LANGUAGE PROCESSORS

UNIT 5: TOP-DOWN PARSING

uc3m

David Griol Barres
dgriol@inf.uc3m.es
Computer Science Department
Carlos III University of Madrid
Leganés (Spain)

CC BY NC SA

OUTLINE

- Possible methodologies
- Backtracking
- Recursive Descent
- Top-Down Predictive Parsing
  - Table-driven LL(1) Parsing

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Possible methodologies

- **BACKTRACKING**
- **RECURSIVE DESCENT**
- **PREDICTIVE PARSING**

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Backtracking

- Based on the information the parser currently has about the input, a decision is made to go with one particular production:
  - If it leads to a dead end:
    - backtrack to the previous decision point.

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Backtracking: example

- <u>Grammar</u>:
  S –> bab | bA
  A –> d | cA
- Input string to be parsed: bcd

1.	S - > bab	match b	ab dead-end, <u>backtrack</u>
2.	S - > bA	match b	bA
3.	A –> d	bd	dead-end, <u>backtrack</u>
4.	A –> cA	match bc	bcA
5.	A –> d	match bcd	end

Backtracking

- A backtracking approach may be tractable for small grammar such as above → slow and impractical for most programming language grammars

↓

Necessary more efficient methodologies

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Recursive Descent

- **Recursive-descent parser** : consists of several small functions, one for each nonterminal in the grammar:
  - We call the functions that correspond to the left side nonterminal of the productions we are applying.
  - If these productions are recursive, we end up calling the functions recursively.
  - Each time we arrive to a nonterminal, we call the function that defines it.
- Problem: Backtrack

Recursive Descent: Example

program –> function_list

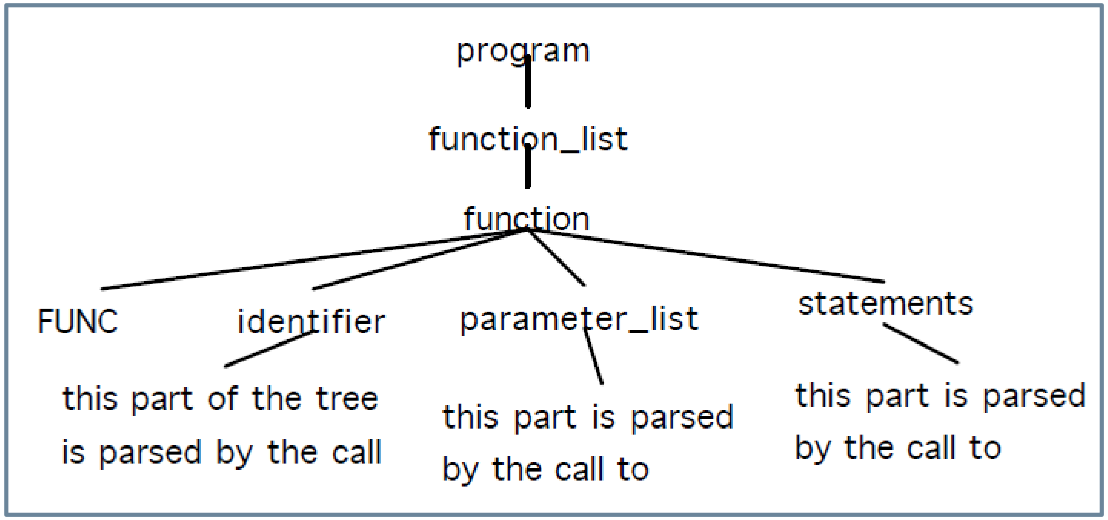
function_list –> function_list function | function

function –> FUNC identifier (parameter_list) statements

```
void ParseFunction()
{
   if (lookahead != T_FUNC) { // anything not FUNC here is wrong
      printf("syntax error \n");
      exit(0);
   } else
      lookahead = yylex();    // global 'lookahead' holds next token
   ParseIdentifier();
   if (lookahead != T_LPAREN) {
      printf("syntax error \n");
      exit(0);
   } else
      lookahead = yylex();
   ParseParameterList();
   if (lookahead!= T_RPAREN) {
      printf("syntax error \n");
      exit(0);
   } else
      lookahead = yylex();
   ParseStatements();
}
```

Recursive Descent: Example

- Parse tree generated:



David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Recursive Descent: Example

- E.g. if statetements:

  if_statement –> IF expression THEN statement close_if

  close_if –> ENDIF | ELSE statement ENDIF

```
void ParseIfStatement()
{
   MatchToken(T_IF);
   ParseExpression();
   MatchToken(T_THEN);
   ParseStatement();
   ParseCloseIf();
}

void ParseCloseIf()
{
   if (lookahead == T_ENDIF)  // if we immediately find ENDIF
      lookahead = yylex();    // predict close_if -> ENDIF
   else {
      MatchToken(T_ELSE);     // otherwise we look for ELSE
      ParseStatement();       // predict close_if -> ELSE stmt ENDIF
      MatchToken(T_ENDIF);
   }
}
```

Top-Down Predictive Parsing

- Predictive parser: choose the production to apply solely on the basis of the next input symbol and the current nonterminal being processed.

