Compiler Principle

Prof. Dongming LU

Mar. 18th, 2024

Content

3 Parsing

3.4 USING PARSER GENERATORS

Yacc (“Yet another compiler-compiler”)

- **A classic and widely used parser generator**

A Yacc specification: three sections separated by %% marks

parser declaration

%%

grammar rules

%%

programs

- **Parser declaration:** a list of N, T and so on
- **Programs:** ordinary C code usable from the semantic action
- **Grammar rules:** production of the form

✓ exp : exp PLUS exp { semantic action }

GRAMMAR 3.30

1. $P \rightarrow L$
2. $S \rightarrow$ id := id
3. $S \rightarrow$ while id do S
4. $S \rightarrow$ begin L end
5. $S \rightarrow$ if id then S
6. $S \rightarrow$ if id then S else S
7. $L \rightarrow S$
8. $L \rightarrow L$; S

Yacc version of Grammar 3.0

```
%{
Int yylex(void)
Void yyerror(char *s) { EM_error(EM_tokPos, “%s”,s)}
%}
% token ID WHILE BEGIN END DO IF THEN ELSE SEMI ASSIGN
% start prog
%%
prog: stmlist
stm: ID ASSIGN ID
     | WHILE ID DO stm
     | BEGIN stmlist END
     | IF ID THEN stm
     | IF ID THEN stm ELSE stm
stmlist: stm
         | stmlist SEMI stm
```

Semantic Actions are omitted

CONFLICTS

- **shift-reduce conflict**
  - ✓Resolved using shift by default in Yacc
- **reduce-reduce conflict**
  - ✓Resolved using the rule appears early in the grammar

shift-reduce conflicts are acceptable

```
state 17: shift/reduce conflict
              (shift ELSE, reduce 4)
  stm : IF ID THEN stm.
  stm : IF ID THEN stm. ELSE stm
  ELSE   shift 19
  .          Reduce by rule 4
```

From: Figure3.32 LR states for Grammar 3.30

PRECEDENCE DIRECTIVES

Ambiguous grammars are **still be useful** if finding ways to resolve the conflict.

An example

$E \rightarrow id$
$E \rightarrow num$
$E \rightarrow E*E$
$E \rightarrow E/E$
$E \rightarrow E+E$
$E \rightarrow E-E$
$E \rightarrow (E)$

Highly ambiguous

Many conflicts in LR(1) parsing table

PRECEDENCE DIRECTIVES

$E \rightarrow E * E\,.$	$+$
$E \rightarrow E.\, + E$	(any)

Bind * tighter than +, reduce instead of shift.

$E \rightarrow E + E\,.$	$+$
$E \rightarrow E.\, + E$	(any)

Make + left-associative, reduce instead of shift.

PRECEDENCE DIRECTIVES

- **Yacc uses precedence directives to resolve this class of shift-reduce conflicts**

% nonassoc EQ NEQ

% left PLUS MINUS

% left TIMES DIV

% right EXP

$E \rightarrow E * E .$	+
$E \rightarrow E . + E$	(any)

- **The precedence declarations give priority to Token**
- **The priority of rule is given by the last token**

PRECEDENCE DIRECTIVES

```
%{ declarations of yylex and yyerror %}
%token INT PLUS MINUS TIMES UMINUS
%start exp
%left PLUS MINUS
%left TIMES
%left UMINUS

%%

exp : INT
      | exp PLUS   exp
      | exp MINUS  exp
      | exp TIMES  exp
      | MINUS  exp     %prec UMINUS
```

- **−6 * 8 is parsed as (−6) * 8, not −(6 * 8)**

- **The token UMINUS is never returned by the lexer.**
- **The directive %prec UMINUS gives the rule exp::= MINUS exp the highest precedence**

Syntax Versus Semantics

```
%token ID ASSIGN PLUS MINUS AND EQUAL
%start stm
%left OR
%left AND
%left PLUS
%%
stm : ID ASSIGN ae
      | ID ASSIGN be
 be : be OR be
      | be AND be
      | ae EQUAL ae
      | ID
ae : ae PLUS ae
     | ID
```

be : ID
ae: ID

conflict

defer this analysis until the "semantic" phase of the compiler

$S \rightarrow \text{id} : E$
$E \rightarrow \text{id}$
$E \rightarrow E \,\&\, E$
$E \rightarrow E = E$
$E \rightarrow E + E$

3.5 ERROR RECOVERY

RECOVERY USING THE ERROR SYMBOL

- **Local error recovery mechanisms**
  - ✓ **Adjusting the parse stack and the input *at the point where the error was detected* in a way that will allow parsing to resume.**
- **Yacc parser generator**
  - ✓ **Uses a special *error* symbol to control the recovery process.**

