The Parser of Pascal

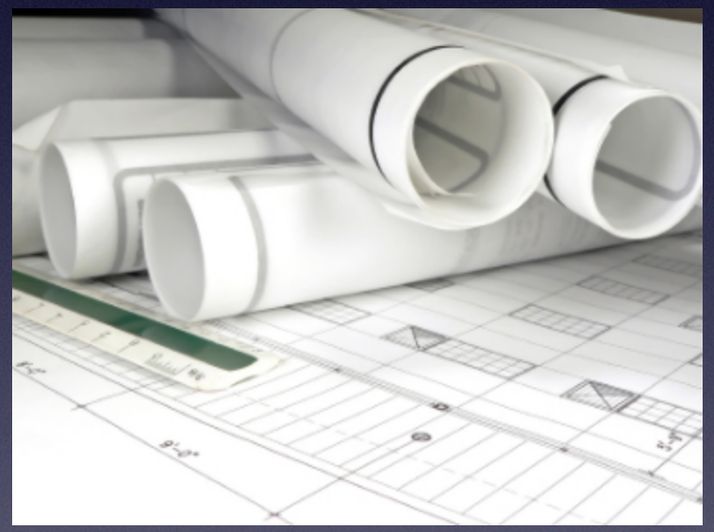
C++ Implementation

Outline

- 1.Introduction
- 2.Journal of testing
- 3.Conclusion

Outline

- **1.Introduction**
- 2.Journal of testing
- 3.Conclusion



Overview

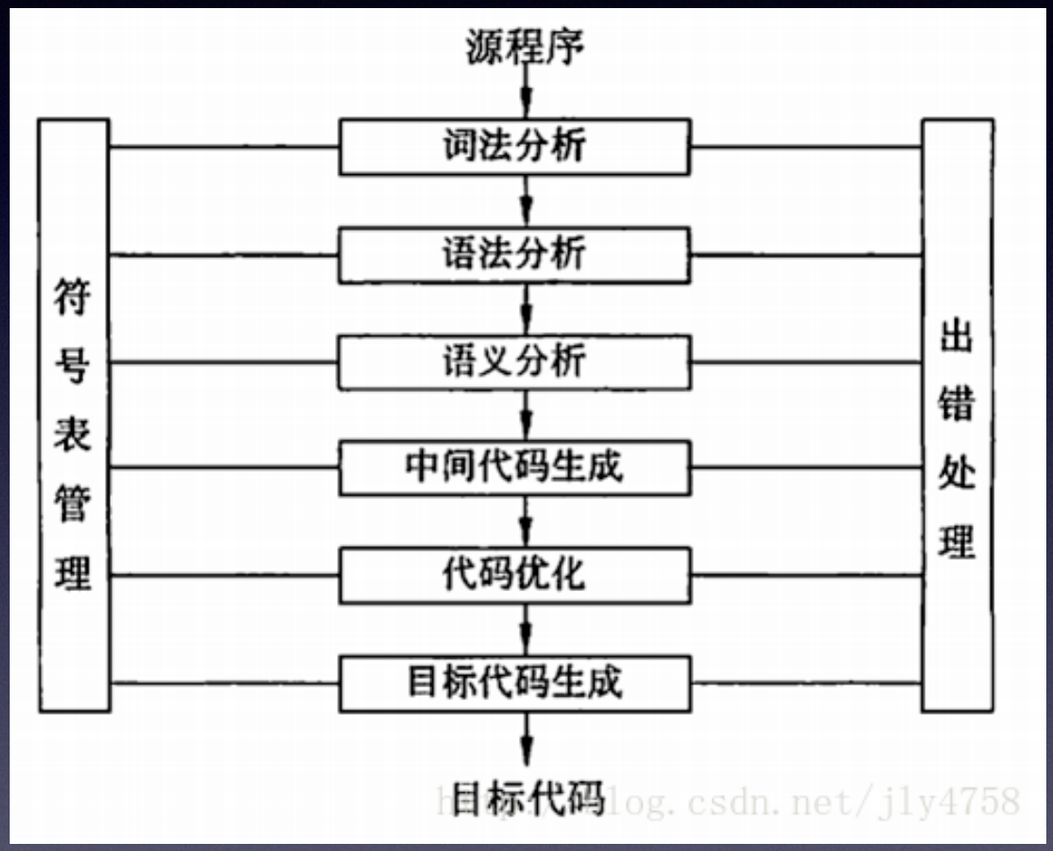


Figure: Processing Pipeline

Overview

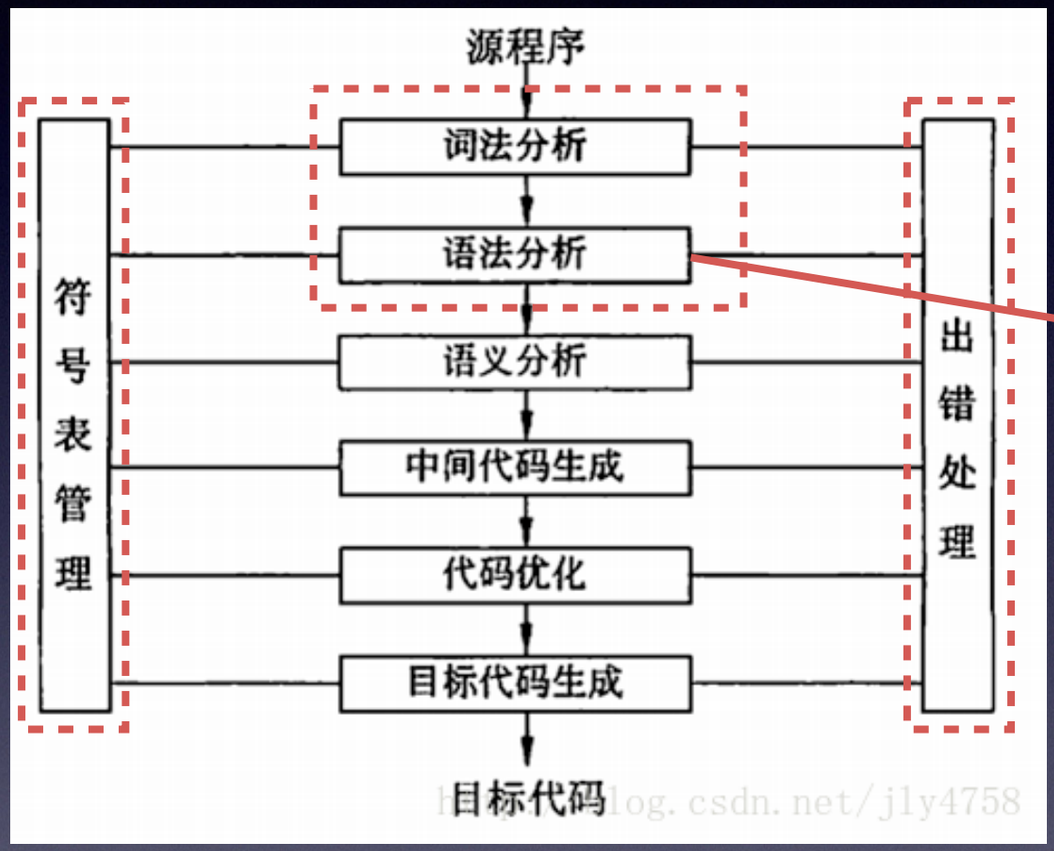


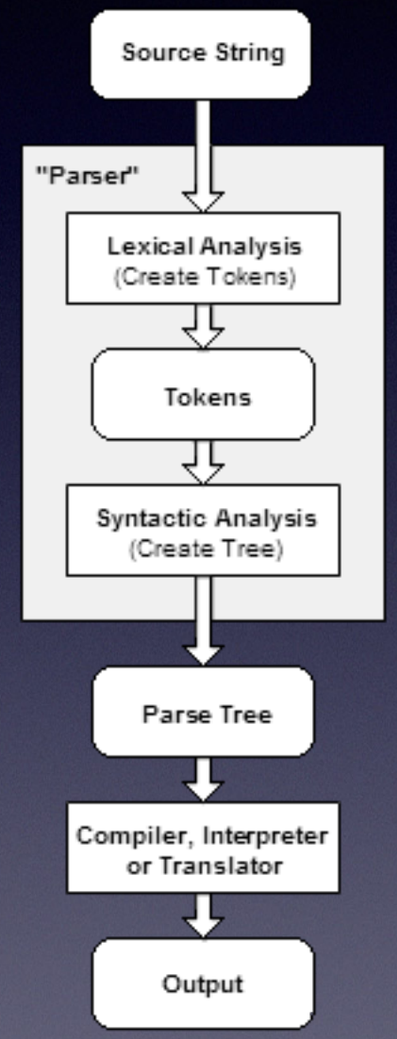
Figure: Processing Pipeline

Introduction

- Writing a C++ parser with respect to SLR(1) mechanism.
