**Bottom-up parser**

A bottom-up parse starts with the string of terminals itself and builds from the

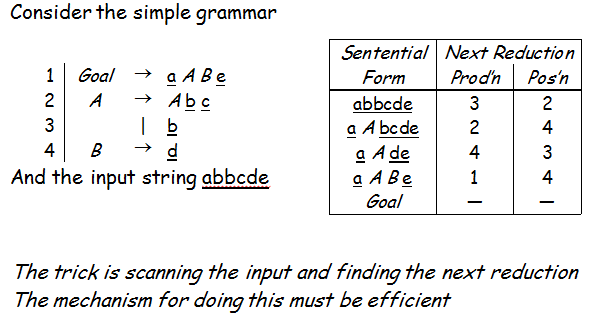
leaves upward, working backwards to the start symbol by applying the productions in reverse. Along the way, a bottom-up parser searches for substrings of the working string that match the right side of some production. When it finds such a substring, it reduces it, i.e., substitutes the left side nonterminal for the matching right side. The goal is to reduce all the way up to the start symbol and report a successful parse.

In general, bottom-up parsing algorithms are more powerful than top-down methods, but not surprisingly, the constructions required are also more complex.

In the bottom-up parsing method, we discuss "shift-reduce" parsing .

**Shift-Reduce Parsing:**

This parsing method is bottom up parsing .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 a production by the symbol on the left .



A bottom-up parser builds a derivation by working from the input sentence back toward the start symbol *S*

*S* ⇒ γ0 ⇒ γ1 ⇒ γ2 ⇒ … ⇒ γn–1 ⇒ γn ⇒ *sentence*

To reduce γi to γi–1 match some *rhs* β against γi then replace β with its corresponding *lhs, A*. *(assuming the production A→β)*

In terms of the parse tree, this is working from leaves to root

* Nodes with no parent in a partial tree form its *upper fringe*
* Since each replacement of β with *A* shrinks the upper fringe,

we call it a *reduction*.

**Handels:**

A handle of a sentential form is a substring α such that :

– a matches the RHS of a production A -> α ; and

– replacing α by the LHS A represents a step in the reverse of a rightmost derivation of s.

**Example** Consider this grammar

E→E+E

E→E\*E

E→(E)

E→id

And consider the rightmost derivation of the input id+id\*id

E→E+E

→E+E\*E

→E+E\*id

→E+id\*id

→id+id\*id

**Handle Pruning:**

The process of discovering a handle & reducing it to the appropriate left-hand side is called *handle pruning*. Handle pruning forms the basis for a bottom-up parsing method.

A rightmost derivation in reverse , often called canonical reduction sequence , is obtained by starting with a string of terminals w which we wish to parse . If w is a sentence of the grammar at hand , then w=γnis the nth right-sentential form for the right-most derivation

S→γ0→γ1→γ2→…→γn-1 →γn =w

To reconstruct this derivation in reverse order , we locate the handle βn in γn and replace βn by the left side of some production An→ βn to obtain the (n-1)st right sentential form γn-1 .We repeat this process , that is locate the handle βn-i in γn-i and reduce this handle to obtain a right sentential form γn-i-1 until we produce γ1 =S .

**Example**

Right sentential form Handle Reduction production

id+id\*id id -

E+id\*id id E→id

E+E\*id id E→id

E+E\*E E\*E E→E\*E

E+E E+E E→E+E

**Stack Implementation of Shift-Reduce Parsing**

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

Stack Input

$ w$

The parser operates by shifting zero or more input onto the stack until a handle β is on top of the stack. The parser then reduce β to the left side of the appropriate production . The parser repeat 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 announces a successful completion of parsing .

**Example**

Stack Input Action

$ id1+id2\*id3$ shift

$id1 +id2\*id3$ reduce by E→id

$E +id2\*id3$ shift

$E+ id2\*id3$ shift

$E+id2 \*id3$ reduce by E→id

$E+E \*id3$ shift

$E+E id3$ shift

$E+E\*id3 $ reduce by E→id

$E+E\*E $ reduce by E→E\*E

$E+E $ reduce by E→E+E

$E $ accept

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

1. In a shift action, the next input symbol is shifted on the top of the stack.
2. In a reduction action, the parser knows the right end of the handle is at the top of the stack. It must locate the left end of the handle within the stack and decide with what nonterminal to replace the handle.
3. In an accept action, the parser announce successful completion of parsing .
4. In an error action , the parser discovers that a syntax error has occurred and calls an error recovery routine .

ambiguous grammars are problematic for bottomup parsers because these grammars could yield more than one handle under some circumstances. These types of grammars create either *shift-reduce* or *reduce-reduce* conflicts. The former refers to a state where the parser cannot decide whether to shift or reduce. The latter refers to a state where the parser has more than one choice of production for reduction. An example of a *shift-reduce* conflict

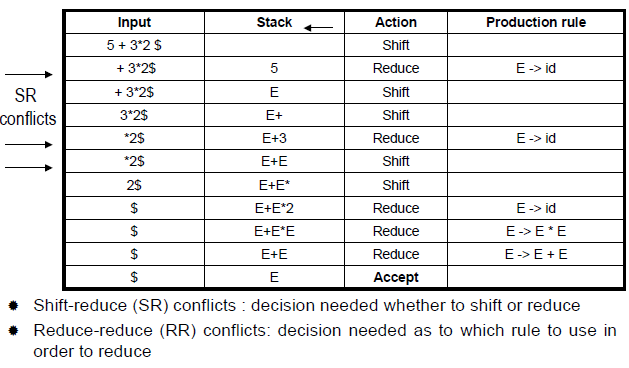
E→E+E

E→E\*E

E→(E)

E→id

id→ 1|2|..|9



Consider this ambiguous grammar:

E → E + E

| int

| (E)

| E \* E

| int \* int + int shift

…… ……

E \* E | + int shift

E \* E + | int shift

E \* E + int | reduce E → int

E \* E + E | reduce E → E + E

E \* E | reduce E → E \* E

E |

􀂄 In the second step E \* E | + int we can either shift or reduce by E → E \* E