                           | PROCEDURE identifier_list
    parameter_group ::= identifier_list : type_identifier
    function_declaration ::= function_heading block
                         |  function_heading DIRECTIVE
    function_heading ::= FUNCTION IDENTIFIER : result_type ;
            | FUNCTION IDENTIFIER
                     ( formal_parameter_section_list ) : result_type ;
    result_type ::= type_identifier
    statement_part ::= compound_statement
    statement ::= unlabelled_statement  |  label : unlabelled_statement
    unlabelled_statement ::= simple_statement  |  structured_statement
    simple_statement ::= assignment_statement  |  procedure_statement
                     | goto_statement        | empty_statement
    assignment_statement ::= variable := expression
--
--  The variable on the left-hand side of the assignment may be a
--  file_identifier, a referenced_variable, a pointer variable,
--  a simple identifier, an indexed variable, a field_designator,
--  a file_buffer or a function identifier.
--
    record_variable ::= variable
    variable ::= IDENTIFIER  | variable [ expression_list ]
              | record_variable . IDENTIFIER
              | variable ^               -- file or pointer variable
    expression ::= simple_expression
                | simple_expression relational_operator simple_expression
    relational_operator ::= =  |  <>  |  <  |  <=  |  >=  |  >  | IN
    simple_expression ::= term  |  sign term
                       | simple_expression adding_operator term
    adding_operator ::= +  |  -  |  OR
    term ::= factor  |  term multiplying_operator factor
    multiplying_operator ::= *  |  /  |  DIV  |  MOD  |  AND
    factor ::= variable       | unsigned_constant
             | ( expression ) | function_designator
             | set            | NOT factor
    unsigned_constant ::= unsigned_number  |  string_literal  |  NIL
    function_designator ::= function_identifier ( actual_parameter_list )
    function_identifier ::= IDENTIFIER
    set ::= [ ]  |  [ element_list ]
    element ::= expression  |  expression .. expression
    procedure_statement ::= procedure_identifier
                         | procedure_identifier ( actual_parameter_list )
    procedure_identifier ::= IDENTIFIER
    actual_parameter ::= expression
--
--  An actual parameter may be a simple variable, a procedure
--  identifier or a function identifier.  These classes are
--  included under expression.
```

```
--
    goto_statement ::= GOTO label
    empty_statement ::= %empty
    structured_statement ::= compound_statement    |  conditional_statement
                           | repetitive_statement  |  with_statement
    restricted_statement ::= simple_statement
                           | compound_statement
                           | IF expression THEN
                                 restricted_statement [;]
                             ELSE restricted_statement
                           | case_statement
                           | restricted_while_statement
                           | repeat_statement
                           | restricted_for_statement
                           | restricted_with_statement
    compound_statement ::= BEGIN statement_list END
    conditional_statement ::= if_statement  |  case_statement
    if_statement ::= IF expression THEN
                         statement
                   | IF expression THEN
                         restricted_statement [;]
                     ELSE statement
--
-- Note that the use of [;] in the specification of an if_statement
-- is an extension that allows the parser to accept such a statement
-- with a ";" preceeding the ``ELSE''.  With this arrangement, a
-- semantic warning message is emitted when the ";" is present.
--
    case_statement ::= CASE expression OF
                           case_list_element_list
                       END
    case_list_element ::= %empty  |  case_label_list : statement
    repetitive_statement ::= while_statement
                           |  repeat_statement
                           |  for_statement
    while_statement ::= WHILE expression DO
                            statement
    restricted_while_statement ::= WHILE expression DO
                                       restricted_statement
    repeat_statement ::= REPEAT
                             statement_list
                         UNTIL expression
    for_statement ::= FOR control_variable := for_list DO
                          statement
    restricted_for_statement ::= FOR control_variable := for_list DO
                                     restricted_statement
    for_list ::= initial_value TO final_value
               | initial_value DOWNTO final_value
    control_variable ::= IDENTIFIER
    initial_value ::= expression
    final_value ::= expression
```