- Two approaches to construct SLR(1) parser:
  - 1.Flex+Bison: Mature and Simple
  - 2.Hand-Writing: Learning SLR(1) the “hard” way

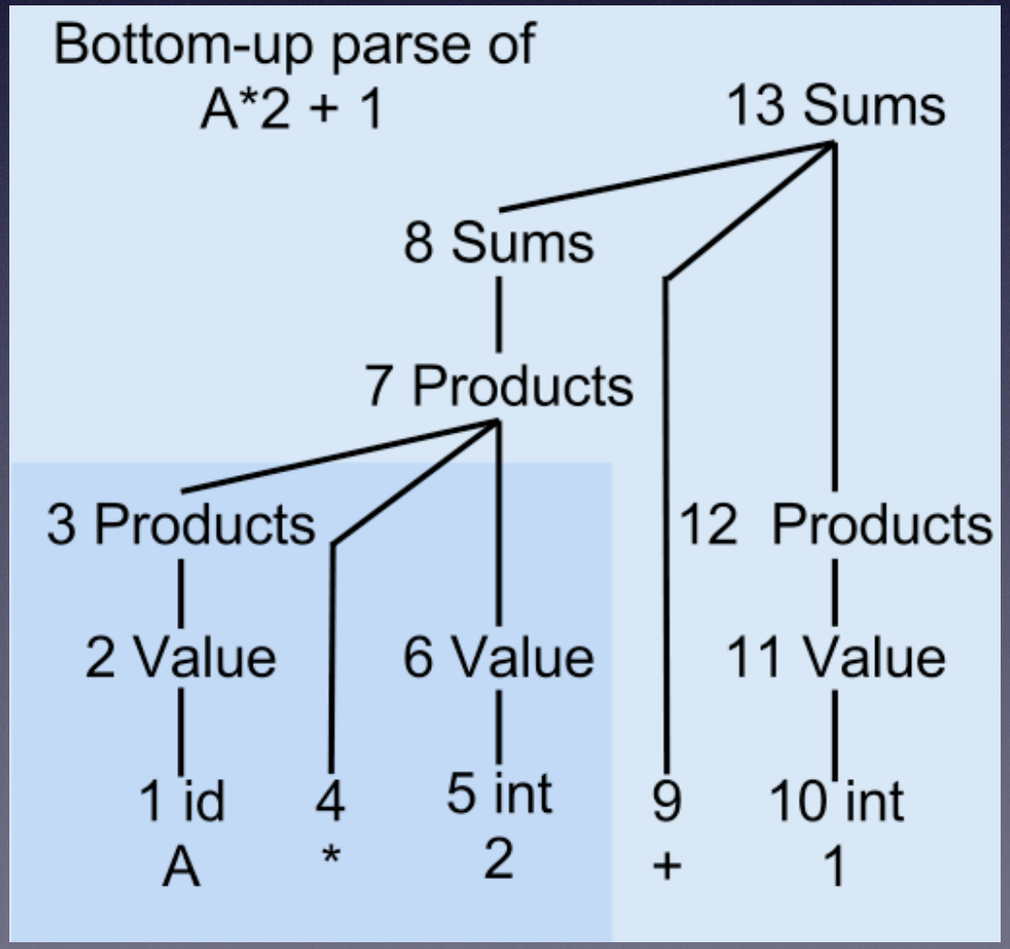
Parser

- Given a grammar and a statement;
- Judging if the statement matches the grammar.



Bottom-Up Parsing

- Scanning and parsing the input text.
- Building up the parse tree bottom up, and left to right.



Shift and Reduce Actions

- **Shift:** advances in the input stream by one symbol
- **Reduce:** applies a completed grammar rule