⇩

The grammar must take a particular form: *LL(1)*

- Necessary conditions:
  - no left-recursive productions,
  - left-factored.
- Sometimes we do not have a predictive grammar nor we can modify it to become a predictive grammar.

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Top-Down Predictive Parsing

Definition of the FIRST set:

- let α be a string of grammar symbols, **FIRST(α)** is the set of terminals that begin the strings derived from α.
  - FIRST(α)={x | ($\alpha \rightarrow_{*}$ x.β), (x $\in \Sigma_T \cup \{\lambda\}$), ($\alpha \in \Sigma^*$)}
- To compute FIRST (u) where u is of the form $X_1, X_2, \ldots, X_n$ do the following:
  1. If X_1 is a terminal, add X_1 to FIRST(u) and you are finished
  2. Else $X_1 \in \Sigma N$, add FIRST(X_1) - λ to FIRST(u)
     a. If $X_1 \rightarrow^* \lambda$, add FIRST($X_2$) - λ to FIRST(u).
     Furthermore, if $X_2 \rightarrow^* \lambda$, add FIRST($X_3$) - λ to FIRST(u), etc.
     b. If $X_1, X_2, \ldots, X_n \rightarrow^* \lambda$, add λ to the FIRST(u)

Top-Down Predictive Parsing

- <u>Example FIRST set :</u>

  E ::= **T**.**E’**

  E’ ::= +.**T**.**E’** | λ

  T ::= **F**.**T’**

  T’ ::= *.**F**.**T’** | λ

  F ::= (.**E**.) | Id

  - FIRST(**E**) = { (, Id }
  - FIRST(**T**) = { (, Id }
  - FIRST(**F**) = { (, Id }
  - FIRST(**E’**) = { +, λ }

  FIRST(**T**.*.Id) = { (, Id }

  FIRST(Id.+.Id) = { Id }

  FIRST(Id) = { Id }

  FIRST(**T’**) = { *, λ }

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Top-Down Predictive Parsing

Definition of the FOLLOW set:

- $FOLLOW(A)=\{x | (S \rightarrow_* \alpha \cdot A \cdot \beta), (A \in \Sigma_N), (\alpha \in \Sigma^*), (\beta \in \Sigma^+),$
  $(x \in FIRST(\beta)-\{\lambda\})\}$
- Set of terminals that can appear inmediately to the right of A in some sentential form. If A is the rightmost symbol in some sentential form, then $ is in FOLLOW(A)

- Algorithm
  1. FOLLOW(S)={$}, S the start symbol and $ is the input right endmarker
  2. If there is a production $A \rightarrow \alpha B \beta \Rightarrow$
     $FOLLOW(B) = (FIRST(\beta)-\{\lambda\}) \cup FOLLOW(B)$
  3. If there is a production $A \rightarrow \alpha B \beta \mid \beta=\lambda$ or $\beta \rightarrow_* \lambda$ $(\lambda \in FIRST(\beta))$
     $\Rightarrow$ $FOLLOW(B) = FOLLOW(A) \cup FOLLOW(B)$
  4. Repeat until nothing can be added to any FOLLOW set

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Top-Down Predictive Parsing

- Example FOLLOW set:

  E ::= **T**.**E'**

  E' ::= +.**T**.**E'** | λ

  T ::= **F**.**T'**

  T' ::= *.**F**.**T'** | λ

  F ::= (.**E**.) | Id

Σ_N	FOLLOW
E	$,)
E'	$,)
F	$, *,), +
T	$, +,)
T'	$, +,)

David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Top-Down Predictive Parsing

For a grammar to be LL(1)

- No ambiguity
- No left recursion
- If A::= α | β then
  - FIRST(α) $\cap$ FIRST(β) = $\emptyset$. For no terminal *a* do both α and β derive strings beginning with *a* (no FIRST/FIRST conflicts)
  - At most one of α and β can derive the empty string
  - If $\beta \rightarrow_* \lambda$, then α does not derive any string beginning with a terminal in FOLLOW(A) (FIRST/FOLLOW conflicts)

Table-driven LL(1) Parsing

- It uses a table to store that production along with an explicit stack to keep track of where we are in the parse.