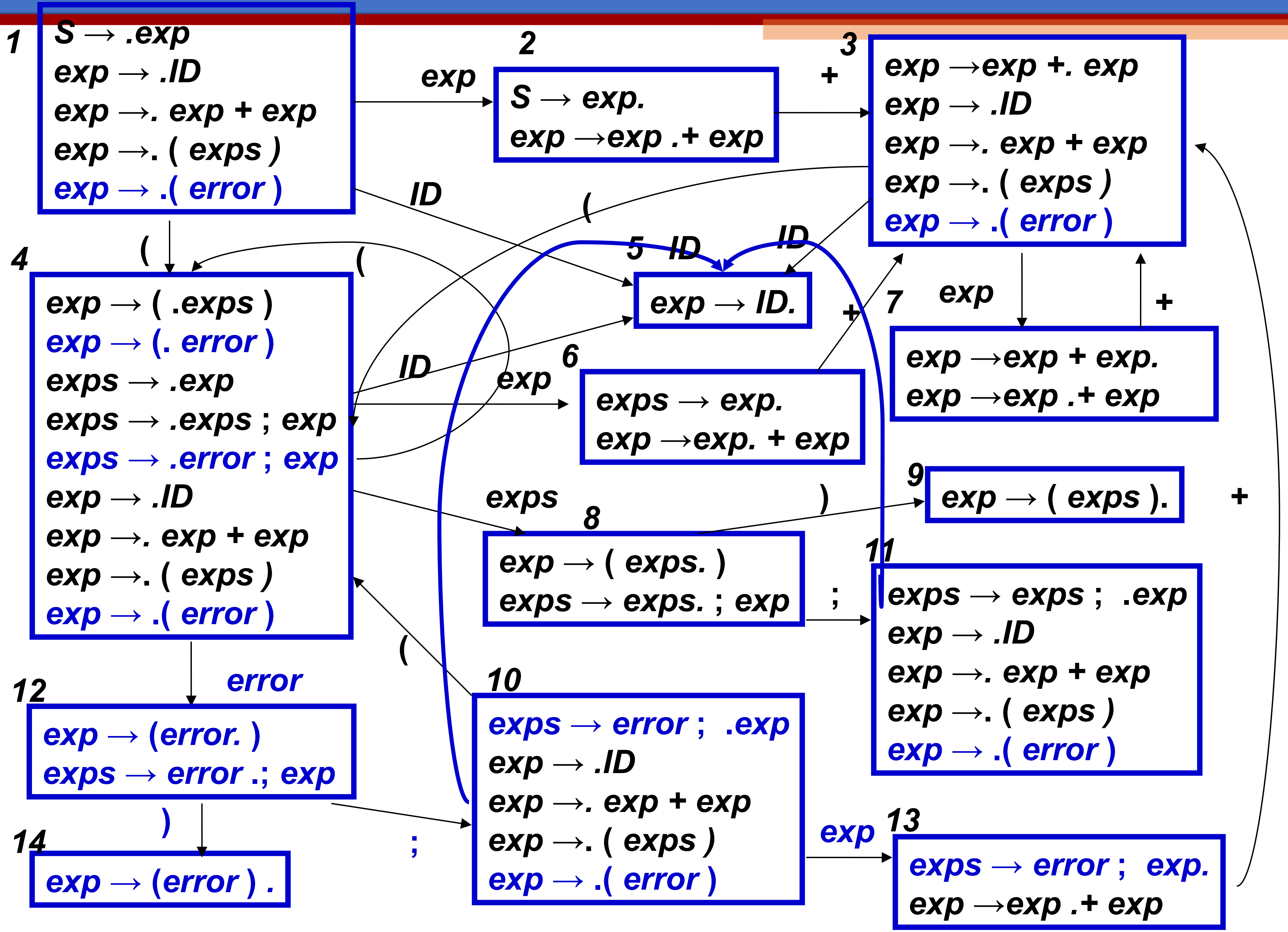
RECOVERY USING THE ERROR SYMBOL

exp* → *ID
exp* → *exp* + *exp
***exp* → (*exps*)**
exps* → *exp
exps* → *exps* ; *exp

***exp* → (*error*)**
exps* → *error* ; *exp

- **if a syntax error is encountered in the middle of an expression, the parser should skip to the next semicolon or right parenthesis and resume parsing.**

These are called *synchronizing tokens*


1
S → .exp
exp → .ID
exp →. exp + exp
exp →. (exps)
exp → .(error)
2
S → exp.
exp →exp .+ exp
3
exp →exp +. exp
exp → .ID
exp →. exp + exp
exp →. (exps)
exp → .(error)
4
exp → (.exps)
exp → (. error)
exps → .exp
exps → .exps ; exp
exps → .error ; exp
exp → .ID
exp →. exp + exp
exp →. (exps)
exp → .(error)
5
exp → ID.
6
exps → exp.
exp →exp. + exp
7
exp →exp + exp.
exp →exp .+ exp
8
exp → (exps.)
exps → exps. ; exp
9
exp → (exps).
10
exps → error ; .exp
exp → .ID
exp →. exp + exp
exp →. (exps)
exp → .(error)
11
exps → exps ; .exp
exp → .ID
exp →. exp + exp
exp →. (exps)
exp → .(error)
12
exp → (error.)
exps → error .; exp
13
exps → error ; exp.
exp →exp .+ exp
14
exp → (error) .
exp
ID
(
+
exps
)
;
error

