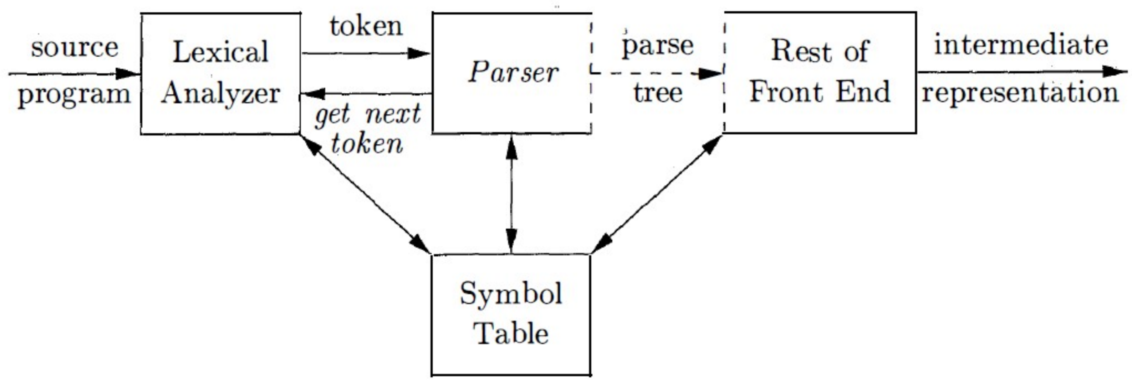
Syntax Analysis (Top-Down Parsing)

YoungMin Kwon

The Role of the Parser



- Obtains strings of tokens from the lexical analyzer and verifies that the string can be generated by the grammar.
- Efficient parsing methods
  - Top-down Parsers:
    - Build parse trees from the root to the leaves
    - Handmade parsers (e.g. LL grammars)
  - Bottom-up Parsers
    - Build parse trees from the leaves to the top
    - Generated by automated tools (e.g. LR grammars)

Context-Free Grammars

- Terminals (tokens)
  - Basic symbols from which strings are formed
- Nonterminals
  - Syntactic variables that denote sets of strings
- Start symbol
  - A nonterminal that denotes the language defined by the grammar
- Productions
  - The manner in which the terminals and nonterminals can be combined to from strings.

Notational Conventions

- a, b, c (small earlier part of the alphabet): a single terminal symbol.
- A, B, C (large earlier part of the alphabet): a single nonterminal symbol.
- x, y, z (small later part of the alphabet): a string of terminals.
- X, Y, Z (large later part of the alphabet): a single grammar symbol (a terminal or a nonterminal symbol).
- α, β, γ (small Greek letters): a string of grammar symbols.
- S: the start symbol.

Derivations

- A production is treated as a rewriting rule
  - The nonterminal on the LHS is replaced by the string on the RHS of the production.
  - Example E -> E + E | E * E | (E) | - E | ID,
    E => - E : "E derives - E"
    E => - E => - (E) => -(ID) : derivation of -(ID) from E
    $E =>^{*} -(ID)$
  - => : derives in one step,
    $=>^{*}$: derives in zero or more steps,
    $=>^{+}$: derives in one or more steps.
  - $\alpha =>^{*} \alpha$
    $\alpha =>^{*} \beta$ and $\beta => \gamma$, then $\alpha =>^{*} \gamma$

Derivations

- Let S be the start symbol of G, then a string of terminals w is in L(G) iff $S \Rightarrow^+ w$.
  - The string w is called a sentence of G
- A language generated by a grammar is called a context-free language
- Two grammars are called equivalent if they generate the same language.
- If $S \Rightarrow^+ \alpha$, where α may contain nonterminals, then α is a sentential form.
  - A sentence is a sentential form with no nonterminals.
  - Leftmost derivation: derivations in which only the leftmost nonterminal in any sentential form is replaced.
  - Rightmost derivation: derivations in which only the rightmost nonterminal in any sentential form is replaced.

