

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 "abcde". 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:-

$$\underline{a}bcde \longrightarrow a\underline{A}bcde \longrightarrow aAcde \longrightarrow aAcBe \longrightarrow S$$

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 $abcde$ 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 $A \rightarrow \beta$ 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 $s \xrightarrow{\text{lm}}^* \alpha A w \xrightarrow{\text{rm}} \alpha \beta w$, then $A \rightarrow \beta$ 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, $abbcd$ is a right-sentential form whose handle is $A \rightarrow b$ at the position 2. Likewise, $aAbcd$ is a right-sentential form whose handle is $A \rightarrow Ab$ 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 $A \rightarrow \beta$ 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) $E \rightarrow E + E$
- (2) $E \rightarrow E * E$
- (3) $E \rightarrow (E)$
- (4) $E \rightarrow id$

And consider the right most derivation.

$$E \xrightarrow{\text{Rm}} \underline{E + E} \xrightarrow{\text{Rm}} E + \underline{E * E} \xrightarrow{\text{Rm}} E + E * \underline{id3} \xrightarrow{\text{Rm}} E + \underline{id2} * id3 \xrightarrow{\text{Rm}} \underline{id1} + id2 * id3$$

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 $E \Rightarrow E * E$ 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 n th right-sentential form of same as yet unknown right most derivation:

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

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 :

Right-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		

We observe that the sequence of right-sentential forms in this example is just the reverse of the sequence in the rightmost derivation in the last example.

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.

<u>Stack</u>	<u>Input</u>
\$	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.

<u>Stack</u>	<u>Input</u>
\$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.

<i>Seq</i>	<i>Stack</i>	<i>Input</i>	<i>Action</i>
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+E*E	\$	Reduce by E \longrightarrow E*E
10	\$E+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.

There is an important fact that justifies the use of a stack in shift reduce parsing : The handle will always eventually appear on top of the stack, never inside. This fact becomes obvious when we consider the possible form of two successive steps in any rightmost derivation. These two steps can be of the form:

$$\begin{aligned}
 (1) \quad & S \xRightarrow[Rm]{*} \alpha Az \\
 & \quad \quad \quad \xRightarrow[Rm]{*} \alpha \beta B y z \\
 & \quad \quad \quad \xRightarrow[Rm]{*} \alpha \beta B y y z \\
 \text{Or} \quad & \\
 (2) \quad & S \Longrightarrow \alpha B x A z
 \end{aligned}$$

$$\xrightarrow[Rm]{*} \alpha Bxyz$$

$$\xrightarrow[Rm]{*} \alpha yxyz$$

In case (1), A is replaced by βBy , and then the right most derivation B on the right side is replaced by a y, in case (2), A is again replaced first, but this time the right side is string y of terminals only. The next right most non-terminal, B, will be somewhere to the left of y.

Let us consider the case (1) in reverse, where a shift-reduce parser has just reached the configuration.

<u>Stack</u>	<u>Input</u>
$\$ \alpha \beta y$	$yz \$$

The parser now reduces the handle y to the B to reach the configuration

<u>Stack</u>	<u>Input</u>
$\$ \alpha \beta B$	$yz \$$

Since B is the right most non-terminal in $\alpha \beta Byz$, the right end of the handle of $\alpha \beta Byz$ cannot occur inside the stack. The parser can therefore shift the string y onto the stack to reach the configuration

<u>Stack</u>	<u>Input</u>
$\$ \alpha \beta By$	$z \$$

In which βB is the handle, and it gets reduced to A. In case (2), in configuration

<u>Stack</u>	<u>Input</u>
$\$ \alpha \beta y$	$yz \$$

The handle y is on top of the stack. After reducing the handle y to B, the parser can shift the string xy to get the next handle y on top of the stack:

<u>Stack</u>	<u>Input</u>
$\$ \alpha Bxy$	$z \$$

Now the parser reduces y to A.

In both cases, after making a reduction the parser had to shift zero or more symbols to get the next handle onto the stack. It never had to go into the stack to find the handle. It is this aspect of handling pruning that makes as a stack a particularly convenient data structure for implementing a shift reduces parser.