RECOVERY USING THE ERROR SYMBOL

error **is considered a terminal symbol.**

- **When the LR parser reaches an error state, it takes the following actions:**

1. **Pop the stack (if necessary) until a state is reached in which the action for the *error* token is *shift*.**
2. **Shift the *error* token.**
3. **Discard input symbols (if necessary) until a lookahead is reached that has a nonerror action in the current state.**
4. **Resume normal parsing.**

S → exp
exp → ID
exp → exp + exp
exp → (exps)
exps → exp
exps → exps ; exp
exp → (error)
exps → error ; exp

input tokens: (a++;b)$

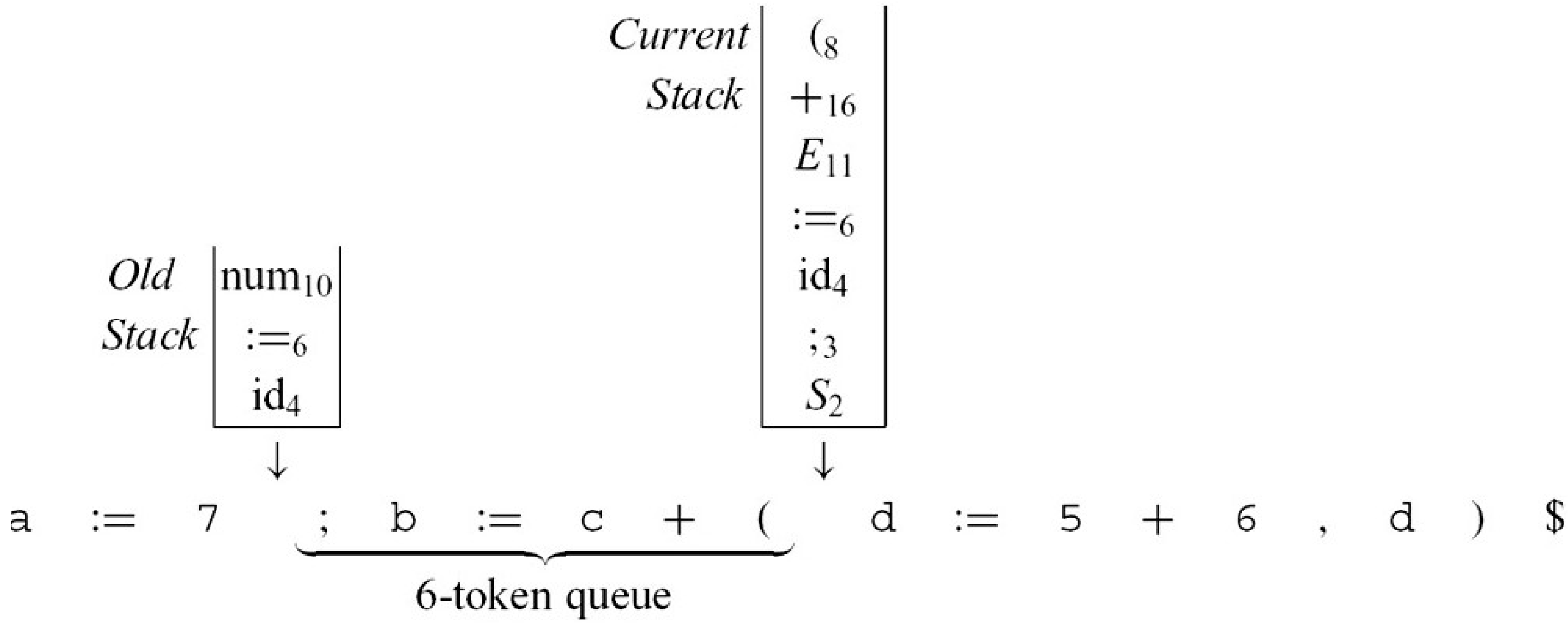
Stack	Input	action
1	(a++;b)$	S4
14, (	a++;b)$	s5
145, (a	++;b)$	r
146, (e	++;b)$	s3
1463, (e+	+;b)$	error
14, (	+;b)$	s12
1412, (error	;b)$	s10
141210, (error;	b)$	s5
1412105, (error; b	)$	r
14121013, (error; e	)$	r
148, (exps	)$	s9
1489, (exps)	$	r
12,e	$	r

GLOBAL ERROR REPAIR

- **Global error repair :** finds the smallest set of insertions and deletions that would turn the source string into a syntactically correct string, *even if the insertions and deletions are not at a point where an LL or LR parser would first report an error*.

- **Burke-Fisher error repair :** single-token insertion, deletion, or replacement at every point that occurs no earlier than *K* tokens before the point where the parser reported the error.

GLOBAL ERROR REPAIR



- The advantage of this technique:
  - ✓ The LL(*k*) or LR(*k*) (or LALR, etc.) grammar is not modified at all (no *error* productions)
  - ✓ Nor are the parsing tables modified.
- The parsing engine, which interprets the parsing tables, is modified.

The end of Chapter 3(4)