Compiladores:
Análisis Sintáctico

Pontificia Universidad Javeriana Cali
Ingenieria de Sistemas y Computación
Prof. Gloria Inés Alvarez V.

Análizador Sintáctico Ascendente: Conflictos

- Una gramática ambigua no puede ser LR(k)

- Pero el analizador puede ser adaptado para reconocer ciertas gramáticas ambiguas, usando reglas que especifican la solución al conflicto:
  - Usando precedencia y asociatividad
  - Resolver el conflicto desplazamiento-reducción optando por el desplazamiento

Recuperación de Error en análisis sintáctico LR

- Los errores se detectan en la tabla de acciones (no en la tabla de goto)
- El parsing LR canónico no realiza reducciones antes de anunciar el error.
- Los parsing SLR y LALR realizan reducciones, pero no hacen desplazamiento de un símbolo de entrada erroneo en la pila.

Recuperación de Error en análisis sintáctico LR

- Recuperación de error en modo pánico:
  - Desciende por la pila buscando un estado s que tenga un goto a un no terminal A (que represente una pieza importante del programa ej. Instrucción, bloque, expresión...)
  - Se descartan símbolos de la entrada, hasta encontrar un símbolo a que pertenece a FOLLOW (A)
  - El parser realiza goto[s,A]

Recuperación de Error en análisis sintáctico LR

- Recuperación de error a nivel de frase:
  - Crea rutinas de recuperación de error para cada entrada erronea de la tabla de parsing, decidiendo con base en los errores usuales que los programadores cometen.
  - La rutina modifica los símbolos en el tope de la pila.
  - Los cambios deben garantizar que el parser no entre en un ciclo infinito.

Generador de analizadores sintácticos: Yacc

- Se encuentra en el módulo PLY de python
- Requiere la definición de un archivo de entrada, en el cual se expresa la gramática mediante funciones python
- Genera un archivo con la tabla de análisis sintáctico llamado: *parsetab.py*
- Genera un archivo de depuración llamado: *parser.out*

Ejemplo

```
# Yacc example

import ply.yacc as yacc

# Get the token map from the lexer.  This is required.
from calclex import tokens

def p_expression_plus(p):
    'expression : expression PLUS term'
    p[0] = p[1] + p[3]

def p_expression_minus(p):
    'expression : expression MINUS term'
    p[0] = p[1] - p[3]

def p_expression_term(p):
    'expression : term'
    p[0] = p[1]

def p_term_times(p):
    'term : term TIMES factor'
    p[0] = p[1] * p[3]

def p_term_div(p):
    'term : term DIVIDE factor'
    p[0] = p[1] / p[3]

def p_term_factor(p):
    'term : factor'
    p[0] = p[1]

def p_factor_num(p):
    'factor : NUMBER'
    p[0] = p[1]

def p_factor_expr(p):
    'factor : LPAREN expression RPAREN'
    p[0] = p[2]

# Error rule for syntax errors
def p_error(p):
    print "Syntax error in input!"
```

Ejemplo

```
# Build the parser
yacc.yacc()

# Use this if you want to build the parser using SLR instead of LALR
# yacc.yacc(method="SLR")

while 1:
   try:
       s = raw_input('calc > ')
   except EOFError:
       break
   if not s: continue
   result = yacc.parse(s)
   print result
```

Ejemplo, agrupamiento de producciones

```
def p_binary_operators(p):
    '''expression : expression '+' term
                  | expression '-' term
       term       : term '*' factor
                  | term '/' factor'''
    if p[2] == '+':
        p[0] = p[1] + p[3]
    elif p[2] == '-':
        p[0] = p[1] - p[3]
    elif p[2] == '*':
        p[0] = p[1] * p[3]
    elif p[2] == '/':
        p[0] = p[1] / p[3]
```

Ejemplo parser.out

```
Unused terminals:


Grammar

Rule 1     expression -> expression PLUS expression
Rule 2     expression -> expression MINUS expression
Rule 3     expression -> expression TIMES expression
Rule 4     expression -> expression DIVIDE expression
Rule 5     expression -> NUMBER
Rule 6     expression -> LPAREN expression RPAREN

Terminals, with rules where they appear

TIMES                : 3
error                :
MINUS                : 2
RPAREN               : 6
LPAREN               : 6
DIVIDE               : 4
PLUS                 : 1
NUMBER               : 5

Nonterminals, with rules where they appear

expression           : 1 1 2 2 3 3 4 4 6 0


Parsing method: LALR


state 0

    S' -> . expression
    expression -> . expression PLUS expression
    expression -> . expression MINUS expression
    expression -> . expression TIMES expression
    expression -> . expression DIVIDE expression
    expression -> . NUMBER
    expression -> . LPAREN expression RPAREN

    NUMBER          shift and go to state 3
    LPAREN          shift and go to state 2
```

Manejo de errores

- El usuario define la función p_error(), la cual hace que el analizador entre en modo recuperación. No volverá a reportar error hasta que haya reconocido correctamente 3 tokens consecutivos
- Si no hay acción de recuperación el token se reemplaza por el token “error”
- Se pueden hacer producciones que contengan el token error como una forma de hacer recuperación (ver Doc PLY 5.8)