Chapter 3: Syntax Analysis (LR parsers)

What are we studying in this chapter?

- Introduction
- Bottom up parser
- Handle
- Handle pruning
- Shift reduce parser
- Conflicts during shift reduce parser

- 6 hours

3.1 Introduction

The top-down parser requires a grammar which do not have left recursion. But, bottom up parsers can be built for variety of grammars even if the grammar has left recursion. If grammar has left recursion, there is no need to eliminate left recursion. Bottom up parsing is more general than top-down parsing and it is as efficient as top-down parser. Bottom up parsing is preferred in practice. Bottom-up parsers handle a large class of grammars.

Now, let us see "What is bottom up parser?"

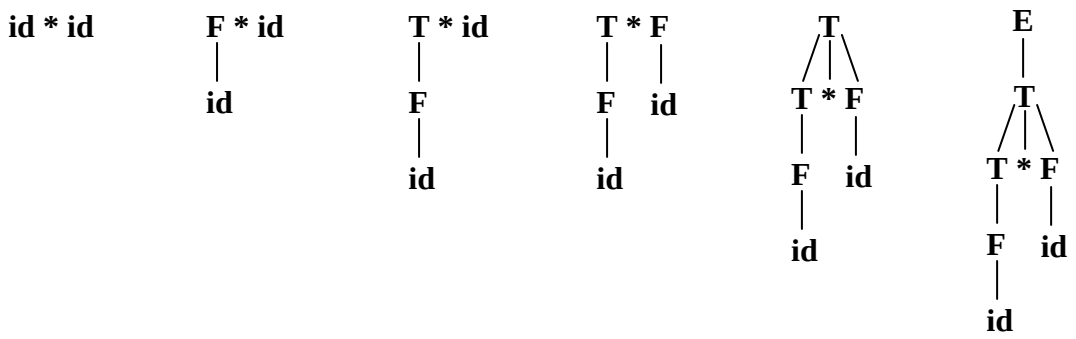
Definition: Bottom up parser reduces the given string of terminals *w* to the start symbol of the grammar. It is an attempt to reduce the input string *w* to the start symbol S using right most derivation in reverse. So, bottom up parsing corresponds to the construction of a parse tree for an input string *w* from the leaves (bottom) and working upwards towards the root (top) and hence the name *bottom up parser*. The leaves represent the string of terminals and the root corresponds to the start symbol. Thus, the process of obtaining the *start symbol* (root or top) from *string of terminals* (leaves or string of terminals) is called *bottom up parser*

Example: Consider the following grammar:

$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id$

Show the tree snapshots that are constructed for the string **id*id** using bottom up parser.

Solution: In bottom up parser, we start from string of terminals and move upwards towards the root and get the start symbol. The tree snapshots that are constructed for the input string **id*id** are shown below:



Observe from above snapshots that we have started from the string of terminals and obtain the start symbol and hence we say parsing is successful. Thus, we can think of bottom up parser is the one that reduces the string of terminals *w* to the start symbol S of the given grammar.

3.2 Reductions

Now, let us see "What is reduction?"

Definition: During bottom up parsing, the given string of terminals are reduced to the start symbol. The process of searching for right hand side of the production in a string consisting of terminals and/or non-terminals and replacing it with corresponding left hand side of the production is called *reduction.*

For example, consider the following grammar:

$E \rightarrow E + T \mid T$
$T \rightarrow T*F \mid F$
$F \rightarrow (E) \mid id$

