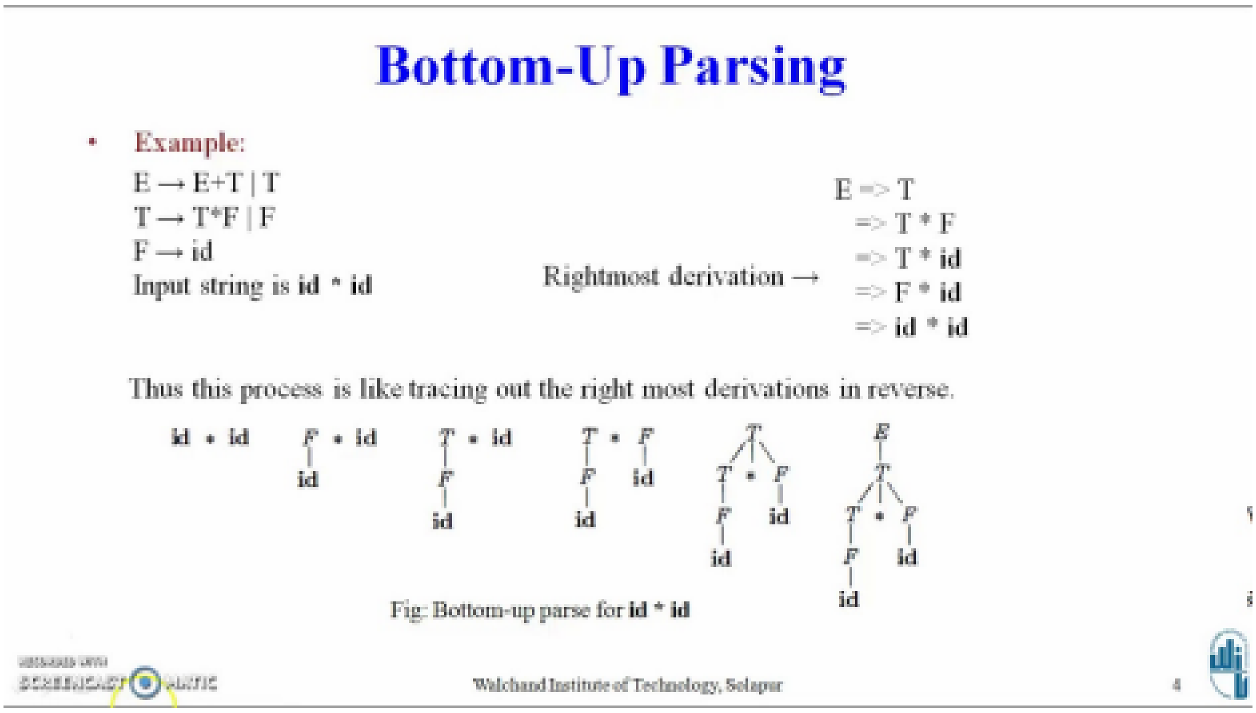
Bottom–up Parsers: Reductions, Handle Pruning, Shift-Reduce Parsing, Conflicts During Shift- Reduce Parsing, LR parsers : SLR, Canonical LR, LALR, Parser Generators : YACC. Syntax Directed translation mechanism and attributed definition.

Intermediate Code Generation: Variants of Syntax Trees, Three-Address Code, Quadruples & Triples, Types and Declarations, Translation of Expressions, Type Checking, Control Flow, Switch- Statements, Intermediate Code for Procedures.

Bottom-Up Parsing:

A bottom-up parse corresponds to the construction of a parse tree for an input string beginning at the leaves (the bottom) and working up towards the root (the top).



Bottom-Up Parsing

- Example:

  E → E+T | T
  T → T*F | F
  F → id
  Input string is id * id

Rightmost derivation →

$$\begin{aligned} E &\Rightarrow T \\ &\Rightarrow T * F \\ &\Rightarrow T * id \\ &\Rightarrow F * id \\ &\Rightarrow id * id \end{aligned}$$

Thus this process is like tracing out the right most derivations in reverse.

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

Fig: Bottom-up parse for **id * id**

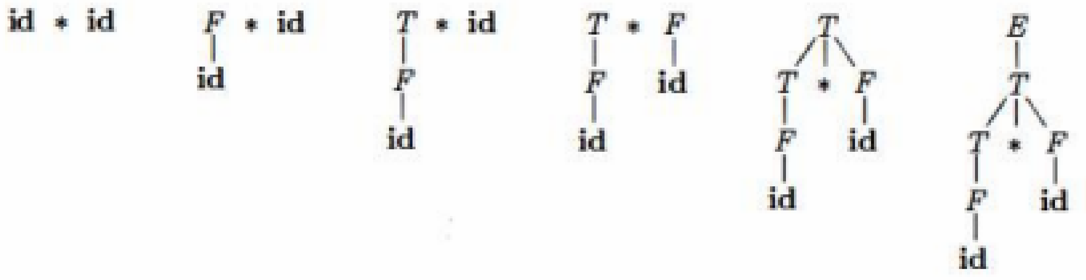


Figure 4.25: A bottom-up parse for **id** * **id**

Shift-reduce parsing is a popular bottom-up technique.

Reductions:

- We can think of bottom-up parsing as the process of "reducing" a string w to the start symbol of the grammar.

