Syntactic analysis

- **Parsing, syntax analysis, or syntactic analysis** is the process of analyzing a string symbols, either in natural language, computer language or data structures, based on the rules of the formal grammar.

There are two types of parser:

- *Top-down parser:*
  - starts at the root of derivation tree and fills in
  - picks a production and tries to match the input
  - may require backtracking
  - some grammars are backtrack-free (*predictive*)
- *Bottom-up parser:*
  - starts at the leaves and fills in
  - starts in a state valid for legal first tokens
  - as input is consumed, changes state to encode possibilities (*recognize valid prefixes*)
  - uses a *stack* to store both state and sentential forms

Bottom-up parser

Simple precedence Parsing

The context free grammar is called the grammar of simple precedence if:

- doesn't contain ε - productions,
- Doesn't contain the productions given in the following form:
- $A \rightarrow \alpha$, $B \rightarrow \alpha$ (the same right side)
- between two neighbors symbols from the string there is just one precedence relation.

Relations of simple precedence

This method of analysis was proposed by the Wirth și Weber and supposed that between two symbols x_1 and x_2 exist the following relations:

- **1)** $\mathbf{x_1 < x_2}$
- **2)** $\mathbf{x_1 < x_2}$
- **3)** $\mathbf{x_1 = x_2}$

The algorithm for constructing the set FIRST (A)

- **1st step:** For all productions
- $FIRST(A)=\{x|A\to x\alpha\}$
- **2nd step:** For all productions FIRST(A) if we have

  $B\in FIRST(A)$ and $B \in V_N$ then:

  - $FIRST(A)= FIRST(A) \cup FIRST(B)$.
- **3rd step:** repeat 2nd step until we have changes.
- **4th step:** Stop.

The algorithm for constructing the set LAST (A)

1st step: For all productions $A\to\alpha\ y$, $\alpha\in (V_T \cup V_N)^*$
we have
$LAST(A)=\{y|A\to\alpha\ y\}$
2nd step: For all elements from LAST(A) if we have
$B\in LAST(A)$ and $B \in V_N$, than:
$LAST(A)= LAST(A) \cup LAST(B)$.
3rd step: repeat 2nd step until we have changes.
4th step: Stop.

Example:

$G = (V_N, V_T, S, P)$:

$V_N = \{ E, T, F\}$; $S=\{E\}$

$V_T = \{+, *, (,), a\}$,

$P=\{E \to E+T|T$

$T \to T * F|F$

$F \to a|(E) \quad \}$

	FIRST	LAST
E	*E,T,F,a, (*	*T,F,a,)*
T	*T,F,a,(*	*F,a,)*
F	*a,(*	*a,)*

Algorithm for constructing the simple precedence relations

Step 1. If $U \to \alpha_1 x_1 x_2\ \alpha_2$ is the production from P, then $x_1 = x_2$.
Step 2. If $U \to \alpha_1 x_1 Y\ \alpha_2$ is the production from P, $Y \in V_N$, $x_1 \in (V_T \cup V_N)$ then $x_1 < x_2$ for $x_2 \in$ FIRST(Y).
Pasul 3. There are two cases:

a) If $U \to \alpha_1 Y x_2 \alpha_2$ is the production from P, $Y \in V_N$, $x_2 \in V_T$ then $x_1 > x_2$ for $x_1 \in$ LAST(Y).

b) If $U \to \alpha_1 YZ \alpha_2$ is the production from P,
$Y,Z \in V_N$, then $x_1 > x_2$ for $x_1 \in$ LAST(Y),
$x_2 \in$ FIRST (z)$\cap V_T$.

Pasul 4. $\$<x_1$, if $x_1 \in$ FIRST ($)
Pasul 5. $x_2 >\$$, if $x_2 \in$ LAST ($)
Pasul 6. Stop.

Example 1:

$G = (V_N, V_T, S, P)$:

$V_N = \{ E, T, F\}$; $S=\{E\}$

$V_T = \{+, *, (,), a\}$,

P={$E \rightarrow E+T|T$

$T \rightarrow T * F|F$

$F \rightarrow a|(E)$ }

	E	**T**	**F**	**+**	*****	**a**	**(**	**)**	**$**
E				=				=	
T				>	=			>	>
F				>	>			>	>
+		< =	<			<	<		
*****			=			<	<		
a				>	>			>	>
(	<=	<	<			<	<		
)				>	>			>	>
$	<	<	<			<	<		

Example 2: it is necessary to build the matrix of simple precedence, to analyze the string *<ab-a*ab*->* string and to build the derivation tree.