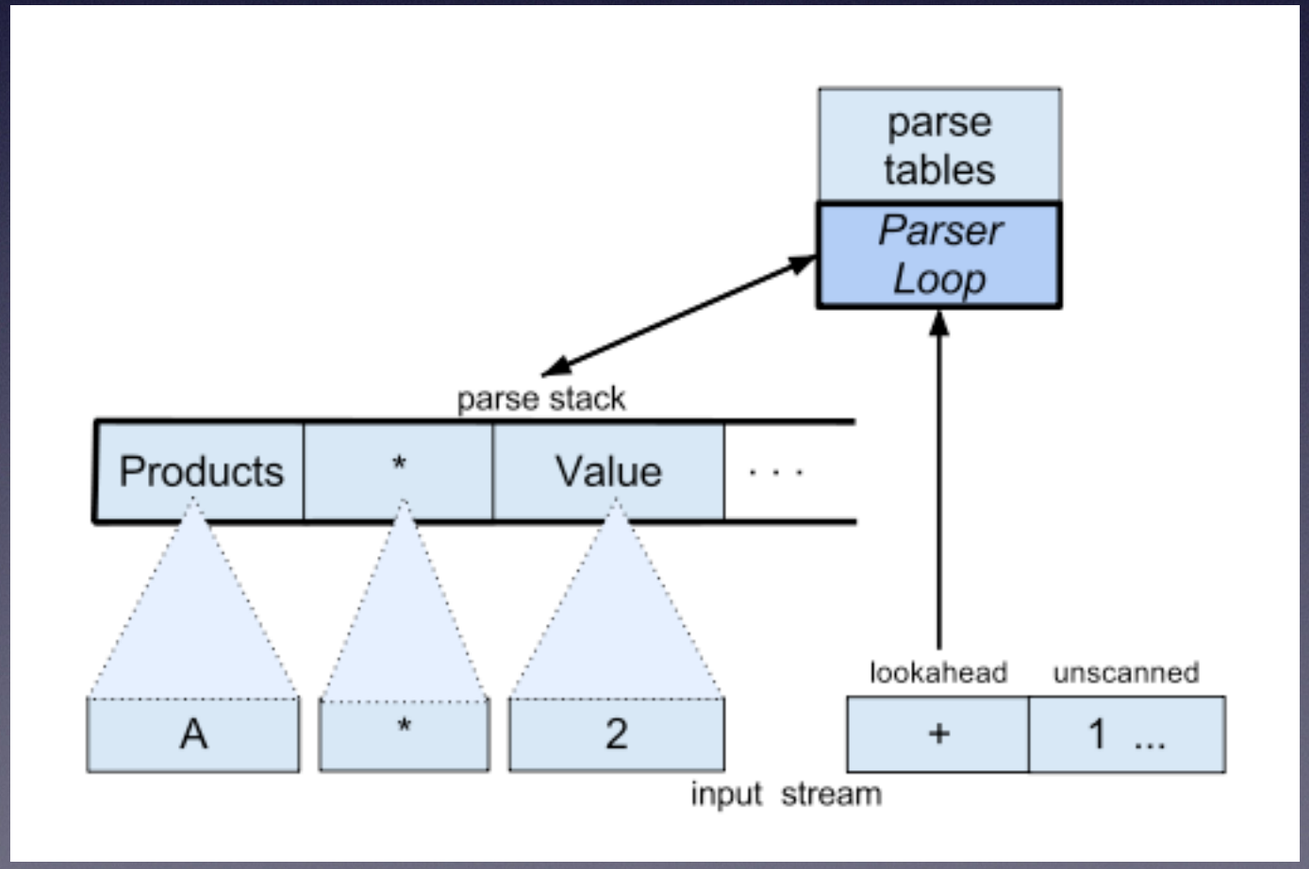


Table-based LR(1)

- Stack: Runtime State, Symbol
- Goto Table: State Transition
- Action Table: Applied Action

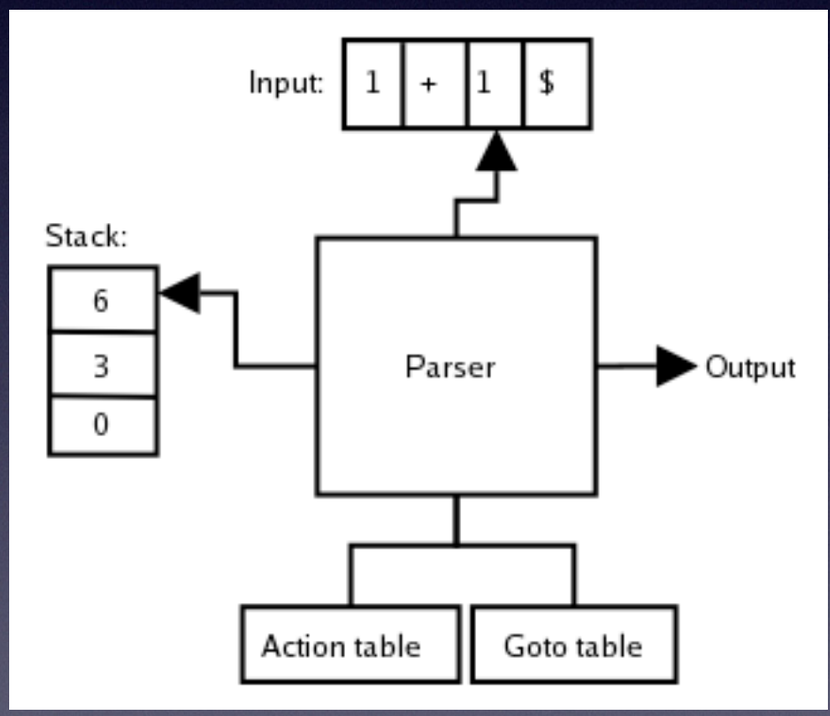
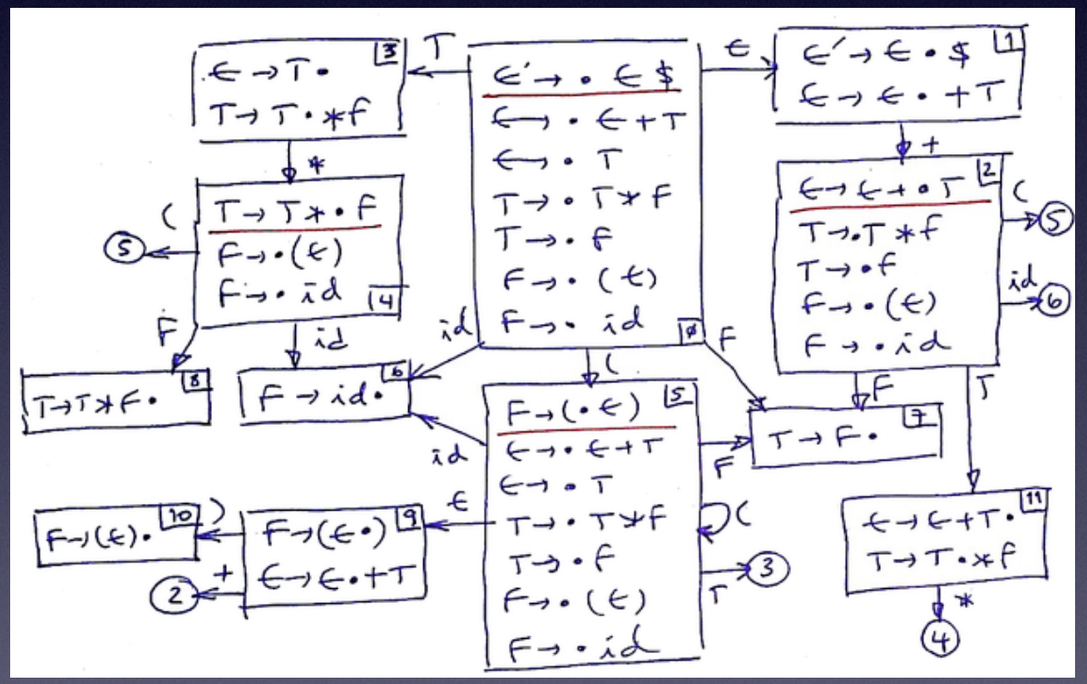


