General Idea of Top-down Parsing

Start with the root (i.e., start symbol) of the parse tree

Grow the tree downwards by expanding productions at the lower levels of the  tree

• Select a nonterminal and extend it by adding children corresponding to the right side of some  production for the nonterminal

Repeat till

• Lower fringe **consists only terminals and the input is consumed**

Top-down parsing finds a leftmost derivation for an input string

General Idea of Top-down Parsing

Start with the root of the parse tree

Grow the tree by expanding productions at the lower levels of the tree

• Extend a nonterminal by adding children corresponding to the right side of some  production for the nonterminal

Repeat till

• Lower fringe consists only terminals and the input is consumed

• Mismatch in the lower fringe and the remaining input stream implies i. Wrong choice of productions while expanding nonterminals, selection of a  production may involve trial-and-error

ii.Input character stream is not part of the language

Implementing Backtracking

• A large subset of CFGs can be parsed without backtracking • The grammar may require transformations

• Steps in backtracking

• Set curr to parent and delete the children

• Expand the node curr with **untried rules** if any

• Create child nodes for each symbol in the right hand of the production • Push those symbols onto the stack in reverse order

• Set curr to the first child node

• **Move** curr **up the tree** if there are no untried rules • Report a syntax error when there are no more

Cost of Backtracking

Backtracking is expensive

• Parser expands a nonterminal with the wrong rule

• Mismatch between the lower fringe of the parse tree and the input is detected • Parser undoes the last few actions

• Parser tries other productions if any

A top-down parser can loop indefinitely with left-recursive

Left Recursion

• A grammar is left-recursive if it has a nonterminal such that there is  a derivation for some string

• **Direct** left recursion: There is a production of the form

**Indirect** left recursion: First symbol on the right-hand side of a rule can derive  the symbol on the left

Avoid Backtracking

• Parser is to select the next rule

• Compare the curr symbol and the next input symbol called the lookahead • Use the lookahead to disambiguate the possible production rules

• Backtrack-free grammar is a CFG for which a leftmost, top-down  parser can always predict the correct rule with one word lookahead  • Also called a predictive grammar

FIRST Set

• **Intuition**

• Each alternative for the leftmost nonterminal leads to a **distinct** terminal  symbol

• Which rule to choose becomes obvious by comparing the **next word** in the  input stream

• Given a string of terminal and nonterminal symbols, FIRST*)* is the  set of all terminal symbols that can begin any string derived from • We also need to keep track of which symbols can produce the empty string •

Pseudocode for a Predictive Parser

void stmt() {

switch(lookahead) {

case expr:

match(expr); match(‘;’); break;

case if:

match(if); match(‘(‘); match(expr); match(‘)’); stmt(); break; case for:

match(for); match(‘(‘); optexpr(); match(‘;’); optexpr();  match(‘;’); optexpr(); match(‘)’); stmt(); break;

case other:

match(other); break;

default:

report(“syntax error”);

}

}

LL(1) Grammars

• Class of grammars for which no backtracking is required • First L stands for left-to-right scan, second L stands for leftmost derivation • There is one lookahead token

• In LL(k), k stands for k lookahead tokens

• Predictive parsers accept LL(k) grammars

• Every LL(1) grammar is a LL(2)

CS 335 Swarnendu Biswas

Predictive Parsing

• Grammars whose predictive parsing tables contain no duplicate  entries are called LL(1)

• No left-recursive or ambiguous grammar can be LL(1)

• If grammar is left-recursive or is ambiguous, then parsing table will have at least one multiply-defined cell

• Some grammars cannot be transformed into LL(1)

Error Recovery in Predictive Parsing

• Error conditions

• Terminal on top of the stack does not match the next input symbol  • Nonterminal is on top of the stack, is the next input symbol, and is error

• Choices

i. Raise an error and quit parsing

ii. Print an error message, try to recover from the error, and continue with  compilation

\Error Recovery in Predictive Parsing

• Panic mode – skip over symbols until a token in a set of synchronizing  (synch) tokens appears

• Add all tokens in FOLLOW(

) to the synch set for, parsing can continue if  the parser sees an input symbol in FOLLOW()

• Add symbols in FIRST() to the synch set for , parsing can continue with  the nonterminal that is at the top of the stack

• Add keywords that can begin constructs

• …

• Other error handling policies

• Skip input if the table does not have an entry

• Pop nonterminal if the table entry is synch