```
    with_statement ::= WITH record_variable_list DO
                           statement
    restricted_with_statement ::= WITH record_variable_list DO
                                      restricted_statement
--
-- Expansion of lists
--
index_type_list ::= index_type  |  index_type_list , index_type
expression_list ::= expression  |  expression_list , expression
identifier_list ::= identifier  |  identifier_list , identifier
field_identifier_list ::= field_identifier
                        | field_identifier_list , field_identifier
file_identifier_list ::= file_identifier
                       | file_identifier_list , file_identifier
label_list ::= label  |  label_list , label
constant_definition_list ::= constant_definition
                           | constant_definition_list ; constant_definition
type_definition_list ::= type_definition
                       | type_definition_list ; type_definition
procedure_or_function_declaration_list ::= procedure_or_function_declaration
                                       | procedure_or_function_declaration_list ;
                                         procedure_or_function_declaration
variant_list ::= variant  |  variant_list ; variant
case_label_list ::= case_label  |  case_label_list , case_label
variable_declaration_list ::= variable_declaration
                            | variable_declaration_list ; variable_declaration
formal_parameter_section_list ::= formal_parameter_section
               | formal_parameter_section_list ; formal_parameter_section
actual_parameter_list ::= actual_parameter
                        | actual_parameter_list , actual_parameter
element_list ::= element  |  element_list , element
statement_list ::= statement  |  statement_list ; statement
case_list_element_list ::= case_list_element
                         | case_list_element_list ; case_list_element
record_variable_list ::= record_variable
                       | record_variable_list , record_variable
--
-- Expand the optional semicolon nonterminal that may appear before ELSE
--
[;] ::= %empty
      | ;          -- Issue a warning when this rule is reduced
```

Names

```
    EOFT_SYMBOL -> 'End of Pascal source'
```

Scopes

```
    if_statement ::= IF expression THEN restricted_statement [;]  .ELSE
    restricted_statement ::= IF expression THEN restricted_statement [;]  .ELSE
    block ::= label_declaration_part constant_definition_part
```