- At each reduction step, a specific substring matching the ***body of a production*** is replaced by the nonterminal at the ***head of that production***.

- The key decisions during bottom-up parsing are about ***when to reduce and about what production to apply***, as the parse proceeds

- By definition, ***a reduction is the reverse of a step in a derivation.***

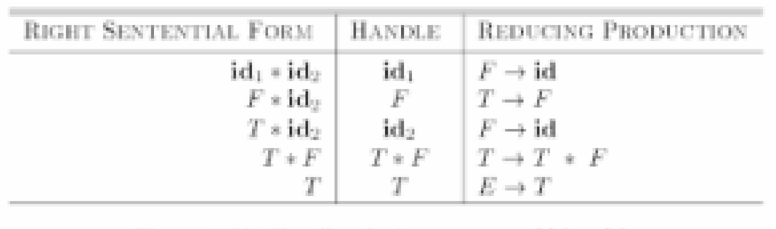
- The goal of bottom-up parsing is therefore to construct a derivation in reverse.

E =>T =>T*F=>T*id=>F*id=>id*id This derivation is infact a rightmost derivation.

Handle Pruning:

HANDLE PRUNING is the general approach used in shift and reduce parsing.

- A “handle” is a substring that matches the body(RHS) of a production.
- Handle reduction is a step in the reverse of rightmost derivation.
- A rightmost derivation in reverse can be obtained by handle pruning.



RIGHT SENTENTIAL FORM	HANDLE	REDUCING PRODUCTION
$\mathbf{id}_1 * \mathbf{id}_2$	$\mathbf{id}_1$	$F \rightarrow \mathbf{id}$
$F * \mathbf{id}_2$	F	$T \rightarrow F$
$T * \mathbf{id}_2$	$\mathbf{id}_2$	$F \rightarrow \mathbf{id}$
$T * F$	$T * F$	$T \rightarrow T * F$
T	T	$E \rightarrow T$

Figure 4.26: Handles during a parse of $\mathbf{id}_1 * \mathbf{id}_2$

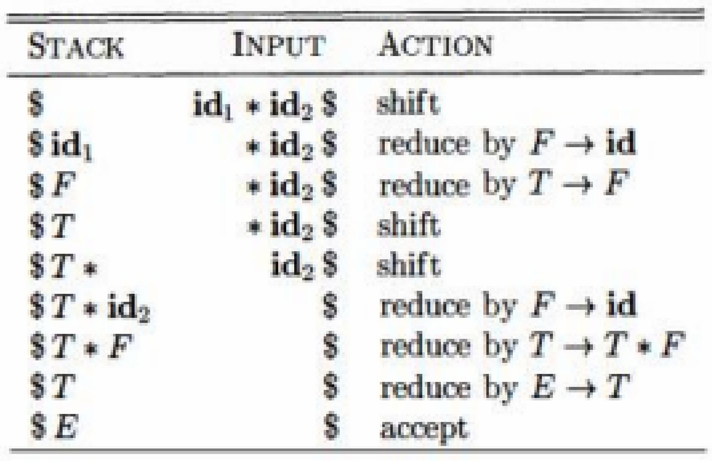
Shift-Reduce Parsing: Shift-reduce parsing is a form of bottom-up parsing in which a stack holds grammar symbols and an input buffer holds the string to be parsed.

The handle always appears at the top of the stack.

We use $ to mark the bottom of the stack and also the right end of the input.

Initially, the stack is empty, and the string w is on the input, as follows:

Stack	Input
$	w$



STACK	INPUT	ACTION
$\$$	$\mathbf{id}_1 * \mathbf{id}_2 \$$	shift
$\$\,\mathbf{id}_1$	$* \mathbf{id}_2 \$$	reduce by $F \rightarrow \mathbf{id}$
$\$\,F$	$* \mathbf{id}_2 \$$	reduce by $T \rightarrow F$
$\$\,T$	$* \mathbf{id}_2 \$$	shift
$\$\,T *$	$\mathbf{id}_2 \$$	shift
$\$\,T * \mathbf{id}_2$	$\$$	reduce by $F \rightarrow \mathbf{id}$
$\$\,T * F$	$\$$	reduce by $T \rightarrow T * F$
$\$\,T$	$\$$	reduce by $E \rightarrow T$
$\$\,E$	$\$$	accept

Figure 4.28: Configurations of a shift-reduce parser on input $\mathbf{id}_1 * \mathbf{id}_2$

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

1. Shift: Shift the next input symbol onto the top of the stack.

2. **Reduce**: The right end of the string to be reduced must be at the top of the stack. Locate the left end of the string within the stack and decide with what nonterminal to replace the string.

3. **Accept**: Announce successful completion of parsing.

4. **Error**: Discover a syntax error and call an error recovery routine.

Conflicts During Shift-Reduce Parsing:

There are context-free grammars for which shift reduce parsing cannot be used.

For such a grammar can reach a configuration in which the shift-reduce parser, knowing the entire stack contents and the next input symbol, cannot decide whether to shift or to reduce (a shift/reduce conflict), or cannot decide which of several reductions to make (a reduce/reduce conflict).

Ex:

Grammar G:

Production rules:

S -> aABe

A -> Abc | b

B-> d

Input String (W): “abbcde”