Elimination of Left Recursion

- A grammar is left recursive if there is a derivation $A \Rightarrow^{+} A\alpha$ for a nonterminal A and some string α.
  - Top-down parsing mechanism cannot handle left-recursive grammars.
- In section 2.4, $A \rightarrow A\ \alpha \mid \beta$ is converted to

  $A \rightarrow \beta\ R$

  $R \rightarrow \alpha\ R \mid \epsilon$.
  - It does not eliminate left recursions involving two or more steps of derivations.

    $S \rightarrow A\ a \mid b$

    $A \rightarrow A\ c \mid S\ d \mid \epsilon$
  - Solution: give orders to nonterminals and if there is a production whose first RHS nonterminal is higher than the LHS, replace the RHS nonterminal with its productions.

Eliminating Left Recursion

arrange the nonterminals in some order $A_1, A_2, \ldots, A_n$.
for (each i from 1 to n) {
    for (each j from 1 to $i-1$) {
        replace each production of the form $A_i \to A_j\gamma$ by the
        productions $A_i \to \delta_1\gamma \mid \delta_2\gamma \mid \cdots \mid \delta_k\gamma$, where
        $A_j \to \delta_1 \mid \delta_2 \mid \cdots \mid \delta_k$ are all current A_j-productions
    }
    eliminate the immediate left recursion among the A_i-productions
}

- Example
  - S -> A a | b,
    A -> A c | S d | ϵ.
  - Order nonterminals as S, A
  - When i = 2, A -> S d is converted to
    A -> A c | A ad | bd | ϵ

S -> A a | b
A -> bd A’ | A’
A’ -> c A’ | ad A’ | ϵ

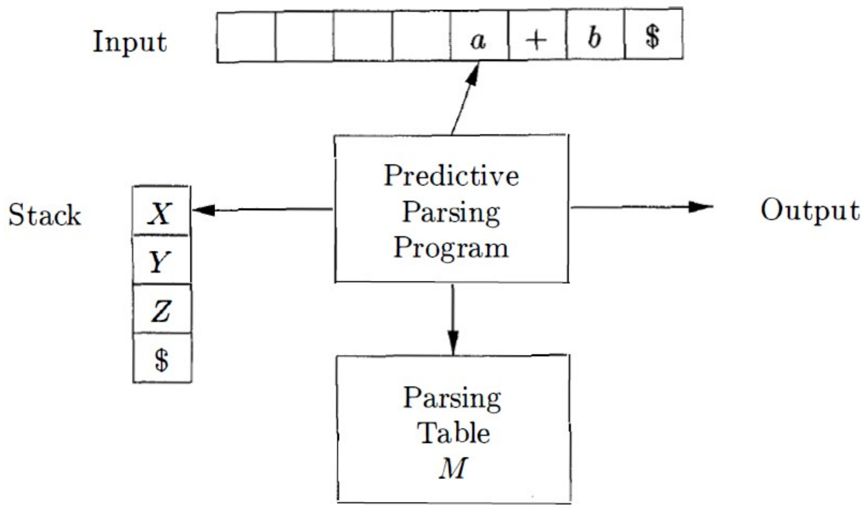
Left Factoring

- In predictive parsing, when we cannot select the production rule immediately, modify the grammar to defer the decision.
  - `stmt -> IF expr THEN stmt ELSE stmt`
`        |  IF expr THEN stmt`
- If $A \to \alpha\ \beta_1 \mid \alpha\ \beta_2$, then modify the grammar as
  $A \to \alpha\ A'$
  $A' \to \beta_1 \mid \beta_2$
  - `stmt -> IF expr THEN stmt stmt'`
`stmt' -> ELSE stmt | ϵ`

Top-Down Parsing

- In many cases, left-recursion removal and left factoring results in a grammar that can be parsed by a recursive-decent parser without backtracking (i.e. a predictive parser).

Nonrecursive Predictive Parsing



- Input: string of terminals followed by $
- Stack: sequence of grammar symbols with $ on the bottom.
- Parsing table: M[A,a], where A is a nonterminal, a is a terminal or $.

