Shift-Reduce Parsing

In this section, we discuss a bottom-up style of parsing called **shift-reduce parsing**. This parsing method is **bottom-up** because it attempts to construct a parse tree **for an input string beginning at the leaves** (the **bottom**) and working up **towards the root**. We can think of this process as one of "reducing" a string **W** to the **start symbol** of a grammar. At each step a string matching the right side of the production is replaced by the symbol on the left.

For example, consider the grammar

$S \longrightarrow a\ A\ c\ B\ e$

$A \longrightarrow Ab \mid b$

$B \longrightarrow d$

and the string "**abbcde**". We **want to reduce this string to S**. We **scan this string looking for substring** that matches the right side of any production. The substring **b** and d qualify. Let us choose the **left most b** and replace it by **A** the left side of the production $A \longrightarrow b$. We continue according to this step:-

$a\underline{\mathbf{b}}bcde \longrightarrow a\underline{Ab}cde \longrightarrow aAcde \longrightarrow \mathbf{aAcBe} \longrightarrow S$

Each replacement of the right side of a production by the left side in the process above is called "reduction". Thus, by a sequence of four reductions we were able to reduce **abbcde to S**. These reductions traced out a right most derivation in reverse

Informally, a **substring which is the right side** of a production such that **replacement of that substring by the production left side** leads eventually to a **reduction** to the **start symbol**, by a reverse of a rightmost derivation is **called a "handle"**. The process of bottom-up parsing may be viewed as one of finding and reducing handles.

Handles

A handle of a right sentential form y is a production A $\longrightarrow \beta$ and a position of y, where the string β may be found and replaced by A to produce the previous right-sentential form in the right most derivation of y.

That is, if s $\overset{*}{\underset{lm}{\Longrightarrow}} \alpha A w \underset{rm}{\Longrightarrow} \alpha\beta w$, then A $\longrightarrow \beta$ in the position following α is a handle of $\alpha\beta w$. The string w to the right of the handle contains only terminal symbols.

In the example above, abbcde is a right-sentential form whose handle is A $\longrightarrow$ b at the position 2. Likewise, a Abcde is a right-sentential form whose handle is A $\longrightarrow$ Ab at position 2.

Sometimes we shall say "the substring β is a handle of $\alpha\beta w$" if the position of β and the production A $\longrightarrow \beta$ we in mind are clear. If a grammar is **unambiguous**, then every right-sentential form of the grammar has exactly one handle.

EX: consider the following grammar:-

(1) E $\longrightarrow$ E+E
(2) E $\longrightarrow$ E*E
(3) E $\longrightarrow$ (E)
(4) E $\longrightarrow$ id

And consider the right most derivation.

$$E \underset{Rm}{\Longrightarrow} \underline{E+E} \underset{Rm}{\Longrightarrow} E+\underline{E*E} \underset{Rm}{\Longrightarrow} E+E*\underline{id3} \underset{Rm}{\Longrightarrow} E+\underline{id2}*id3 \underset{Rm}{\Longrightarrow} \underline{id1}+id2*id3$$

We have subscripted the id's for notational convenience and underlined a handle of each right-sentential form for example, id1 is the handle of the right sentential form id1+id2*id3, because id is the right side of the production E $\rightarrow$ id, and replacing id1 by E produces the previous right sentential form E+id2*id3.

Note: the string appearing to the right of the handle contains only terminal symbols.

Because grammar (2) is ambiguous, there is another rightmost derivation of the same string. This derivation begins E $\Longrightarrow$ E*E and produce another set of handles. In particular E+E is a handle of E+E*id3 according to this derivation.