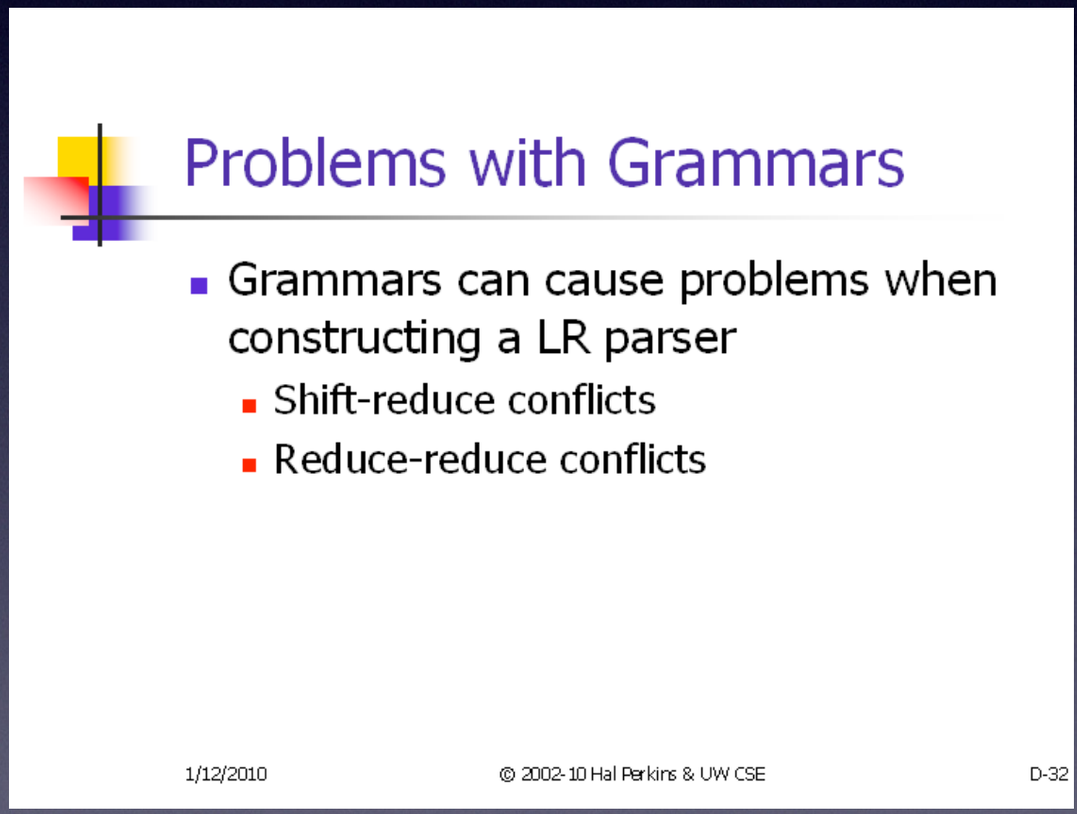
Table Construction

- Items, eg. E → • E + B
- Closure of item sets
- Building the canonical LR(1) collection
- Finite automaton machine



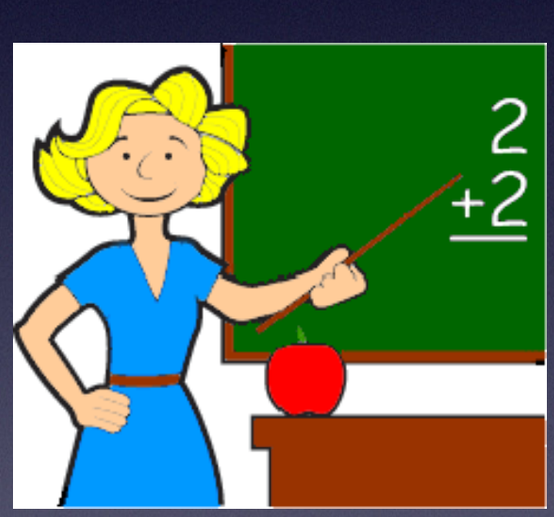
Conflict Resolution

- Manually defining the priority in the item set.
- See our testing example below.



Outline

- 1.Introduction
- **2.Journal of testing**
- 3.Conclusion



Input Grammar

- **S program id; | compound_stmt.**
- **compound_stmt begin stmts end**
- **stmts stmt | stmts; stmt**
- **stmt id := expr | compound_stmt | if_stmt | for_stmt | while bool do stmt | ε**
- **if_stmt if bool then stmt | if bool then stmt else stmt**
- **for_stmt for id := expr to expr do stmt | for id := expr downto expr do stmt**
- **bool expr > expr | expr < expr**
- **expr expr + expr | expr - expr | expr * expr | expr / expr | expr ^ factor | factor**
- **factor id | num | (expr)**

Test Program

```
program test;
    begin
        x:=19;
        for i:=100 downto 15 do
            if x<y+(15-9) then y:=x
                else begin
                while x+y*z>x do begin y:=y+y^5-1 end;
                z:=z*7+x
    end
end.
```

- Generating Token file by using our lexer program.

parser_tests — -bash — 58×24

```
ChenMac:parser_tests wasdns$ cat given_example_tocken.txt
Token(PROGRAM, 'PROGRAM')
Token(ID, 'test')
Token(BEGIN, 'BEGIN')
Token(ID, 'x')
Token(ASSIGN, ':=')
Token(INTEGER_CONST, 19)
Token(SEMI, ';')
Token(ID, 'for')
Token(ID, 'i')
Token(ASSIGN, ':=')
Token(INTEGER_CONST, 100)
Token(ID, 'downto')
Token(INTEGER_CONST, 15)
Token(ID, 'do')
Token(ID, 'if')
Token(ID, 'x')
Token(LANGBRA, '<')
Token(ID, 'y')
Token(PLUS, '+')
Token(LPAREN, '(')
Token(INTEGER_CONST, 15)
Token(MINUS, '-')
Token(INTEGER_CONST, 9)
```

Figure2: Our Lexer Output(Partly)

- Converting the token.txt with the parser-familiar format

build — -bash — 58×24

```
ChenMac:build wasdns$ cat input.txt
program
id
;
begin
id
:=
num
;
for
id
:=
num
downto
num
do
if
id
<
id
+
(
num
-
```

Figure3: Converted Lexer Output(Partly)

- Generating output files in build/ folder, using Shell scripts.

```
ChenMac:pascal-compiler wasdns$ ls
LICENSE                 docs                    run_lexer.py
README                  lexer_tests             run_lexer_ply.py
build                   parser_tests            run_parser_demo.sh
cleanup.sh              ply_frontend            src
ChenMac:pascal-compiler wasdns$ ./run_parser_demo.sh
ChenMac:pascal-compiler wasdns$ cd build/
ChenMac:build wasdns$ ls
action_and_goto.txt     grammar.txt             parser
error.txt               input.txt               slr.txt
first_and_follow.txt    output.txt
ChenMac:build wasdns$
```

- Generating output files in build/ folder, using Shell scripts.

```
ChenMac:pascal-compiler wasdns$ ls
LICENSE                 docs                    run_lexer.py
README                  lexer_tests             run_lexer_ply.py
build                   parser_tests            run_parser_demo.sh
cleanup.sh              ply_frontend            src
ChenMac:pascal-compiler wasdns$ ./run_parser_demo.sh
ChenMac:pascal-compiler wasdns$ cd build/
ChenMac:build wasdns$ ls
action_and_goto.txt     grammar.txt             parser
error.txt               input.txt               slr.txt
first_and_follow.txt    output.txt
ChenMac:build wasdns$
```

What's does this script do?