E ::= **T**.**E'**

E' ::= +.**T**.**E'** | λ

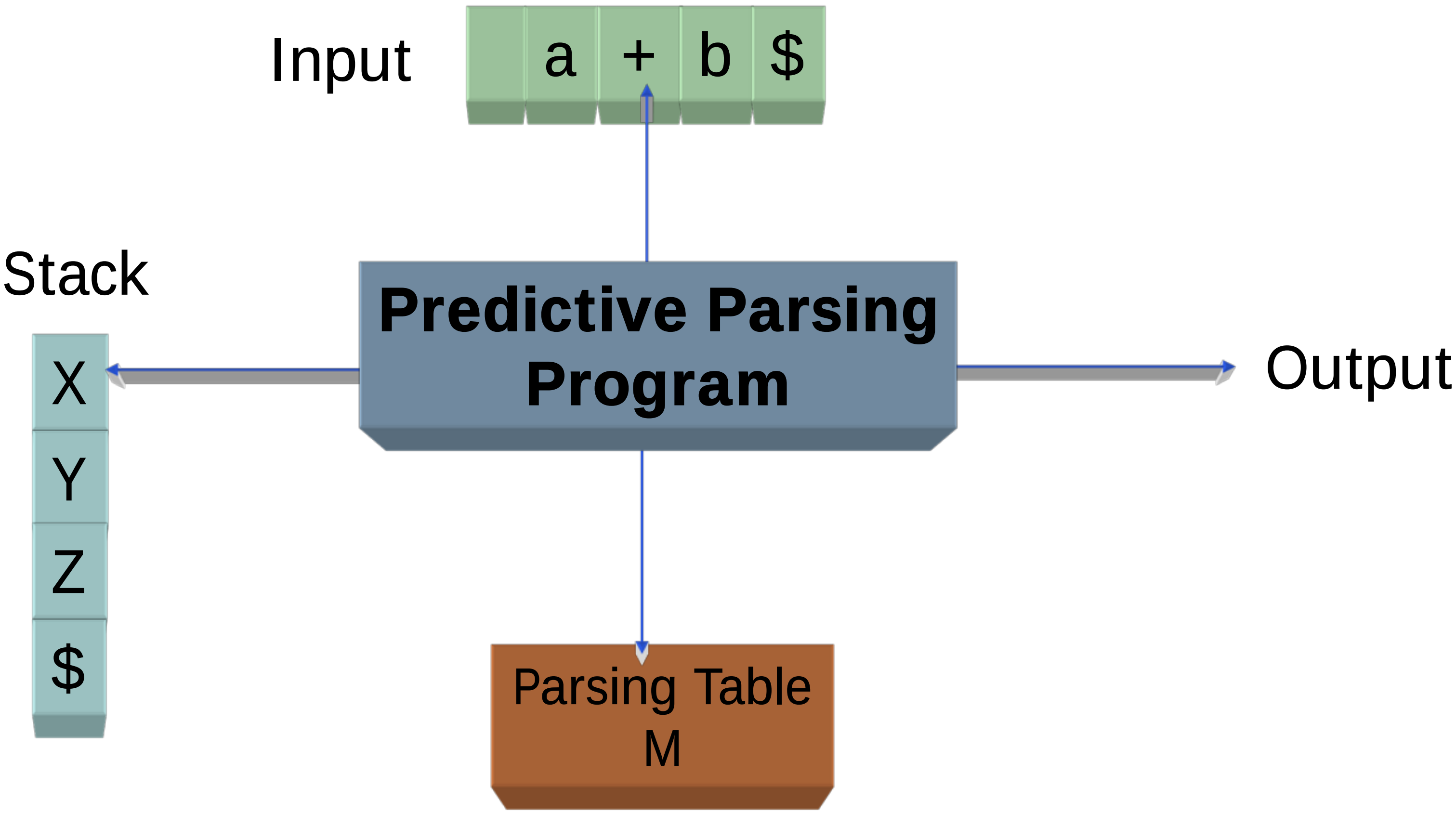
T ::= **F**.**T'**

T' ::= *.**F**.**T'** | λ

F ::= (.**E**.) | Id

	Id	**+**	*****	**(**	**)**	**$**
E	T.E'			T.E'		
E'		+.T.E'			λ	λ
T	F.T'			F.T'		
T'		λ	*.F.T'		λ	λ
F	Id			(.E.)		

Table-driven LL(1) Parsing



David Griol Barres Carlos III University of Madrid dgriol@inf.uc3m.es

Table-driven LL(1) Parsing

- **<u>To construct a predictive parsing table for a grammar</u>**:
- Compute the first and follow sets for the grammar.
- Build a table M:
  - the leftmost column labeled with all the nonterminals
  - the top row labeled with all the terminals in the grammar, along with $.

Table-driven LL(1) Parsing

- To construct a predictive parsing table for a grammar
  - Suppose $A \rightarrow \alpha$ where $a \in \Sigma_T$ | $a \in FIRST(\alpha)$. The parser will expand A by α when the current input symbol is a
- Algorithm to construct the table

ForAll (A::= α) $\in$ P do

  ForAll a $\in$ FIRST(α) do

    table(A,a)= α

  If λ $\in$ FIRST(α)

  then ForAll b $\in$ FOLLOW(A) do

    table(A,b)= λ

ForAll $A \in \Sigma_N$ and $c \in \Sigma_T$ do

  If table(A,c)= $\emptyset$

  then table(A,c)= error

Table-driven LL(1) Parsing

- Let X be the symbol on top of the stack and a the current input symbol. These two symbols determine the action of the parser
- There are three possibilities
  - X=a=\$, the parser halts and announces succesful termination
  - X=a$\neq$\$, the parser pops X off the stack and advances the input pointer to the next input symbol
  - $X \in \Sigma_N$, the program consults entry M[X,a]
    - If M[X,a]=UVW, the parser replaces X on top of the stack by WVU (with U on top)
    - If M[X,a]= error, then the parser calls an error recovery routine