$G = (V_N, V_T, S, P)$:

V_N = { S, A, D, Z};

V_T = {-, *, a, b},

P={$S \rightarrow D$,

$A \rightarrow DZ$,

$D \rightarrow b$,

$Z \rightarrow$ -,

$D \rightarrow DA$,

$D \rightarrow a$,

$Z \rightarrow$ *}

a) Building the sets FIRST and LAST

	FIRST	**LAST**
S	D, a,b	D,A,a,b,Z,*,-
D	D,a,b	A,a,bZ,*,-
A	D,a,b	Z,*,-

Z	*,-	*,-

b) Building the matrix of precedence relations

1. $x_1 = x_2$

   D→DA D=A

   A→DZ D=Z

2. $x_1 < x_2$

   B→$\alpha_1 x_1 A \alpha_2$, $x_1 \in V_T \cup V_N$, $A \in V_N$, x_1<FIRST(A)

   D→DA D<FIRST(A)={D,a,b}

   A→DZ D<FIRST(Z)={*,-}

3. $x_1 > x_2$

   - B→$\alpha_1 A x_2 \alpha_2$, $x_2 \in V_T, A \in V_N$, LAST(A)>x_2.
   - B→$\alpha_1 A C \alpha_2$, $A, C \in V_N$,
     LAST(A)>FIRST(C) $\cap V_T$

D→DA LAST(D)>FIRST(A) $\cap V_T$
{A,a,b,z,*,-}>{a,b}

A→DZ LAST(D)>FIRST(Z) $\cap V_T$
{A,a,b,z,*,-}>{*,-}

	S	A	D	Z	a	b	*	-	$
S									
A					>	>	>	>	
D		=	<	=	<	<	<	<	
Z					>	>	>	>	
a					>	>	>	>	
b					>	>	>	>	
*					>	>	>	>	

-					>	>	>	>	
$									

$\$ < x_1$

$x_1 \in \mathrm{FIRST(S)}$

$x_2 > \$$

$x_2 \in \mathrm{LAST(S)}$

c) Analysis of string <ab-a*ab*->

<a>b>->a>*>a>b>*>->

<D<b>->a>*a>b>*>->

<D<D<->a>*a>b>*>->

<D<D=Z>a>*a>b>*>->

<D=A>a>*>a>b>*>->

<D<a>*>a>b>*>->

<D<D<*>a>b>*>->

<D<D=Z>a>b>*>->

<D=A>a>b>*>->

<D<a>b>*>->

<D<D<b>*>->

<D<D<D<*>->

<D<D<D=Z>->

<D<D=A>->

<D<D<->

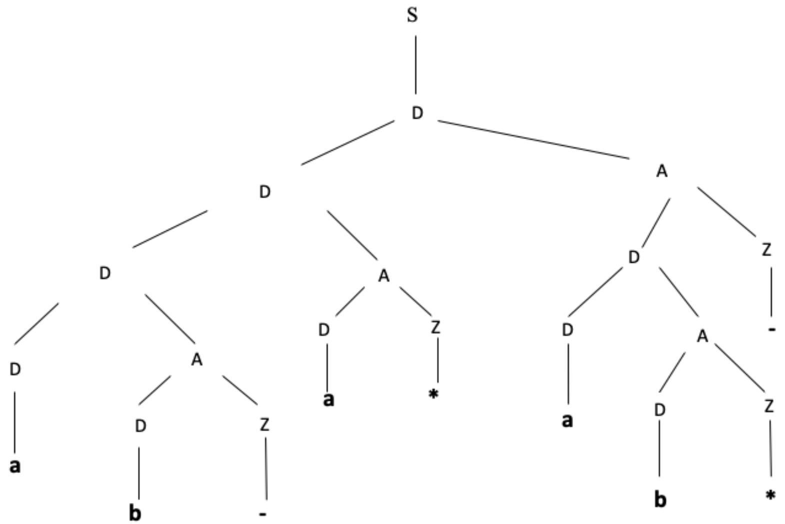
<D<D=Z>

<D=A>

<D>

S

d) Building the derivation tree



zsTop-Down Parsing

Predictive parsers, that is, recursive-descent parsers without backtracking, can be constructed for the LL(1) class grammars.