1.Creating build/ folder;
2.Running Lexer and generating **output files** in build/;
3.**Compiling** parser program;
4.Reading and parsing the grammar and the output files;
5.Generating results in build/ folder.

- Generating output files in build/ folder, using Shell scripts.

```
ChenMac:pascal-compiler wasdns$ ls
LICENSE                 docs                    run_lexer.py
README                  lexer_tests             run_lexer_ply.py
build                   parser_tests            run_parser_demo.sh
cleanup.sh              ply_frontend            src
ChenMac:pascal-compiler wasdns$ ./run_parser_demo.sh
ChenMac:pascal-compiler wasdns$ cd build/
ChenMac:build wasdns$ ls
action_and_goto.txt     grammar.txt             parser
error.txt               input.txt               slr.txt
first_and_follow.txt    output.txt
ChenMac:build wasdns$
```

Results of running parser

1.first_and_follow.txt: Print the first set and follow set

first_and_follow.txt　　使用“文本编辑”打开

```
====FIRST====

FIRST(S') = { program }
FIRST(S) = { program }
FIRST(compound_stmt) = { begin }
FIRST(stmts) = { ; ,  begin ,  for ,  id ,  if ,  while ,  ε }
FIRST(stmt) = { begin ,  for ,  id ,  if ,  while ,  ε }
FIRST(if_stmt) = { if }
FIRST(for_stmt) = { for }
FIRST(bool) = { ( ,  id ,  num }
FIRST(expr) = { ( ,  id ,  num }
FIRST(factor) = { ( ,  id ,  num }
FIRST(id) = { id }
FIRST(;) = { ; }
FIRST(.) = { . }
FIRST(begin) = { begin }
FIRST(end) = { end }
FIRST(:=) = { := }
FIRST(while) = { while }
FIRST(do) = { do }
FIRST(if) = { if }
FIRST(then) = { then }
FIRST(else) = { else }
FIRST(to) = { to }
FIRST(downto) = { downto }
FIRST(>) = { > }
FIRST(<) = { < }
FIRST(+) = { + }
FIRST(-) = { - }
FIRST(*) = { * }
FIRST(/) = { / }
FIRST(^) = { ^ }
FIRST(num) = { num }
FIRST(() = { ( }
FIRST()) = { ) }
FIRST(program) = { program }
FIRST(for) = { for }
FIRST($) = { $ }

==============

====FOLLOW====

FOLLOW(S) = { $ }
FOLLOW(compound_stmt) = { . ,  ; ,  else ,  end }
FOLLOW(stmts) = { ; ,  end }
FOLLOW(stmt) = { ; ,  else ,  end }
FOLLOW(if_stmt) = { ; ,  else ,  end }
FOLLOW(for_stmt) = { ; ,  else ,  end }
FOLLOW(bool) = { do ,  then }
FOLLOW(expr) = { ) ,  * ,  + ,  - ,  / ,  ; ,  < ,  > ,  ^ ,  do ,  downto ,  else ,  end ,  then ,  to }
FOLLOW(factor) = { ) ,  * ,  + ,  - ,  / ,  ; ,  < ,  > ,  ^ ,  do ,  downto ,  else ,  end ,  then ,  to }

==============
```

Figure: First and Follow

first_and_follow.txt　　使用“文本编辑”打开

```
====FIRST====

FIRST(S') = { program }
FIRST(S) = { program }
FIRST(compound_stmt) = { begin }
FIRST(stmts) = { ; ,  begin ,  for ,  id ,  if ,  while ,  ε }
FIRST(stmt) = { begin ,  for ,  id ,  if ,  while ,  ε }
FIRST(if_stmt) = { if }
FIRST(for_stmt) = { for }
FIRST(bool) = { ( ,  id ,  num }
FIRST(expr) = { ( ,  id ,  num }
FIRST(factor) = { ( ,  id ,  num }
FIRST(id) = { id }
FIRST(;) = { ; }
FIRST(.) = { . }
FIRST(begin) = { begin }
FIRST(end) = { end }
FIRST(:=) = { := }
FIRST(while) = { while }
FIRST(do) = { do }
FIRST(if) = { if }
FIRST(then) = { then }
FIRST(else) = { else }
FIRST(to) = { to }
FIRST(downto) = { downto }
FIRST(>) = { > }
FIRST(<) = { < }
FIRST(+) = { + }
FIRST(-) = { - }
FIRST(*) = { * }
FIRST(/) = { / }
FIRST(^) = { ^ }
FIRST(num) = { num }
FIRST(() = { ( }
FIRST()) = { ) }
FIRST(program) = { program }
FIRST(for) = { for }
FIRST($) = { $ }

==============

====FOLLOW====

FOLLOW(S) = { $ }
FOLLOW(compound_stmt) = { . ,  ; ,  else ,  end }
FOLLOW(stmts) = { ; ,  end }
FOLLOW(stmt) = { ; ,  else ,  end }
FOLLOW(if_stmt) = { ; ,  else ,  end }
FOLLOW(for_stmt) = { ; ,  else ,  end }
FOLLOW(bool) = { do ,  then }
FOLLOW(expr) = { ) ,  * ,  + ,  - ,  / ,  ; ,  < ,  > ,  ^ ,  do ,  downto ,  else ,  end ,  then ,  to }
FOLLOW(factor) = { ) ,  * ,  + ,  - ,  / ,  ; ,  < ,  > ,  ^ ,  do ,  downto ,  else ,  end ,  then ,  to }

==============
```