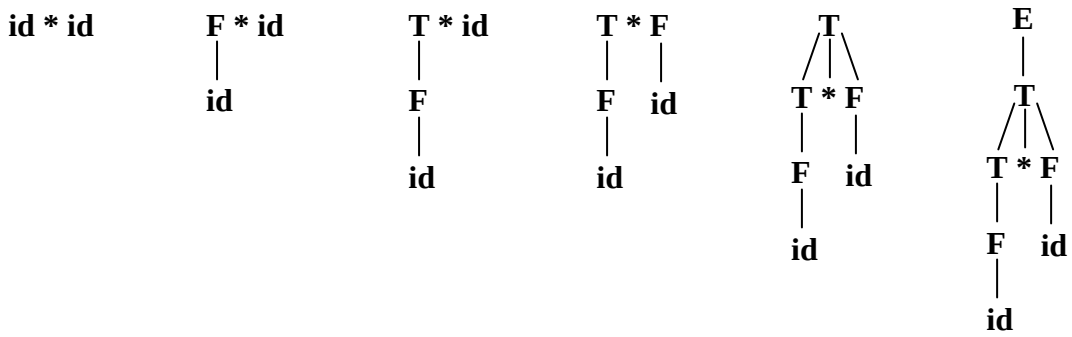
The right most derivation to get the string "**id * id"** is shown below:

$E \Rightarrow T$	using $E \rightarrow T$
$\Rightarrow T * F$	using $T \rightarrow T*F$
$\Rightarrow T * id$	using $F \rightarrow id$
$\Rightarrow F * id$	using $T \rightarrow F$
$\Rightarrow id * id$	using $F \rightarrow id$

Now, if we start from **id * id** and moving backwards by applying right most derivation in reverse we get the start symbols S and the various sentential forms that we get while moving backwards are:

id * id, F * id, T * id, T * F, T, E

Thus, using series of reductions we are above to get the start symbol and the tree snapshots corresponding to the above sentential forms are shown below:



Thus, using series of reductions as shown above, we are able to get the start symbol S and we say parsing is successful.

Note: The main points to be considered during reduction process are: *when to reduce* and *what production to use for reducing* as the parsing proceeds.

3.3 Handle pruning

Before understanding the term *handle pruning,* let us see "What is informal definition of handle?"

Definition: A substring that matches the right hand side of a production (i.e., that matches the body of a production) and replacing it with equivalent non-terminal on LHS of the production to get one step in right most derivation in reverse is called the *handle*.

For example, the handles that we get while we parse the string **id*id** are shown below:

Right sentential form in reverse	Handle	Reduction production
id * id	id	F → id
F * id	F	T → F
T * **id**	id	F → id
T * F	T * F	T → T * F
T	T	E → T
E		

Note: The strings of characters underlined are the handles.

Now, let us see "What is the formal definition of handle?"

Definition: Formally, if there is a right most derivation of the form $S \overset{*}{\Rightarrow} \alpha Aw \Rightarrow \alpha\beta w$ then the string β in i^{th} right sentential form $\alpha\beta w$ is the *handle* if there is a production of the form $A \rightarrow \beta$ such that replacing β by A in $\alpha\beta w$ results in previous $(i-1)^{th}$ right sentential form αAw. Note that *the string w to the right of the handle must contain only string of terminals*.

Now, let us see "What is handle pruning?"

Definition: The process of discovering a handle in a right sentential form γ_n and reducing it to the appropriate left hand side to get the previous right sentential form γ_{n-1} is called *handle pruning.*

For example, consider the following right most derivation:

$$S \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \gamma_3 \Rightarrow \ldots\ldots\ldots \Rightarrow \gamma_{n-1} \Rightarrow \gamma_n$$

Observe the following points from above right most derivation

- In n^{th} right sentential form γ_n, search for the handle β_n and replace it with corresponding left hand side of the production $A_n \rightarrow \beta_n$ to get $(n-1)^{th}$ right sentential form γ_{n-1}.
- Similarly, we locate the handle β_{n-1} in γ_{n-1} and replace β_{n-1} with corresponding left hand side of the production $A_{n-1} \rightarrow \beta_{n-1}$ to get $(n-2)^{nd}$ right sentential form γ_{n-2}.
- Repeating this process i.e., using repeated handle pruning we reduce the given string *w* to the start symbol S and we say parsing is successful.