The first “L” stands for scanning input from left to right. The second “L” for producing a leftmost derivation. The “1” for using one input symbol of look-ahead at each step to make parsing decisions.

A top-down parser starts with the root of the parse tree, labeled with the start or goal symbol of the grammar.

To build a parse, it repeats the following steps until the fringe of the parse tree matches the input string

1. At a node labeled A, select a production $A \rightarrow \alpha$ and construct the appropriate child for each symbol of α
2. When a terminal is added to the fringe that doesn´t match the input string, backtrack
3. Find the next node to be expanded (must have a label in V_n)

The key is selecting the right production in step 1

$\Rightarrow$ should be guided by input string.

LL(1) grammar:

A context free grammar is **LL(1) grammar,** if for any production we have satisfied the following rules:

- There is no left recursion.
- There is no ambiguity.
- The grammar is left factoring.
- But some grammars that satisfies these conditions can be no LL(1) grammars.

To build the parser table, it should be obtained the FIRST and FOLLOW sets for the grammar**.**

Left factoring

Left factoring is a process by which the grammar with common prefixes is transformed to make it useful for Top down parsers.

In left factoring:

- It is used one production for each common prefixes.
- The common prefix may be a terminal or a non-terminal or a combination of both.

If $\alpha \neq \varepsilon$ then replace all of the A productions

$$A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \dots \mid \alpha\beta_n$$

with

$$A \to \alpha A'$$
$$A' \to \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

where A´ is a new non-terminal symbol.

Example:

Present the given the grammar in the LL(1) grammar form

$G = (V_N, V_T, S, P)$, $V_N = \{S, A, B\}$, $V_T = \{a, b\}$,

---

$P=\{S \to AB;$

$A \to bA|bB|a;$

$B \to A|Aa|b\}.$

Solution:

It is applied left factoring and it is obtained

$P'=\{S \to AB;$

$A \to bX|a$

$X \to A|B$

$B \to AY|b$

$Y \to \varepsilon|a\ \}$

FIRST sets

For a production A $\to$ $a\beta$ is defined the FIRST(A) as:

- the set of terminal symbols that begin strings derived from α:

$\{\ a \in V_t \mid A \to a\beta\ \}$

- If $A \to \varepsilon$ then $\varepsilon \in$ FIRST(A)

Algorithm to build FIRST(A):

1. If $A \to a$, $a \in V_t$, then FIRST(A) = $\{\ a\ \}$
2. If $A \to \varepsilon$ then add ε to FIRST(A)
3. It is given the production $A \to Y_1\ Y_2\ \dots\ Y_k$

   a) if $Y_1 \in V_N$ then put FIRST(Y_1) $/\{\varepsilon\}$ in FIRST(A)

   b) if $Y_1 \in V_N$ and $Y_1 \to \varepsilon$

   Put FIRST(Y_2) $/\{\varepsilon\}$ in FIRST(A)

   $\forall i$: $1 < i \le k$, if $\varepsilon \in$ FIRST(Y_1) $\cap \dots \cap$ FIRST(Y_{i-1})

   then put FIRST(Y_i) / $\{\varepsilon\}$ in FIRST(A)

   c) If $\varepsilon \in$ FIRST(Y_1) $\cap \dots \cap$ FIRST(Y_k). then put ε in FIRST(A)

Repeat until no more additions can be made.

Example:

Determine the FIRST set for the given grammar:

$G = (V_N, V_T, S, P)$, $V_N = \{S, A, B, C\}$, $V_T = \{a, b, d\}$,

$P=\{ S \to AB$

$A \to BCd$

$B \to a \mid \varepsilon$

$C \to b \mid \varepsilon \}$

Solution:

FIRST(S)=FIRST(A) /{ε} ={a, b,d}

FIRST(A)=FIRST(B) /{ε} $\cup$ FIRST(C) /{ε}={a,b,d}

FIRST(B)={a, ε}

FIRST(C)={b, ε}

FOLLOW sets

For a non-terminal A, define FOLLOW(A) represents: the set of terminals that can appear immediately to the right of A in some sentential form.

A terminal symbol has no FOLLOW set.