First

Figure: First and Follow

first_and_follow.txt　　使用“文本编辑”打开

```
====FIRST====

FIRST(S') = { program }
FIRST(S) = { program }
FIRST(compound_stmt) = { begin }
FIRST(stmts) = { ; ,  begin ,  for ,  id ,  if ,  while ,  ε }
FIRST(stmt) = { begin ,  for ,  id ,  if ,  while ,  ε }
FIRST(if_stmt) = { if }
FIRST(for_stmt) = { for }
FIRST(bool) = { ( ,  id ,  num }
FIRST(expr) = { ( ,  id ,  num }
FIRST(factor) = { ( ,  id ,  num }
FIRST(id) = { id }
FIRST(;) = { ; }
FIRST(.) = { . }
FIRST(begin) = { begin }
FIRST(end) = { end }
FIRST(:=) = { := }
FIRST(while) = { while }
FIRST(do) = { do }
FIRST(if) = { if }
FIRST(then) = { then }
FIRST(else) = { else }
FIRST(to) = { to }
FIRST(downto) = { downto }
FIRST(>) = { > }
FIRST(<) = { < }
FIRST(+) = { + }
FIRST(-) = { - }
FIRST(*) = { * }
FIRST(/) = { / }
FIRST(^) = { ^ }
FIRST(num) = { num }
FIRST(() = { ( }
FIRST()) = { ) }
FIRST(program) = { program }
FIRST(for) = { for }
FIRST($) = { $ }

=============

====FOLLOW====

FOLLOW(S) = { $ }
FOLLOW(compound_stmt) = { . ,  ; ,  else ,  end }
FOLLOW(stmts) = { ; ,  end }
FOLLOW(stmt) = { ; ,  else ,  end }
FOLLOW(if_stmt) = { ; ,  else ,  end }
FOLLOW(for_stmt) = { ; ,  else ,  end }
FOLLOW(bool) = { do ,  then }
FOLLOW(expr) = { ) ,  * ,  + ,  - ,  / ,  ; ,  < ,  > ,  ^ ,  do ,  downto ,  else ,  end ,  then ,  to }
FOLLOW(factor) = { ) ,  * ,  + ,  - ,  / ,  ; ,  < ,  > ,  ^ ,  do ,  downto ,  else ,  end ,  then ,  to }

=============
```

Follow

Figure: First and Follow

- Generating output files in build/ folder, using Shell scripts.

```
ChenMac:pascal-compiler wasdns$ ls
LICENSE                 docs                    run_lexer.py
README                  lexer_tests             run_lexer_ply.py
build                   parser_tests            run_parser_demo.sh
cleanup.sh              ply_frontend            src
ChenMac:pascal-compiler wasdns$ ./run_parser_demo.sh
ChenMac:pascal-compiler wasdns$ cd build/
ChenMac:build wasdns$ ls
action_and_goto.txt     grammar.txt             parser
error.txt               input.txt               slr.txt
first_and_follow.txt    output.txt
ChenMac:build wasdns$
```

Results of running parser

1.first_and_follow.txt: Print the first set and follow set
2.action_and_goto.txt: Print the action and goto table

action_and_goto.txt

state	id	;	.	begin	end	:=	while	do	if	then	else	to	downto	>	<	+	-	*	/	^	num	(	)	program	for	$	S	compound_stmt	stmts	stmt	if_stmt	for_stmt	bool	expr	factor
0																								S2			1								
1																										acc									
2	S3																																		
3		S4																																	
4				S6																								5							
5			S7																																
6	S10	r10		S6		r10		S14			S15			r10												S16			11		8	9	12	13			
7																										r1									
8		S18				S17																													
9		r3				r3																													
10							S19																												
11		r6				r6						r6																							
12		r7				r7						r7																							
13		r8				r8						r8																							
14	S23																					S24	S25										20	21	22
15	S23																					S24	S25										26	21	22
16	S27																																		
17		r2	r2			r2						r2																							
18	S10	r10		S6		r10		S14			S15			r10												S16			11		28	12	13			
19	S23																					S24	S25										29	22	
20								S30																											
21														S31	S32	S33	S34	S35	S36	S37															
22		r22				r22			r22			r22	r22	r22	r22	r22	r22	r22	r22	r22	r22			r22											
23		r23				r23			r23			r23	r23	r23	r23	r23	r23	r23	r23	r23	r23			r23											
24		r24				r24			r24			r24	r24	r24	r24	r24	r24	r24	r24	r24	r24			r24											
25	S23																					S24	S25										38	22	
26										S39																									
27							S40																												
28		r4				r4																													
29		r5				r5						r5				S33	S34	S35	S36	S37															
30	S10	r10		S6		r10		S14			S15			r10												S16			11		41	12	13			
31	S23																					S24	S25										42	22	