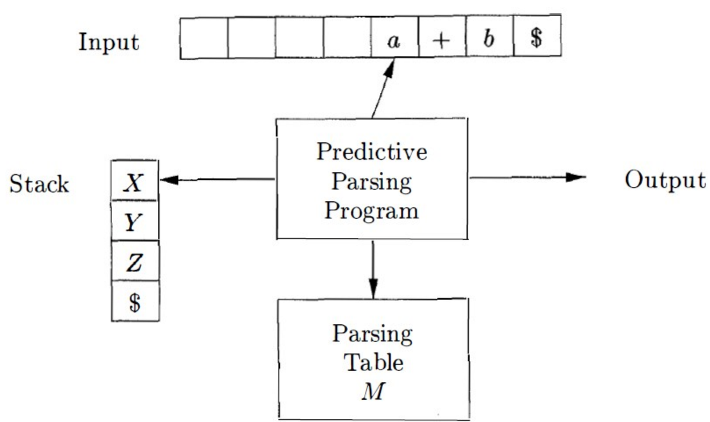
- Let X be the symbol on top of the stack, and a be the current input
- If X = a = $, announce the success.
- If X = a ≠ $, pops X and advance the input pointer
- If X is nonterminal
  – If M[X,a] = {X->UVW}, replace X on top of the stack with WVU (with U on top)
  – If M[X,a] = error, declare an error

Nonrecursive Predictive Parsing

set ip to point to the first symbol of w;
set X to the top stack symbol;
while ($X \neq \$$) { /* stack is not empty */
    if (X is a) pop the stack and advance ip;
    else if (X is a terminal) $error()$;
    else if ($M[X, a]$ is an error entry) $error()$;
    else if ($M[X, a] = X \rightarrow Y_1 Y_2 \cdots Y_k$) {
        output the production $X \rightarrow Y_1 Y_2 \cdots Y_k$;
        pop the stack;
        push $Y_k, Y_{k-1}, \ldots, Y_1$ onto the stack, with Y_1 on top;
    }
    set X to the top stack symbol;
}

Example

- id + id * id



NON - TERMINAL	INPUT SYMBOL id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to \mathbf{id}$			$F \to (E)$		

MATCHED	STACK	INPUT	ACTION
	$E\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	
	$TE'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	output $E \to TE'$
	$FT'E'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	output $T \to FT'$
	$\mathbf{id}\ T'E'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	output $F \to \mathbf{id}$
$\mathbf{id}$	$T'E'\$$	$+ \mathbf{id} * \mathbf{id}\$$	match **id**
$\mathbf{id}$	$E'\$$	$+ \mathbf{id} * \mathbf{id}\$$	output $T' \to \epsilon$
$\mathbf{id}$	$+ TE'\$$	$+ \mathbf{id} * \mathbf{id}\$$	output $E' \to + TE'$
$\mathbf{id} +$	$TE'\$$	$\mathbf{id} * \mathbf{id}\$$	match +
$\mathbf{id} +$	$FT'E'\$$	$\mathbf{id} * \mathbf{id}\$$	output $T \to FT'$
$\mathbf{id} +$	$\mathbf{id}\ T'E'\$$	$\mathbf{id} * \mathbf{id}\$$	output $F \to \mathbf{id}$
$\mathbf{id} + \mathbf{id}$	$T'E'\$$	$* \mathbf{id}\$$	match **id**
$\mathbf{id} + \mathbf{id}$	$* FT'E'\$$	$* \mathbf{id}\$$	output $T' \to * FT'$
$\mathbf{id} + \mathbf{id} *$	$FT'E'\$$	$\mathbf{id}\$$	match *
$\mathbf{id} + \mathbf{id} *$	$\mathbf{id}\ T'E'\$$	$\mathbf{id}\$$	output $F \to \mathbf{id}$
$\mathbf{id} + \mathbf{id} * \mathbf{id}$	$T'E'\$$	$\$$	match **id**
$\mathbf{id} + \mathbf{id} * \mathbf{id}$	$E'\$$	$\$$	output $T' \to \epsilon$
$\mathbf{id} + \mathbf{id} * \mathbf{id}$	$\$$	$\$$	output $E' \to \epsilon$

- Quiz: id + (id)