Example: For the following grammar $S \rightarrow 0S1 \mid 01$, indicate the handles in the following right sentential form 00001111

Solution: The given grammar is shown below:

$$S \rightarrow 0S1 \mid 01$$

The given right sentential form 00001111 can be reduced to the start symbol as shown below:

Right sentential form in reverse	Handle	Reduction production
0 0 0 **0 1** 1 1 1	0 1	S → 0 1
0 0 **0 S 1** 1 1	0 S 1	S → 0 S 1
0 **0 S 1** 1	0 S 1	S → 0 S 1
0 **S** 1	0 S 1	S → 0 S 1
S		

Note: The strings of characters that are underlined are the handles. The handles are searched and are replaced with appropriate LHS of the production to get the start symbol

Example: For the following grammar "S → S S + | S S * | a" indicate the handles in the following right sentential form "**S S + a * a +**"

Solution: The given grammar is shown below:

S → S S + | S S * | a

The given right sentential form "**S S + a * a +**" can be reduced to the start symbol as shown below:

Right sentential form in reverse	Handle	Reduction production
S S + a * a +	S S +	S → S S +
S **a * a +**	a	S → a
S S * a +	S S *	S → S S *
S **a** +	a	S → a
S S +	S S +	S → S S +
S		

Note: The strings of characters that are underlined are the handles. The handles are searched and are replaced with appropriate LHS of the production to get the start symbol

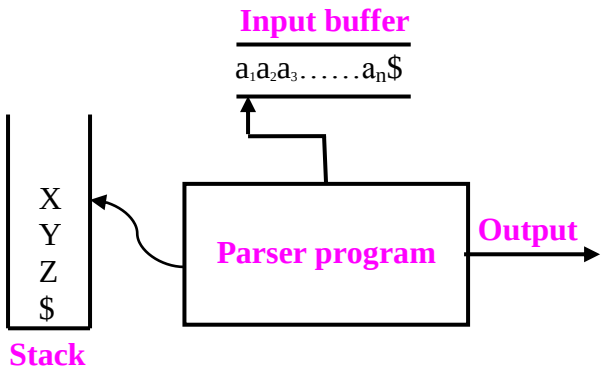
3.4 Shift reduce parser

Now, let us see "What is shift reduce parser? How shift reduce parser works?"

Definition: *Shift-reduce parser* is an efficient way of implementing bottom up parser using explicit stack. The stack contains grammar symbols and input buffer holds the string to be parsed. In this parser, we start from string of terminals and get the start symbol. During parsing, always the handle will appear on top of the stack just before it is identified as the handle.

3.6 💻 Syntax Analyzer

The block diagram showing the various parts of shift-reduce parser are shown below:



The predictive parser has four components namely:

- Input
- Stack
- Parsing program
- Output

- **Input :** The input buffer contains the string to be parsed and the input string ends with '$'. Here, $ indicates the end of the input.
- **Stack :** It contains sequence of grammar symbols and '$' is placed initially on top of the stack indicating stack is empty.
- **Parser :** It is a program which shifts the input symbols on the stack or reduce the handle on the stack to appropriate LHS of the production.

  The *initial configuration* of the parser is shown below:

STACK	INPUT
$	w$

  The *final configuration* of the parser is shown below:

STACK	INPUT
$S	$

  In the final configuration the parser halts and announces successful completion of parsing.

- **Output:** As output, the productions that are used are used to reduce are displayed till we get the start symbol on the stack

Working of the parser: The various actions performed by the shift-reduce parser are shown below:

1) The parser keeps shifting the input symbols on to the stack till the handle β appears on top of the stack.
2) The handle β is then reduced to appropriate LHS of the production
3) The above two steps are repeatedly performed till stack contains the start symbol and input is empty or an error is encountered.

Thus, the four possible actions of the shift reduce parser are:

- **Shift:** Here, the next input symbol is shifted on to the top of the stack.
- **Reduce:** By knowing the appropriate handle on the stack, the parser reduces this to the left hand side of the appropriate production.
- **Accept:** The parser announces successful completion of parsing
- **Error:** The parser discovers an error and the appropriate error message is displayed on the screen with the help of error handler and calls an error recovery routine.