Algorithm to build FOLLOW(*A*):

1. If \$ is the input end marker and S is the start symbol then \$ $\in$ FOLLOW (S).
2. If $A \to \alpha B \beta$ then:
   a) FIRST(β) / {ε} $\subseteq$ FOLLOW(B), or FOLLOW(B)= FIRST(β) / {ε}.
   b) If $\beta = \varepsilon$ (i.e., $A \to \alpha B$) or $\varepsilon \in$ FIRST(β) (i.e., $\beta \to \varepsilon$) then FOLLOW(A) $\subseteq$ FOLLOW(B)

Repeat until no more additions can be made

Example:

Determine the FOLLOW set for the given grammar:

$G= (V_N, V_T, S, P)$, $V_N = \{S, A, B, C\}$, $V_T = \{a, b, d\}$,

$P=\{ S \to AB$

$A \to BCd$

$B \rightarrow a \mid \varepsilon$

$C \rightarrow b \mid \varepsilon\}$

Solution:

Applying the given rules are obtained:

FOLLOW(S)={\$} (1st rule)

FOLLOW(A)=FIRST(B) / {ε} ={a} (2.a rule)

FOLLOW(S) ⊆ FOLLOW(A) (2.b rule)

FOLLOW(S) ⊆ FOLLOW(B) (2.b rule)

FOLLOW(B)=FIRST(C)= {b}(2.a rule)

FOLLOW(B)={d}(2.b rule)

FOLLOW(C)={d}(2.a rule)

The FOLLOW sets are:

FOLLOW(S)= {\$}

FOLLOW(A)= {\$, a}

FOLLOW(B)= {\$, b, d}

FOLLOW(C)= {d}

Construction of a predictive parsing table

The following rules are used to construct the predictive parsing table:

1. If it is given the production $A \rightarrow \alpha$, then for each terminal a in FIRST(α), add $A \rightarrow \alpha$ to matrix $M[A, a]$

2. If it is given the production $A \rightarrow \alpha$ and ε is in FIRST(α), then for each terminal b in FOLLOW(A), add $A \rightarrow \alpha$ to matrix $M[A,b]$.

Predictive parsing algorithm

- Set input pointer (ip) to the first token a.
- Push \$ and start symbol to the stack.
- Set X to the top stack symbol.

```
while (X != $) { /*stack is not empty*/
  if (X is token a) pop the stack and advance ip;
     else if (X is another token) error();
     else if (M[X,a] is an error entry) error();
     else if (M[X,a] = X → Y1Y2…Yk) {
              output the production X → Y1Y2…Yk;
```

pop the stack; /* pop X */

/* leftmost derivation*/

push $Y_k, Y_{k-1}, \ldots, Y_1$ onto the stack, with Y_1 on top;

}

set X to the top stack symbol $Y1$;

} // end while

Example:

Analize the word ***ab-a*ab*-*** using theLL(1) parse :

$G = (V_N, V_T, S, P)$, $V_N = \{S, A, D, Z\}$, $V_T = \{a, b, -, *\}$,

$P = \{ S \to D$

$A \to DZ$

$D \to b \mid DA \mid a$

$Z \to - \mid *$

The given grammar should be transformed to the LL(1) grammar and it should be removed the left recursion.

$P' = \{ S \to D$

$A \to DZ$

$D \to bD' \mid aD'$

$D' \to AD' \mid \varepsilon$

$Z \to - \mid *$

Construction of the FIRST ald FOLLOW sets

FIRST(S) =FIRST(D)=$\{a, b\}$

FIRST(A)=FIRST(D) =$\{a, b\}$

FIRST(D)=$\{a, b\}$

FIRST(D')=FIRST(A)=$\{a, b\} \cup \{\varepsilon\}$

FIRST(Z) = $\{-, *\}$

FOLLOW(S) =$\{\$\}$

FOLLOW(S) $\subseteq$ FOLLOW(D)

FOLLOW(A)$\subseteq$ FOLLOW(Z)

FOLLOW(D)⊆ FOLLOW(D')
FOLLOW(D')⊆ FOLLOW(A)
FOLLOW(D)= FIRST(Z)
FOLLOW(A)= FIRST(D')

	FIRST	**FOLLOW**
S	{b, a}	{\$}
A	{b, a}	{ a, b, -, *, \$}
D	{b, a}	{-, *, \$}
D‘	{a, b, ε}	{-, *, \$}
Z	{-, *}	{ a, b, -, *, \$}

