Jijel University
Faculty of exact sciences and computer science
Department of computer science

Top-Down Analysis

Compilation December 30, 2020

Dr. Hamida Bouaziz

Sommaire

- **Top-Down parsing** is a kind of syntactic analysis that attempts to find the **left-most derivations** for an input string w.

- It is equivalent to constructing a **parse tree** for the input string w that **starts from the root** and creates the nodes of the parse tree in a predefined order.

- The reason that top-down parsing seeks the left-most derivations for an input string and not right-most derivations is that the input string is scanned by the parser from the left to the right, one token at a time.

- One of the most efficient deterministic Top-down parsing methods currently known is: **The predictive Top-Down Analysis**.

The predictive Top-Down Analysis

- A **backtracking parser** is a non-deterministic recognizer of the language generated by a grammar.
- By carefully writing a grammar, you can get a top-down analysis without any backtrack, i.e. get a **deterministic** or a **predictive analyzer**.
- The predictive parser is capable of predicting which alternatives are the right choice for the expansion of non-terminals. In this case, writing carefully a grammar, means **eliminating the left-recursion** of the grammar and **left-factoring** the result.
- In this kind of analysis, a table called **predictive table** is used.

Let a grammar G=$\langle V_t, V_N, S, R \rangle$.
The **predictive table** of G is a two dimensional table, where:

- The rows represent the non-terminal symbols V_N of G.
- The columns are indexed by the terminal symbols V_t of G and the # symbol ($V_t \cup \{\#\}$). # marks the end.
- The cells of the table may contain production rules of G.

The first phase of building such table is the computation of the sets **FIRST and FOLLOW** of each symbol in V_N.

Five cases are considered:

1. If $A \rightarrow \alpha_1|\alpha_2|...|\alpha_n$ then FIRST(A)=FIRST(α_1) $\cup$ FIRST(α_2) $\cup$ ... $\cup$ FIRST(α_n).
2. FIRST(a) ={a}. where a$\in V_t$
3. FIRST(aγ) ={a}. where a$\in V_t$ and $\gamma \in (V_t \cup V_N)^*$.
4. FIRST(ε)={ε}
5. FIRST($Bx_1...x_n$) $\supset$ FIRST(B) $\setminus$ {ε} , where B$\in V_N$ and $X_i \in (V_t \cup V_N)^*$.
   - If $\varepsilon \in$ FIRST(B) then FIRST($Bx_1...x_n$) $\supset$ FIRST(x_1) $\setminus$ {ε}
   - If $\varepsilon \in$ FIRST(x_1) then FIRST($Bx_1...x_n$) $\supset$ FIRST(x_2) $\setminus$ {ε}
   - ....
   - If $\varepsilon \in$ FIRST(x_n) then $\varepsilon \in$ FIRST($Bx_1...x_n$).

Example

Consider G a CFG: G=$\langle$ V_t, V_N, Expression, R $\rangle$, where:

- V_t={+, *, (,), identifier}
- V_N={Expression, Term, AS, MS, Factor}
- R={
  Expression $\rightarrow$ Term AS
  AS $\rightarrow$ + Term AS | ε
  Term $\rightarrow$ Factor MS
  MS $\rightarrow$ * Factor MS | ε
  Factor $\rightarrow$ identifier | (Expression)
  }

Example(2)

Computation of FIRST sets:

- **FIRST(Expression)**= FIRST(Term AS) = FIRST(Term) = {identifier, (}
- **FIRST(AS)**= FIRST(+Term AS) $\cup$ FIRST(ε) = {+, ε}
- **FIRST(Term)**= FIRST(Factor MS) = {identifier, (}
- **FIRST(MS)**= FIRST(*Term MS) $\cup$ FIRST(ε) = {*, ε}
- **FIRST(Factor)**= FIRST(identifier) $\cup$ FIRST((Expression)) = {identifier, (}

The compute the set **FOLLOW(A)**, the following steps are repeated until stabilization:

1. If A=S (i.e. A is the start symbol) then # $\in$ FOLLOW(A).
2. If B$\rightarrow \alpha$Aβ with A,B$\in V_N$ and α, $\beta \in (V_t \cup V_N)^*$
   FIRST(β) $\setminus \{\varepsilon\} \subset$ FOLLOW(A)
   If $\varepsilon \in$ FIRST(β) then:
   FOLLOW(B) $\subset$ FOLLOW(A)
3. If B$\rightarrow \alpha$A then
   FOLLOW(B) $\subset$ FOLLOW(A)

Example

Consider the grammar of the previous example. Now, we compute the FOLLOW sets:

- FOLLOW(Expression)= $\{ \# \} \cup \{)\} = \{\#,)\}$
- FOLLOW(AS)= FOLLOW(Expression) = $\{\#,)\}$
- FOLLOW(Term)= FIRST(AS) $\{\varepsilon\} \cup$ FOLLOW(AS)$_{\varepsilon \in \text{FIRST(AS)}}$ = $\{ +, \#,) \}$
- FOLLOW(MS)= FOLLOW(Term) = $\{ +, \#,) \}$
- FOLLOW(Factor)= FIRST(MS) $\{\varepsilon\} \cup$ FOLLOW(Term)$_{\varepsilon \in \text{FIRST(MS)}} \cup$ FOLLOW(MS)$_{\varepsilon \in \text{FIRST(MS)}}$ = $\{ *, +, \#,) \}$