Example: Show the sequence of moves made by the shift-reduce parser for the string **id*id** using the following grammar:

E → E +T | T
T → T*F | F
F → (E) | id

Solution: The sequence of moves made by the shift-reduce parser for the string **id*id** is shown below:

STACK	INPUT	ACTION
$	id * id$	shift
$ id	* id$	Reduce F→ id
$ F	* id$	Reduce T → F
$ T	* id$	Shift * [Instead of reducing E→ T]
$ T *	id$	Shift id
$ T * id	$	Reduce F→ id
$ T * F	$	Reduce T→ T * F
$ T	$	Reduce E→ T
$ E	$	ACCEPT

Since stack contains the start symbol and input is empty, we say parsing is successful. Observe the following points:

- The parser has started from the string **id*id$** in the input and it has reduced the input string to start symbol **E** on the stack.
- Thus, starting from string of terminals i.e., *bottom* (from the leaves) it has obtained the start symbol S which appears at the *top* of the parse tree (root)
- So, the shift reduce parser is called *bottom-up-parser*.

3.5 Conflicts during shift reduce parsing

The shift reduce parser cannot be used for all context free grammars. We may get some conflicts during parsing. Now, let us see "What are the conflicts that arise during shift reduce parsing?" The two types of conflicts that arise during shift reduce parsing are:

- Shift-reduce conflicts
- Reduce-reduce conflicts

- **Shift reduce conflicts:** During parsing, by knowing the contents of the stack and the next input symbol, if the parser cannot decide whether to shift the input the symbol on to the stack or to reduce the handle on top of the stack, it results in *shift-reduce conflict.* For example, consider the following grammar:

  E → E +T | T
  T → T*F | F
  F → (E) | id

  The sequence of moves made by the parser for the string **id*id** is shown below:

STACK	INPUT	ACTION
$	id * id	Shift **id**
$ id	* id	Reduce F → id
$ F	* id	

  Now, observe that the parser has reached a state where it cannot decide whether to shift input symbol * on to the stack or to reduce using the production T → F. This conflict is called shift-reduce conflict.

- **Reduce-reduce conflicts:** During parsing, if shift-reduce parser finds a handle on top of the stack, the parser has to reduce the handle to the left hand side of the production. But, if two or more productions have same handle on right hand side of the production, the parser cannot decide which production has to be selected for reducing. This conflict is called *reduce-reduce conflict.* For example, consider the following grammar:

  E → E +T | T
  T → T*F | F
  F → (E) | id

  The sequence of moves made by the parser for the string **id*id** is shown below:

STACK	INPUT	ACTION
$	id * id$	Shift **id**
$ id	* id$	Reduce F → id
$ F	* id$	Reduce T → F
$ T	* id$	Shift *
$ T *	id$	Shift **id**
$ T * id	$	Reduce F → id
$ T * F	$	

Now, observe that the parser has reached a state where it cannot decide whether to reduce the handle F to T using the production T → F or to reduce the handle T*F to T using the production T → T * F. This conflict is called *reduce-reduce conflict.*

Exercises

1) What is bottom up parser? Show the tree snapshots that are constructed for the string **id*id** using bottom up parser.
   E → E +T | T
   T → T*F | F
   F → (E) | id

2) What is reduction? What is informal definition of handle? What is the formal definition of handle?

3) What is handle pruning?

4) For the following grammar S → 0S1 | 01, indicate the handles in the following right sentential form 00001111

5) For the following grammar "S → S S + | S S * | a" indicate the handles in the following right sentential form "S S + a * a +

6) What is shift reduce parser? How shift reduce parser works?

7) Show the sequence of moves made by the shift-reduce parser for the string **id*id** using the following grammar:

   E → E +T | T
   T → T*F | F
   F → (E) | id

8) What are the conflicts that arise during shift reduce parser