***Compilers***

***Principle ,Techniques, and Tools***

***Anbar University – Computer College***

**Bottom-Up Parsing**

The term "Bottom-Up Parsing" refer to the order in which nodes in the parse tree are constructed, construction starts at the leaves and proceeds towards the root. Bottom-Up Parsing can handle a large class of grammars.

1. **Shift-Reduce Parsing:** Is a general style of Bottom-upsyntax analysis , it attempts to construct a parse tree for an input string beginning at leaves and working up towards the root,(reducing a string *w* to the start symbol of grammar).At each reduction step a particular substring matching the right side of production is replaced by the

symbol on the left of that production.

**Example :** consider the grammar

S aABe

A Abc b

B d

And the input is **abbcde**

The implementation Bottom-Up Parsing is

**a b b c d e**

**a A b c d e**

**a A d e**

**a A B e**

**S**

**Accept**

**Handle** : Is a substring that matches the right side of aproduction.

**Stack Implementation of Shift-Reduce Parsing:**

A convenient way to implement a shift-reduce parser is to use a *Stack* to hold a grammar symbols and an input buffer to hold thesting *w* to be parsed. We use $ to mark the bottom of *stack* and also the right end of the input string. There are actually four possible actions:

***With My Best Wishes*** *32* ***Esam***

***Compilers***

***Principle ,Techniques, and Tools***

***Anbar University – Computer College***

1. **Shift :** The next input symbol is Shifted onto the topof *stack*.
2. **Reduce :** Replace the handle with nonterminal.
3. **Accept :** The parser announces successfulcompletion of parsing .
4. **Error :** The parser discovers that syntax error has

occurred and calls an error recovery routine.

**Example:** Consider the following grammar

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **E** | **E+E** | | **E\*E** | **(E)** | **id** | |  |
|  |  | | |  |  |  | | |
|  | And the input string is **id + id \* id,** then the | | | | | | | |
|  | implementation is : | | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  | Stack |  |  | Input Buffer | | |  | Action |
|  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| $ | |  |  |  | id+id\*id$ | | | Shift |
|  | $id |  |  |  | +id\*id$ | | | Reduce: E→id |
|  | $E |  |  |  | +id\*id$ | | | Shift |
|  | $E+ |  |  |  | id\*id$ | | | Shift |
|  | $E+id |  |  |  | \*id$ | | | Reduce: E→id |
|  | $E+E |  |  |  | \*id$ | | | Shift(\*) |
|  | $E+E\* |  |  |  |  | id$ | | Shift |
|  | $E+E\*id |  |  | $ | | |  | Reduce: E→id |
|  | $E+E\*E |  |  | $ | | |  | Reduce: E→E\*E |
|  | $E+E |  |  | $ | | |  | Reduce: E→E+E |
|  | $E |  |  | $ | | |  | Accept |
|  |  |  |  |  |  |  |  |  |



**Conflicts During Shift-Reduce Parsing:**

There are context free grammars for which shift-reduce parsing cannot be used. Ambiguous grammars lead to parsing conflicts. Can fix by rewriting grammar or by making appropriate choice of action during parsing. There are two type of conflicts :

***With My Best Wishes*** *33* ***Esam***

***Compilers***

***Principle ,Techniques, and Tools***

1. **Shift/Reduce** conflicts: should we shift or reduce? (Seeprevious example (\*))
2. **Reduce/Reduce** conflicts: which production should wereduce with? for example:

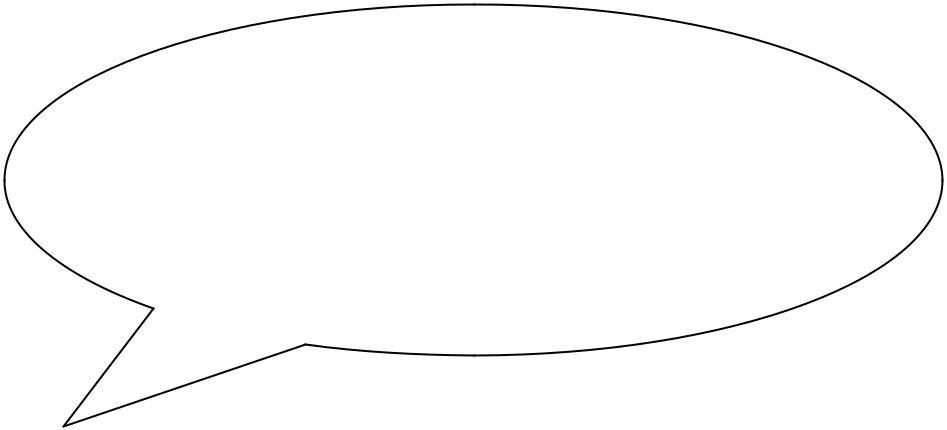
stmt → id(param)

param → id

expr → id(expr) | id

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Stack** | | **Input Buffer** |  | **Action** | |
|  |  |  |  |  |  |
| $...id(id | | ,id)...$ |  | Reduce by ?? | |

Should we reduce to **param** or to **expr** ?



***Example is Better than Precept***