FIRST and FOLLOW

- FIRST(α)
  - The set of terminals that begin the strings derived from α
  - If $\alpha \Rightarrow^* a\ \beta$ then a is in FIRST(α)
  - If $\alpha \Rightarrow^* \epsilon$, then ϵ is in FIRST(α)
- FOLLOW(A)
  - The set of terminals a that can appear immediately to the right of A in some sentential form.
  - If $S \Rightarrow^* \alpha\ A\ a\ \beta$, then a is in FOLLOW(A)
- Compute FIRST(X)
  - If X is terminal, then FIRST(X) is $\{X\}$.
  - If $X \rightarrow \epsilon$ is a production, then add ϵ to FIRST(X).
  - If X is nonterminal and $X \rightarrow Y_1\ Y_2 \ldots Y_k$ is a production,
    - Add a to FIRST(X) if $a \in$ FIRST(Y_i) and $\epsilon \in$ FIRST(Y_j) for 1 <= j < i.
    - Add ϵ to FIRST(X) if $\epsilon \in$ FIRST(Y_j) for 1 <= j <= k.

FIRST and FOLLOW

- Compute FIRST($X_1 \ldots X_n$)
  - Add a to FIRST($X_1 \ldots X_n$) if $a \in$FIRST(X_i) and $\epsilon \in$FIRST(X_j) for 1 <= j < i.
  - Add ϵ to FIRST($X_1 \ldots X_n$) if $\epsilon \in$FIRST(X_j) for 1 <= j < n.
- Compute FOLLOW(A)
  - Add $ to FOLLOW(S) if S is the start symbol.
  - If there is a production $A \rightarrow \alpha\ B\ \beta$, then add FIRST($\beta$)-{$\epsilon$} to FOLLOW($B$).
  - If there is a production $A \rightarrow \alpha\ B$ or a production $A \rightarrow \alpha\ B\ \beta$ where $\epsilon \in$FIRST(β), then add FOLLOW(A) to FOLLOW(B).

FIRST and FOLLOW

- **Example**

```
E   -> T E'
E'  -> + T E' | ϵ
T   -> F T'
T'  -> * F T' | ϵ
F   -> ( E ) | ID
```

FIRST(E) = FIRST(T) = FIRST(F) = { (, ID }

FIRST(E') = { +, ϵ }

FIRST(T') = { *, ϵ }

FOLLOW(E) = FOLLOW(E') = {), $ }

FOLLOW(T) = FOLLOW(T') = { +,), $ }

FOLLOW(F) = { +, *,), $ }

Building a Predictive Parsing Table

- For each production $A \rightarrow \alpha$ do
  - For each terminal a in FIRST(α), add $A \rightarrow \alpha$ to $M[A,a]$.
  - If $\epsilon \in$ FIRST(α), add $A \rightarrow \alpha$ to $M[A,b]$ for each $b \in$ FOLLOW(A). (b is a terminal or \$)
  - Make each undefined entry of M be error.

Building a Predictive Parsing Table

- Example

```
E -> T E'
E' -> + T E' | ϵ
T -> F T'
T' -> * F T' | ϵ
F -> ( E ) | ID
```

FIRST(E) = FIRST(T) = FIRST(F) = { (, ID }
FIRST(E') = { +, ϵ }
FIRST(T') = { *, ϵ }
FOLLOW(E) = FOLLOW(E') = {), $ }
FOLLOW(T) = FOLLOW(T') = { +,), $ }
FOLLOW(F) = { +, *,), $ }

NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to \mathbf{id}$			$F \to (E)$		

LL(1) Grammars

- LL(1): a grammar whose predictive parsing table has no multiply-defined entries.
  - First L: scanning input from left to right
  - Second L: producing a leftmost derivation.
  - 1: using 1 input symbol of lookahead
- Grammar G is LL(1) iff whenever $A \rightarrow \alpha \mid \beta$ are two distinct productions of G, then the following holds
  - For no terminal a do both α and β derive strings beginning with a.
  - At most one of α and β can derive the empty string
  - If $\beta \Rightarrow^* \epsilon$, then α does not derive any string beginning with a terminal in FOLLOW(A).