```
          type_definition_part variable_declaration_part
          procedure_and_function_declaration_part  .statement_part
case_statement ::= CASE expression OF case_list_element_list  .END
variant ::= case_label_list COLON ( field_list  .)
function_designator ::= function_identifier ( actual_parameter_list  .)
variable ::= variable [ expression_list  .]
repeat_statement ::= REPEAT statement_list  .UNTIL expression
compound_statement ::= BEGIN statement_list  .END
set ::= [ element_list  .]
factor ::= ( expression  .)
record_type ::= RECORD field_list  .END
procedure_and_function_declaration_part ::=
                      procedure_or_function_declaration_list  .;
```

Bibliography

[1] D. E. Knuth
On the Translation of Languages from Left to Right
Information and Control 8:6, 607-639, 1965

[2] D. R. Fulkerson and D. A. Gross.
Incidence matrices and interval graphs.
Pacific J. Math., 15 (1965), no. 3.

[3] F. L. DeRemer
Practical Translators for LR(k) Languages.
Ph.D. dissertation, MIT, Cambridge, Mass., 1969

[4] F. L. DeRemer
Simple LR(k) Grammars.
Comm. ACM 14, 7, 453-460 July 1971

[5] D. E. Knuth
Top-down syntax analysis.
Acta Informatica 1, 79-110 (1971)

[6] W. R. Lalonde
An Efficient LALR Parser Generator.
Tech. Rep. 2, Computer Systems Research Group, Univ. Toronto, 1971

[7] R. E. Tarjan
Depth-First Search and Linear Graph Algorithms.
SIAM J. Computing 1, 146-160 (1972)

[8] R. M. Karp
Reducibility among combinatorial problems
R. E. Miller and J. W. Thatcher (eds.), Complexity of Computer Computations,
Plenum Press, New York, 85-103. 1972

[9] Alfred V. Aho, Jeffrey D. Ullman
The Theory of Parsing, Translation, and Compiling Volume I & II
Prentice Hall, Inc 1972

[10] S. C. Johnson
YACC - Yet Another Compiler-Compiler.
Tech. Rep. CSTR 32, Bell Labs., Murray Hill, N.J., 1974.

[11] Kathleen Jensen, Niklaus Wirth
Pascal User Manual And Report, Second Edition
Springer-Verlag, 1974

[12] Marc L. Joliat A Simple Technique for Partial Elimination of Unit Productions from LR(k) Parsers.
IEEE Transactions on Computers, 763-764, July 1976

[13] D. Pager
A Practical General Method for Constructing LR(k) Parsers.
Acta Inf. 7, 3 (1977), 249-268.

[14] J. Eve, R. Kurki-Suonio
On Computing the Transitive Closure of a Relation.
Acta Inf. 8 (1977), 303-314.

[15] S. F. Ziegler. Smaller faster table driven parser.
Unpublished manuscript, Madison Academic Computing Center,
University of Wisconsin, Madison, Wisconsin, 1977.

[16] S. Even, D. I. Lichtenstein, Y. Shiloach
Remarks on Ziegler's method for matrix compression.
Unpublished manuscript, 1977

[17] Penello T.J. and DeRemer, F.A.
A Forward Move for LR Error Recovery
Conf. Record ACM Symposium on Principles of Programming Languages,
January, 1978

[18] Mickunas M.D., and Modry, J. A.
Automatic Error Recovery for LR parsers
Comm. ACM 21, 6
(June 1978), 459-465

[19] Ripley, G. D., and Druseikis, F.C.
A statistical analysis of syntax errors
Journal of Computer Languages 3,4 (1978) (227-240)

[20] D. J. Ripley
Pascal Syntax Errors Data Base
RCA Laboratories, Princeton, N.J., Apr 1979

[21] S. L. Graham, C. B. Haley, W. N. Joy
Practical LR Error Recovery
Proceedings of the SIGPLAN 79 Symposium on Compiler Construction
(August 6-10, 1979, Denver) ACM, New York, 1979, pp 168-175.

[22] Michael R. Garey, David S. Johnson.
Computers and Intractability.
W. H. Freeman and Company, 1979

[23] Robert E. Tarjan, Andrew C. Yao.
Storing a sparse Table.
Communications of the ACM 22 606-611, Nov.1979

[24] Bent Bruun Kristensen, Ole Lehrmann Madsen
Methods for Computing LALR(k) Lookahead
ACM Transactions on Programming Languages and Systems,
Vol. 3, No. 1, January 1981, Pages 60-82.

[25] Philippe Charles
Implementation of an LALR Parser Generator
M.S. thesis, NYU, June 1982

[26] Frank DeRemer, Thomas Penello
Efficient Computation of LALR(1) Look-Ahead Sets
ACM Transactions on Programming Languages and Systems,
Vol. 4, No. 4, October 1982, Pages 615-649.

[27] Seppo Sippu, Eljas Soisalon-Soininen
A Syntax-Error-Handling Technique and its Experimental Analysis
ACM Transactions on Programming Languages and Systems, Vol. 5, No. 4,
October 1983, Pages 656-679

[28] Ref. Manual for the ADA Programming Language
ANSI/Mil-STD-1815A-1983, U.S. Dept. of Defense.

[29] Philippe G. Charles, Gerald A. Fisher.
An LALR(1) grammar for ANSI Ada.
Ada Letters vol. 3, No. 4 37-50 (Jan-Feb. 1984).

[30] Peter Dencker, Karl Durre, and Johannes Heuft.
Optimization of Parser Tables for Portable Compilers.
Acm Transactions on Programming Languages and Systems
6,4 (Oct. 1984), 546-572.

[31] Philippe G. Charles, Gerald A. Fisher.
An LALR(1) grammar for ANSI Ada.
Ada Letters vol. 3, No. 4 37-50 (Jan-Feb. 1984).

[32] Joseph C. H. Park, K. M. Choe, C. H. Chang
A New Analysis of LALR Formalisms
ACM Transactions on Programming Languages and Systems,
Vol. 7, No. 1, January 1985, Pages 159-175.

[33] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman.
Compilers, Principles, Techniques, and Tools
Addison Wesley, 1986

[34] J. T. Schwartz, R. B. B. Dewar, E. Dubinsky, E. Schonberg.
Programming With Sets: An Introduction to SETL
Springer-Verlag, 1986

[35] Thomas J. Sager
A Short Proof of a Conjecture of DeRemer and Pennello
ACM Transactions on Programming Languages and Systems, Vol. 8, No. 2, April 1986, Pages 264-271

[36] Fred Ives
Unifying View of Recent LALR(1) Lookahead Set Algorithms
SIGPLAN Notices 21, 7 (July 1986), 131-135

[37] Michael Burke, Gerald A. Fisher
A Practical Method for LR and LL Syntactic Error Diagnosis and Recovery
ACM Transactions on Programming Languages and Systems, Vol. 9, No. 2,
April 1987, Pages 164-197

[38] C. Fisher, R. J. LeBlanc
Crafting a Compiler
Benjamin/Cummings, Menlo Park, 1988

[39] Manuel E. Bermudez, George Logothetis
Simple Computation of LALR(1) Lookaheaed Sets
Information Processing Letters 31 (1989) 233-238

[40] Philippe Charles, Laurent Pautet
Efficient Representation of LR Parsing Information for Error Recovery
Unpublished paper, 1989

[41] Philippe Charles
Recent Algorithms for constructing LALR parsers
(Unpublished) Survey paper, 1989

[42] Kirk Snyder
The SETL2 Programming Language
Technical Report 490,
Courant Institute of Mathematical Sciences, New York university, 1990

[43] Jüergen Uhl: Private communications

[44] Franco Gasperoni: Private communications