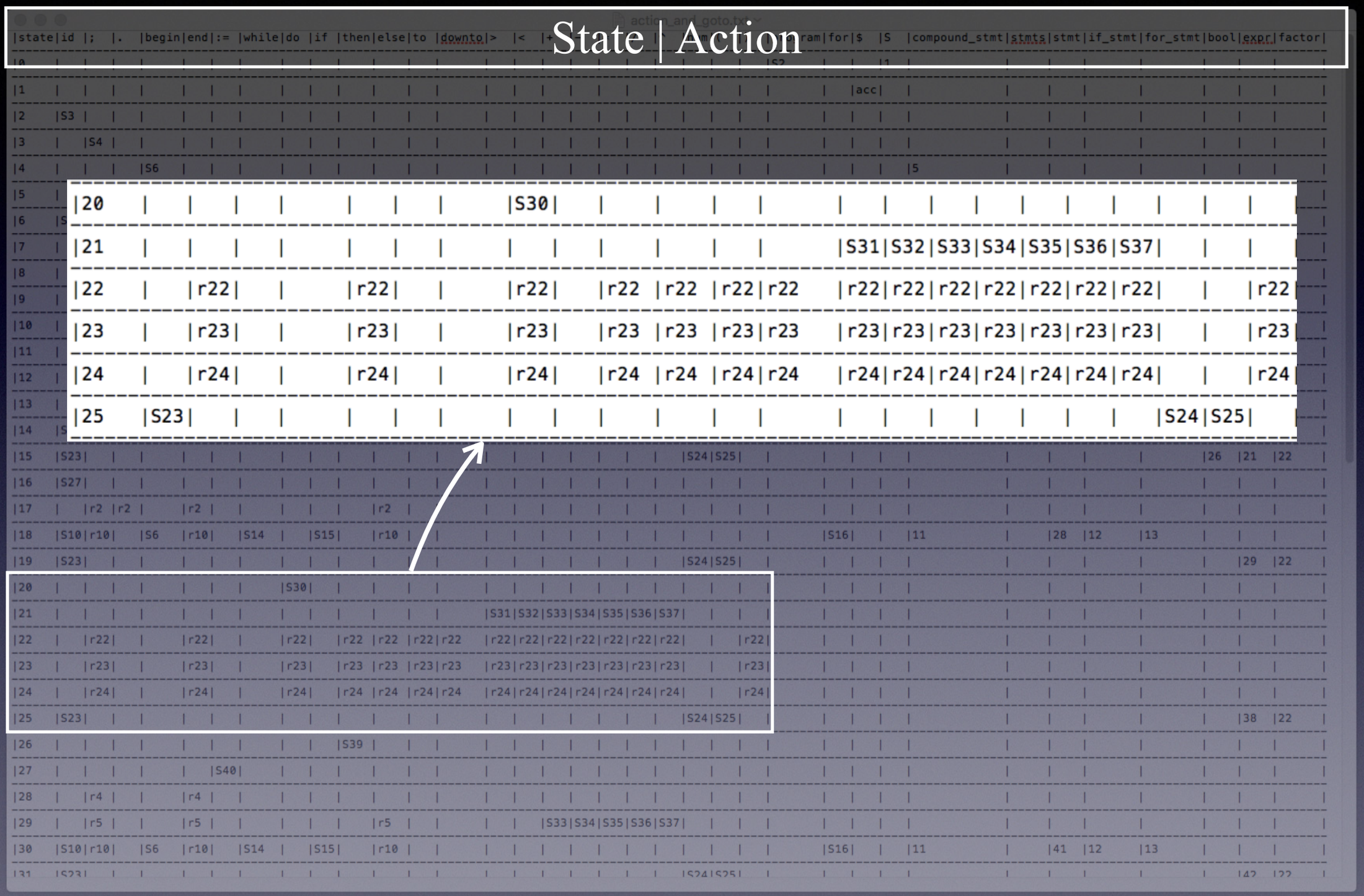
Figure: Action and Goto

State | Action

state	id	;	.	begin	end	:=	while	do	if	then	else	to	downto	>	<	+	-	*	/	=	<>	<=	>=	program	for	$	S	compound_stmt	stmts	stmt	if_stmt	for_stmt	bool	expr	factor
0																										S2		1								
1																											acc									
2	S3																																			
3		S4																																		
4				S6																								5								
5			S7																																	
6	S10	r10		S6		r10		S14		S15		r10															S16			11		8	9	12	13			
7																											r1									
8		S18				S17																														
9		r3				r3																														
10							S19																													
11		r6				r6						r6																								
12		r7				r7						r7																								
13		r8				r8						r8																								
14	S23																			S24	S25													20	21	22
15	S23																			S24	S25													26	21	22
16	S27																																			
17		r2	r2			r2						r2																								
18	S10	r10		S6		r10		S14		S15		r10															S16			11		28	12	13			
19	S23																			S24	S25													29	22	
20								S30																												
21															S31	S32	S33	S34	S35	S36	S37															
22		r22				r22				r22		r22		r22	r22	r22	r22	r22	r22	r22	r22															
23		r23				r23				r23		r23		r23	r23	r23	r23	r23	r23	r23	r23															
24		r24				r24				r24		r24		r24	r24	r24	r24	r24	r24	r24	r24															
25	S23																			S24	S25													38	22	
26										S39																										
27							S40																													
28		r4				r4																														
29		r5				r5						r5				S33	S34	S35	S36	S37																
30	S10	r10		S6		r10		S14		S15		r10															S16			11		41	12	13			
31	S23																			S24	S25													42	22	

Figure: Action and Goto

State | Action



state	id	;	.	begin	end	:=	while	do	if	then	else	to	downto	>	<	+	ram	for	$	S	compound_stmt	stmts	stmt	if_stmt	for_stmt	bool	expr	factor
0																			S2		1							
1																			acc									
2	S3																											
3		S4																										
4				S6																	5							
5																												
6	S																											
7																												
8																												
9																												
10																												
11																												
12																												
13																												
14	S																											
15	S23																		S24	S25							26	21	22
16	S27																											
17		r2	r2		r2					r2																		
18	S10	r10		S6	r10		S14		S15		r10										S16		11			28	12	13
19	S23																		S24	S25							29	22
20								S30																				
21																	S31	S32	S33	S34	S35	S36	S37					
22		r22			r22				r22			r22		r22	r22	r22	r22	r22	r22	r22	r22	r22	r22				r22	
23		r23			r23				r23			r23		r23	r23	r23	r23	r23	r23	r23	r23	r23	r23				r23	
24		r24			r24				r24			r24		r24	r24	r24	r24	r24	r24	r24	r24	r24	r24				r24	
25	S23																		S24	S25							38	22
26										S39																		
27						S40																						
28		r4			r4																							
29		r5			r5						r5					S33	S34	S35	S36	S37								
30	S10	r10		S6	r10		S14		S15		r10										S16		11			41	12	13
31	S23																		S24	S25							42	22

Figure: Action and Goto

- Generating output files in build/ folder, using Shell scripts.

```
ChenMac:pascal-compiler wasdns$ ls
LICENSE                 docs                    run_lexer.py
README                  lexer_tests             run_lexer_ply.py
build                   parser_tests            run_parser_demo.sh
cleanup.sh              ply_frontend            src
ChenMac:pascal-compiler wasdns$ ./run_parser_demo.sh
ChenMac:pascal-compiler wasdns$ cd build/
ChenMac:build wasdns$ ls
action_and_goto.txt     grammar.txt             parser
error.txt               input.txt               slr.txt
first_and_follow.txt    output.txt
ChenMac:build wasdns$
```

Results of running parser

1.first_and_follow.txt: Print the first set and follow set
2.action_and_goto.txt: Print the action and goto table
3.output.txt: Print the runtime stack and applied actions

output.txt

```
(112) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 18 10 19 29 35 22  symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts ; id := expr * factor INPUT: + id end end . $ 根据expr-> factor规约

(113) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 18 10 19 29 35 46  symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts ; id := expr * expr INPUT: + id end end . $ 移入

(114) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 18 10 19 29 35 46 33  symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts ; id := expr * expr + INPUT: id end end . $ 移入

(115) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 18 10 19 29 35 46 33 23  symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts ; id := expr * expr + id INPUT: end end . $ 根据factor-> id规约

(116) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 18 10 19 29 35 46 33 22  symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts ; id := expr * expr + factor INPUT: end end . $ 根据expr-> factor规约

(117) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 18 10 19 29 35 46 33 44  symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts ; id := expr * expr + expr INPUT: end end . $ 根据expr-> expr + expr规约

(118) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 18 10 19 29 35 46  symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts ; id := expr * expr INPUT: end end . $ 根据expr-> expr * expr规约

(119) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 18 10 19 29  symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts ; id := expr INPUT: end end . $ 根据stmt-> id := expr规约

(120) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 18 28  symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts ; stmt INPUT: end end . $ 根据stmts-> stmts ; stmt规约

(121) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8  symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts INPUT: end end . $ 移入

(122) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 17  symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts end INPUT: end . $ 根据compound_stmt-> begin stmts end规约

(123) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 11  symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else compound_stmt INPUT: end . $ 根据stmt-> compound_stmt规约

(124) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 55  symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else stmt INPUT: end . $ 根据if_stmt-> if bool then stmt else stmt规约

(125) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 12  symbolStack: program id ; begin stmts ; for id := expr downto expr do if_stmt INPUT: end . $ 根据stmt-> if_stmt规约

(126) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 61  symbolStack: program id ; begin stmts ; for id := expr downto expr do stmt INPUT: end . $ 根据for_stmt-> for id := expr downto expr do stmt规约

(127) stateStack: 0 2 3 4 6 8 18 13  symbolStack: program id ; begin stmts ; for_stmt INPUT: end . $ 根据stmt-> for_stmt规约

(128) stateStack: 0 2 3 4 6 8 18 28  symbolStack: program id ; begin stmts ; stmt INPUT: end . $ 根据stmts-> stmts ; stmt规约

(129) stateStack: 0 2 3 4 6 8  symbolStack: program id ; begin stmts INPUT: end . $ 移入

(130) stateStack: 0 2 3 4 6 8 17  symbolStack: program id ; begin stmts end INPUT: . $ 根据compound_stmt-> begin stmts end规约

(131) stateStack: 0 2 3 4 5  symbolStack: program id ; compound_stmt INPUT: . $ 移入

(132) stateStack: 0 2 3 4 5 7  symbolStack: program id ; compound_stmt . INPUT: $ 根据S-> program id ; compound_stmt .规约

(133) stateStack: 0 1  symbolStack: S  INPUT: $ accept
```