Predictive table construction algorithm

To build the predictive table, The following algorithm is used:

for (each rule of the form A$\rightarrow\alpha$) **do**
    if ($\alpha \neq \varepsilon$) **then**
        put A$\rightarrow \alpha$ in all cells T[A,a], where a $\in$ FIRST(α);
    else
        put A$\rightarrow \alpha$ in all cells T[A,b], where b$\in$ FOLLOW(A);

- All **empty cells** correspond to a **syntactic error**.
- If each cell of the table contains at most one production then the grammar is considered as a **LL(1) grammar**.

Example

To build the predictive analysis table of the grammar, already described in the previous examples, we use the FIRST and FOLLOW sets:

	FIRST	FOLLOW
Expression	identifier (	#)
AS	+ ε	#)
Term	identifier (	+ #)
MS	* ε	+ #)
Factor	identifier (	* + #)

The predictive table is the following:

	+	*	(	)	identifier	#
Expression			R 1		R 1	
AS	R 2.1			R 2.2		R 2.2
Term			R 3		R 3	
MS	R 4.2	R 4.1		R 4.2		R 4.2
Factor			R 6		R 5	

LL (1) is a predictive analysis based on the construction of a predictive analysis table for an LL (1) grammar. The LL (1) analysis is done **deterministically** by reading a single token at a time in the sentence to be analyzed. The meaning of letters LL (1) is:

- **L** (**L**eft to right): read the sentence to be analyzed from the left to the right.
- **L** (**L**eft most derivation): derive the sentence by executing the left most derivation.
- **1**: The parser needs to read only one token to decide which grammar's rule must be executed.

Formal definition of LL(1) grammar

A context-free grammar G=$\langle$ V_t, V_N, S, R $\rangle$ is said LL(1) **iff** it verifies the following conditions:
For each rule of the form A$\rightarrow \alpha \mid \beta$:

- FIRST(α) $\cap$ FIRST(β) = $\emptyset$
- If $\alpha \rightarrow^* \varepsilon$ then $\beta \not\Leftrightarrow^* \varepsilon$
- If $\alpha \rightarrow^* \varepsilon$ then FIRST(α) $\cap$ FOLLOW(A) =$\emptyset$

Note:

- A left-recursive grammar is not an LL(1) grammar.
- A non left-factorized grammar is not an LL(1) grammar.
- The previous two conditions are **necessary** for a grammar to be LL(1).

Example

Consider the previous grammar which is factorized and not left-recursive:
We have two rules that take the form A$\rightarrow \alpha \mid \beta$ which are:

- **AS$\rightarrow$ * Term AS** $\mid \varepsilon$
  - FIRST(+Term AS) $\cap$ FIRST (ε)={+} $\cup$ {ε}= $\emptyset$. Thus, condition 1 of LL(1) grammars is verified
  - The second production of AS derive to ε. However, the first one does not. Thus, condition 2 of LL(1) grammars is verified.
  - The first production does not derive to ε. However, the second production do. So, we must verify the intersection between the two sets FIRST(+ Term AS) and FOLLOW(AS). The intersection is equal to {+} $\cap$ {#,)} =$\emptyset$. Thus, condition 3 of LL(1) grammars is fulfilled.

Example(2)

- **MS$\rightarrow$ + Factor MS** $\mid \varepsilon$
  - FIRST(* Factor MS) $\cap$ FIRST (ε)={*} $\cup$ {ε}= $\emptyset$. Thus, condition 1 of LL(1) grammars is verified
  - The second production of MS derive to ε. However, the first one does not. Thus, condition 2 of LL(1) grammars is verified.
  - The first production does not derive to ε. However, the second production do. So, we must verify the intersection between the two sets FIRST(* Factor MS) and FOLLOW(MS). The intersection is equal to {*} $\cap$ {+,#,)} =$\emptyset$. Thus, condition 3 of LL(1) grammars is fulfilled.

Thus, we conclude that grammar is LL(1).

Example(1)

Consider the grammar G=$\langle$ {a,b,c}, {S,A,B,C}, S, R $\rangle$, where:
R={S $\rightarrow$ ABc | aSc, A $\rightarrow \varepsilon$ | aAB, B $\rightarrow$ bC | ε, C $\rightarrow$ Bc | aC | ε }
is G LL(1)?

- G is not left-recursive.
- G is factorized.
- The two previous criteria are **necessary** for a grammar to be LL(1) but **not sufficient**. Thus, we need to verify the aforementioned conditions.