Construction of a predictive parsing table

Non-terminal	**Input symbol**				
	a	**b**	*****	**-**	**\$**
S	**D**	**D**			
A	**DZ**	**DZ**			
D	**aD‘**	**bD‘**			
D‘	**AD‘**	**AD‘**	ε	ε	ε
Z			*****	**-**	

Analysis of the word ***ab-a*ab*-***

Stack	**Input**	**Output**
S$	*ab-a*ab*-$*	*D*
D$	*ab-a*ab*-$*	*aD'*
aD'$	*ab-a*ab*-$*	***terminal***
D'$	*b-a*ab*-$*	*AD'*
AD'$	*b-a*ab*-$*	*DZ*
DZD'$	*b-a*ab*-$*	*bD'*
bD'ZD'$	*b-a*ab*-$*	***terminal***
D'ZD'$	*-a*ab*-$*	ε
ZD'$	*-a*ab*-$*	-
-D'$	*-a*ab*-$*	***terminal***
D'$	*a*ab*-$*	*AD'*
AD'$	*a*ab*-$*	*DZ*
DZD'$	*a*ab*-$*	*aD'*
aD'ZD'$	*a*ab*-$*	***terminal***
D'ZD'$	**ab*-$*	ε
ZD'$	**ab*-$*	*
**D'$*	**ab*-$*	***terminal***
D'$	*ab*-$*	*AD'*
AD'$	*ab*-$*	*DZ*
DZD'$	*ab*-$*	*aD'*
aD'ZD'$	*ab*-$*	***terminal***
D'ZD'$	*b*-$*	*AD'*
AD'ZD'$	*b*-$*	*DZ*
DZD'ZD'$	*b*-$*	*bD'*
bD'ZD'ZD'$	*b*-$*	***terminal***
D'ZD'ZD'$	**-$*	ε
ZD'ZD'$	**-$*	*
**D'ZD'$*	*-$*	***terminal***
D'ZD'$	*-$*	ε

ZD'\$	*-\$*	-
-D'\$	*-\$*	***terminal***
D'\$	*\$*	ε
\$	\$	*Accepted*

Practical Tasks

1. Convert the given grammar $G = (V_N, V_T, S, P)$ to the LL(1) grammar

- $V_N = \{A\}$; $V_T = \{q\}$;
  $P = \{ A \rightarrow qB|qC \}$
- $V_N = \{S, T, U, V\}$; $V_T = \{+,*,0,1,2,3,4,5,6,7,8,9\}$
  $P = \{ P=\{S \rightarrow T+S|T$
  $T \rightarrow U*T|U$
  $U \rightarrow (S)|V$
  $V \rightarrow 0|1|...|9\}$
- $V_N = \{S, A, B\}$; $V_T = \{a, b\}$
  $P' = \{S \rightarrow AB;$
  $A \rightarrow Aa|a$
  $B \rightarrow aA|a|b\}$

2. Determine the FIRSTand FOLLOW sets for the given grammar:

- $G = (V_N, V_T, S, P)$, $V_N = \{S, A, B, C\}$, $V_T = \{a, b, d\}$,
  $P = \{ S \rightarrow AB$
  $A \rightarrow BCd|\ \varepsilon$
  $B \rightarrow a|\ \varepsilon$
  $C \rightarrow b|\ \varepsilon\}$
- $G = (V_N, V_T, S, P)$, $V_N = \{S, A, B, C\}$, $V_T = \{a, b\}$,
  $P = \{ S \rightarrow CB$
  $A \rightarrow BCa|\ \varepsilon$
  $B \rightarrow a|\ \varepsilon$
  $C \rightarrow b|\ \varepsilon\}$

3. For the given grammar build the LL(1) parse table and analyze the given string:

- $G = (V_N, V_T, S, P)$, $V_N = \{S, A, B, D\}$, $V_T = \{a, b, c, d\}$,

  $P = \{ S \rightarrow dA$

  $A \rightarrow B \mid BcA$

  $B \rightarrow bD$

  $D \rightarrow a \mid aD \}$

  String: ***dbaacbaaa***

- $G = (V_N, V_T, S, P)$, $V_N = \{C, T, L, A, B\}$, $V_T = \{d, e, i, v, x, y\}$,

  $S = \{C\}$.

  $P = \{ C \rightarrow Ti$

  $T \rightarrow veB$

  $B \rightarrow Ld$

  $L \rightarrow A \mid LrA$

  $A \rightarrow x \mid y \}$

  String: ***vexryrxrxdi***