Figure: Runtime Output

output.txt

No. | State Stack | Symbol Stack | Input | Action

(112) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 18 10 19 29 35 22 symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts ; id := expr * factor INPUT: + id end end . $ 移入

(113) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 18 10 19 29 35 46 symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts ; id := expr * expr INPUT: + id end end . $ 移入

(114) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 18 10 19 29 35 46 33 symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts ; id := expr * expr + INPUT: id end end . $ 移入

(115) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 18 10 19 29 35 46 33 23 symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts ; id := expr * expr + id INPUT: end end . $ 根据factor-> id规约

(116) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 18 10 19 29 35 46 33 22 symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts ; id := expr * expr + factor INPUT: end end . $ 根据expr-> factor规约

(117) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 18 10 19 29 35 46 33 44 symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts ; id := expr * expr + expr INPUT: end end . $ 根据expr-> expr + expr规约

(118) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 18 10 19 29 35 46 symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts ; id := expr * expr INPUT: end end . $ 根据expr-> expr * expr规约

(119) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 18 10 19 29 symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts ; id := expr INPUT: end end . $ 根据stmt-> id := expr规约

(120) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 18 28 symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts ; stmt INPUT: end end . $ 根据stmts-> stmts ; stmt规约

(121) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts INPUT: end end . $ 移入

(122) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 6 8 17 symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else begin stmts end INPUT: end . $ 根据compound_stmt-> begin stmts end规约

(123) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 11 symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else compound_stmt INPUT: end . $ 根据stmt-> compound_stmt规约

(124) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 15 26 39 50 52 55 symbolStack: program id ; begin stmts ; for id := expr downto expr do if bool then stmt else stmt INPUT: end . $ 根据if_stmt-> if bool then stmt else stmt规约

(125) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 12 symbolStack: program id ; begin stmts ; for id := expr downto expr do if_stmt INPUT: end . $ 根据stmt-> if_stmt规约

(126) stateStack: 0 2 3 4 6 8 18 16 27 40 51 54 57 59 61 symbolStack: program id ; begin stmts ; for id := expr downto expr do stmt INPUT: end . $ 根据for_stmt-> for id := expr downto expr do stmt规约

(127) stateStack: 0 2 3 4 6 8 18 13 symbolStack: program id ; begin stmts ; for_stmt INPUT: end . $ 根据stmt-> for_stmt规约

(128) stateStack: 0 2 3 4 6 8 18 28 symbolStack: program id ; begin stmts ; stmt INPUT: end . $ 根据stmts-> stmts ; stmt规约

(129) stateStack: 0 2 3 4 6 8 symbolStack: program id ; begin stmts INPUT: end . $ 移入

(130) stateStack: 0 2 3 4 6 8 17 symbolStack: program id ; begin stmts end INPUT: . $ 根据compound_stmt-> begin stmts end规约

(131) stateStack: 0 2 3 4 5 symbolStack: program id ; compound_stmt INPUT: . $ 移入

(132) stateStack: 0 2 3 4 5 7 symbolStack: program id ; compound_stmt . INPUT: $ 根据S-> program id ; compound_stmt .规约

(133) stateStack: 0 1 symbolStack: S INPUT: $ accept

Figure: Runtime Output

No. | State Stack | Symbol Stack | Input | Action

```
(127) stateStack: 0 2 3 4 6 8 18 13  symbolStack: program id ; begin stmts ; for_stmt INPUT: end . $ 根据stmt-> for_stmt规约
(128) stateStack: 0 2 3 4 6 8 18 28  symbolStack: program id ; begin stmts ; stmt INPUT: end . $ 根据stmts-> stmts ; stmt规约
(129) stateStack: 0 2 3 4 6 8  symbolStack: program id ; begin stmts INPUT: end . $ 移入
(130) stateStack: 0 2 3 4 6 8 17  symbolStack: program id ; begin stmts end INPUT: . $ 根据compound_stmt-> begin stmts end规约
(131) stateStack: 0 2 3 4 5  symbolStack: program id ; compound_stmt INPUT: . $ 移入
(132) stateStack: 0 2 3 4 5 7  symbolStack: program id ; compound_stmt . INPUT: $ 根据S-> program id ; compound_stmt .规约
(133) stateStack: 0 1  symbolStack: S  INPUT: $ accept
```

Figure: Runtime Output

- Generating output files in build/ folder, using Shell scripts.

```
ChenMac:pascal-compiler wasdns$ ls
LICENSE                 docs                    run_lexer.py
README                  lexer_tests             run_lexer_ply.py
build                   parser_tests            run_parser_demo.sh
cleanup.sh              ply_frontend            src
ChenMac:pascal-compiler wasdns$ ./run_parser_demo.sh
ChenMac:pascal-compiler wasdns$ cd build/
ChenMac:build wasdns$ ls
action_and_goto.txt     grammar.txt             parser
error.txt               input.txt               slr.txt
first_and_follow.txt    output.txt
ChenMac:build wasdns$
```

Results of running parser

1.first_and_follow.txt: Print the first set and follow set
2.action_and_goto.txt: Print the action and goto table
3.output.txt: Print the runtime stack and applied actions
4.error.txt: The runtime error log

```
582  int main() {
583      char gramarFile[50] = "./grammar.txt";
584      char outputFile[50] = "./output.txt";
585      char actionAndGotoFile[50] = "./action_and_goto.txt";
586      char DFAFile[50] = "./slr.txt";
587      char errorFile[50] = "./error.txt";
588      char inputFile[50] = "./input.txt";
589
590      init();
591      getGrammar(gramarFile);
592      getCanonical();
593
594      calFirst();
595      printFirst();
596      calFollow();
597      printFollow();
598
599      printDFA(DFAFile);
600      setActionAndGoto();
601      printActionAndGoto(actionAndGotoFile);
602      readInput(inputFile);
603      solve(outputFile, errorFile);
604      return 0;
605  }
```

Figure: Overview of Main Procedure

Outline

- 1.Introduction
- 2.Journal of testing
- **3.Conclusion**

Conclusion

- Introduction of SLR(1) and Our Program;
- Giving an Example of Testing Our Program.

Experiences

- Requiring enough time to complete this task.
- Fully understanding of parser mechanism.
- The other things are coming soon. Stay Tuned!

That’s all.

Thank you!

Group Members: 吴媛媛, 林诗尧, 陈翔