Handle Pruning

A right most derivation in reverse, often called a "**canonical reduction sequence**", is obtained by "**handle pruning**". That is, we start with a string of terminal w which we wish to parse. If w is a sentence of the grammar at hand, then $w=y_n$, where y_n is the nth right-sentential form of same as yet unknown right most derivation:

$$S= \quad y_0 \underset{Rm}{\Rightarrow} y_1 \underset{Rm}{\Rightarrow} y_2 \underset{Rm}{\Rightarrow} y_{n-1} \underset{Rm}{\Rightarrow} y_n=w.$$

To reconstruct this derivation in reverse order, we locate the handle β_n in y_n and replace β_n by the left side of some production $A_n \longrightarrow \beta_n$ to obtain the previous right sentential form y_{n-1}. We then repeat this process. That is, we locate the handle β_{n-1}in y_{n-1} and reduce this handle to obtain the right sentential form y_{n-2}. If by continuing this process we produce a right sentential form consisting only of the start symbol S, then we halt and give successful completion of parsing. The reverse of the sequence of production use in the reduction is right most derivation of the input string.

EX: consider the **grammar (2)** and input string id1+id2*id3. Then following sequence of reductions reduce id1+id2+id3 to start symbol E:

Sentential Form	Handle	Reducing Production
id1+id2*id3	id1	$E \longrightarrow id$
E+id2*id3	id2	$E \longrightarrow id$
E+E*id3	id3	$E \longrightarrow id$
E+E*E	E*E	$E \longrightarrow E*E$
E+E	E+E	$E \longrightarrow E+E$
E		

Stack Implementation of Shift-Reduce Parsing

There **are two problems that must be solved if we are to automate parsing by handle pruning**. The **first** is **how to locate a handle in a right-sentential**, and the **second** is **what production to choose in case there is more than one production with the same right side**. Before we get to these questions, let us first consider the type of **data structures** to use in **handling pruning parser**.

A convenient way to **implement a shift-reduce parser** is to use a **stack** and **an input buffer**. We shall use $ to mark the bottom of the stack and the right of the input.

Stack	Input
$	W$

The parser operates by **shifting zero or more input symbols onto the stack until a handle β is on top of the stack**. The parser then reduces β to the **left side of the appropriate production**. The parser then reduces this cycle until it has detected an error or until the stack contains the start symbol and the input is empty.

Stack	Input
$S	$

In this configuration the parser halts and makes successful parsing.

EX: Let us step through the actions a **shift-reduce parser** might make in parsing the input string **id1+id2*id3** according to the grammar (2). The sequence is shown in Fig.5. Note that because grammar (2) has two right most derivations for this input there is another sequence of steps a shift reduces parser could take.

Seq	Stack	Input	Action
1	$	**id_1**$+id_2*id_3$$	Shift
2	$$id_1$	$+id_2*id_3$$	Reduce by $E \longrightarrow id$
3	$E	$+id_2*id_3$$	Shift
4	$E+	**id_2**$*id_3$$	Shift
5	$$E+id_2$	$*id_3$$	Reduce by $E \longrightarrow id$
6	$E+E	$*id_3$$	Shift
7	$E+E*	**id_3**$	Shift
8	$$E+E*id_3$	$	Reduce by $E \longrightarrow id$
9	$E+$\underline{\mathbf{E*E}}$	$	Reduce by $E \longrightarrow E*E$
10	$$\underline{E+\mathbf{E}}$	$	Reduce by $E \longrightarrow E+E$
11	$**E**	$	**Accept**

Fig.5: Shift-reduce parsing actions

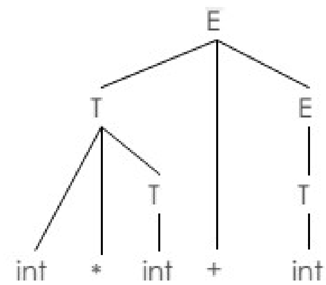
While the primary operations of the parser are shifting and reduce, there are actually **four possible actions** a shift reduce parser can make: **(1) shift, (2) reduce, (3) accept, and (4) error**

1. In a shift action, the next input symbol is shifted to the top of the stack.
2. In a reduce action, the parser knows the right end of the handle is at about the top of the stack. It must then locate the left end of the handle within the stack and decide with what non-terminal to replace the handle.
3. In an accept action, the parser complete parsing successfully.
4. In an error action, the parser discovers that a syntax error has occurred and calls an error recovery routine.