Example(2)

For the rules $S \rightarrow ABc \mid aSc$:

- FIRST(ABc)= FIRST(A) $\{\varepsilon\} \cup$ FIRST(B) $\{\varepsilon\} \cup$ FIRST(c)= $\{a\} \cup \{b\} \cup \{c\} = \{a, b, c\}$
- FIRST(aSc) = {a}

FIRST(ABc) $\cap$ FIRST(aSc) = {a}.
The first condition of LL(1) grammars is not verified.
Thus, this grammar is not LL(1).

- The implementation of a LL(1) parser may be done using a **stack**.
- By having an LL(1) grammar G and a sentence w, the analyzer determines which production of G to be applied.
- The analysis starts from the start Symbol S of G, and goes on until the construction of sentence w.
- To determine the rule to be applied, the parser consults the predictive table of G.

- We have an input string. The input string ends with the end mark #. The stack at a given moment contains grammar symbols with the symbol #. # marks the bottom of the stack.
- Initially, the stack contains the start symbol S of the grammar above the # symbol.
- At each step of the analysis, one of the following 6 cases may be encountered:
  - The element on the top of the stack is a terminal. Thus:
    - If this element is the same as the current token **(case1)**, then go to the next token in the string to be analyzed.
    - Otherwise **(case2)**, stop the analysis and report a syntactic error.

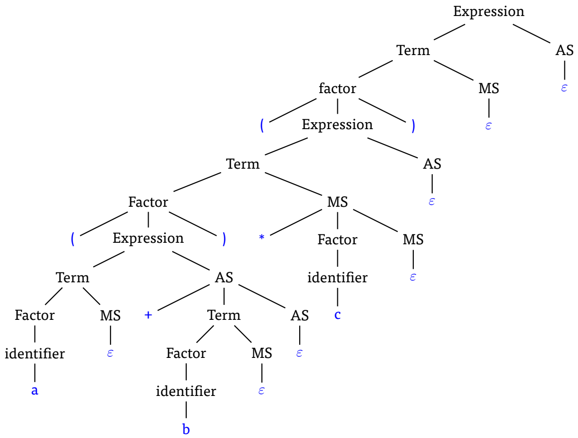
The operation of a LL(1) table-driven predictive parser(3)

- The element on the top of the stack is a non-terminal. Thus:
  - If the cell of the predictive table, indexed by the this non terminal and the current token, is empty **(case3)**, stop the analysis and report a syntactic error.
  - Otherwise **(case4)**, this non-terminal must be unstacked and replaced by the mirror of the right hand side of the rule in the cell of the analysis table.
- The element on the top of the stack is the symbol #, thus:
  - If the current token is the symbol # **(case5)**, then stop the analysis and declare that the string is accepted.
  - Otherwise **(case6)**, stop the analysis and report a syntactic error.

Example

Consider the following arithmetic expression:

((a+b)*c)



Input Buffer	Stack	action	rule
((a+b)*c)#	# Expression	production	Expression→Term AS
((a+b)*c)#	#AS Term	production	Term→Factor MS
((a+b)*c)#	#AS MS Factor	production	Factor→(Expression)
((a+b)*c)#	#AS MS) Expression (	matching	
(a+b)*c)#	#AS MS) Expression	production	Expression→Term AS
(a+b)*c)#	#AS MS) AS Term	production	Term→Factor MS
(a+b)*c)#	#AS MS) AS MS Factor	production	Factor→(Expression)
(a+b)*c)#	#AS MS) AS MS) Expression (	matching	
a+b)*c)#	#AS MS) AS MS) Expression	production	Expression→Term AS
a+b)*c)#	#AS MS) AS MS) AS Term	production	Term→Factor MS
a+b)*c)#	#AS MS) AS MS) AS MS Factor	production	Factor→identifier
a+b)*c)#	#AS MS) AS MS) AS MS identifier	matching	
+b)*c)#	#AS MS) AS MS) AS MS	production	MS→ε
+b)*c)#	#AS MS) AS MS) AS	production	AS→+ Term AS
+b)*c)#	#AS MS) AS MS) AS Term +	matching	
b)*c)#	#AS MS) AS MS) AS Term	production	Term→Factor MS
b)*c)#	#AS MS) AS MS) AS MS Factor	production	Factor→identifier
b)*c)#	#AS MS) AS MS) AS MS identifier	matching	
)*c)#	#AS MS) AS MS) AS MS	production	MS→ε
)*c)#	#AS MS) AS MS) AS	production	AS→ε
)*c)#	#AS MS) AS MS)	matching	
*c)#	#AS MS) AS MS	production	MS→* Factor MS
*c)#	#AS MS) AS MS Factor *	matching	
c)#	#AS MS) AS MS Factor	production	Factor→identifier

THE OPERATION OF A LL(1) TABLE-DRIVEN PREDICTIVE PARSER(6)

c)#	#AS MS) AS MS identifier	matching	
)#	#AS MS) AS MS	production	MS$\rightarrow\varepsilon$
)#	#AS MS) AS	production	AS$\rightarrow\varepsilon$
)#	#AS MS)	matching	
#	#AS MS	production	MS$\rightarrow\varepsilon$
#	#AS	production	AS$\rightarrow\varepsilon$
#	#	accept	