Examples

1. Bottom-up parsing reduces a string to the start symbol by inverting productions

a. leftmost derivation

String	**Grammar**	Parse Tree
int * int + int **int * T** + int **T** + **int** T + **T** **T + E** **E**	T → int * T T → int E → T E → T + E	

b. task: do rightmost derivation

2. Shift Reduce Parsing

1.

Parsing Stack	Input	Action
$	int * int + int$	Shift
$ int	* int + int$	reduce T→ int
$ T *	int + int$	Shift
$ T * int	+ int$	Shift
$ T * int+	int$	Shift
$ T * int+ int	$	Shift
Reject		

2.

Parsing Stack	Input	Action
$	int * int + int$	Shift
$ int	* int + int$	Shift
$ int *	int + int$	Shift
$ int * int	+ int$	Reduce T → **int**
$ int * **T**	+int$	Reduce T → **int * T**
$ **T**	+int$	Shift
$ T+	int$	Shift
$ T+ **int**	$	Reduce T → **int**
$ T+ **T**	$	Reduce E → **T**
$ T+ **E**	$	Reduce E → **T + E**
$E	$	**Accept**

3. Give bottom-up parses for the following input strings and grammars:

1. The input **abaca** according to the grammar **S → SbS | ScS | a**

2. The input **000111** according to the grammar **S → 0S1 | 01**

1.	**2.**
S → SbS | ScS | a **a**baca, S → a Sb**a**ca, S → a **SbS**ca, S → SbS Sc**a**, S → a **ScS**, S → ScS **S**	00**01**11, S → 01 0**0S1**1, S → 0S1 **0S1**, S → 0S1 **S**

4. Consider the context-free grammar:
S' → S
S → (S)S | ε
Parse steps for input string: **(ε) ε** (using **shift –reduce parsing**)

Parsing Stack	Input	Action
$	(ε) ε $	Shift
$(	ε) ε $	Shift
$(**ε**	) ε $	Reduce **S →ε**
$(**S**	) ε $	Shift
$(S)	$	Shift
$(S) **ε**	$	Reduce **S →ε**
$**(S)S**	$	Reduce **S →(S)S**
$**S**	$	Reduce **S' →S**
$**S'**	$	**accept**

5. Consider the context-free grammar:

S →αAz
A → BBy
B → γ
Parse steps for input string: **αBγyz** (using **shift –reduce parsing**)

Parsing Stack	Input	Action
$	αBγyz$	Shift
$α	Bγyz$	Shift
$αB	γyz$	Shift
$αBγ	yz$	Reduce **B →**γ
$αBB	yz$	Shift
$α**BBy**	z$	Reduce **A →BBy**
$αA	z$	Shift
$**αAz**	$	Reduce **S → αA z**
$S	$	**accept**

6. Given the Grammar:
S → S-S
S → S+S
S → (S)
S → a

Show how the string **a1-(a2+a3)** be generated by this grammar (using **Shift–Reduce parsing**) and Construct a parse tree for this string, where the **S → S-S** is the start symbol.

Parsing Stack	Input	Action
$	**a1-(a2+a3)**$	Shift
$**a1**	**-(a2+a3)**$	Reduce **S → a**
$**S**	**-(a2+a3)**$	Shift
$ **S-**	**(a2+a3)**$	Shift
$ **S-(**	**a2+a3)**$	Shift
$ **S-(a2**	**+a3)**$	Reduce **S → a**
$ **S-(S**	**+a3)**$	Shift
$ **S-(S+**	**a3)**$	Shift
$ **S-(S+a3**	**)**$	Reduce **S → a**
$ **S-(S+S**	**)**$	Shift
$ **S-(S+S)**	$	Reduce **S → (S+S)**
$ **S-(S)**	$	Reduce **S → (S)**
$**S-S**	$	Reduce **S → S-S**
$**S**	$	**Accept**