

Grammars and Parsing

Context Free Grammar (CFG)

- CFG is a 4-tuple (V_t, V_n, S, P) where:
 - A finite *terminal vocabulary* V_t
 - The token produced by the scanner
 - A finite set of *nonterminal symbols*, V_n
 - A *start symbol* $S \in V_n$ that starts all derivations
 - P , a finite set of *productions* of the form $A \rightarrow X_1 \dots X_m$ where
 - $A \in V_n, X_i \in V_t \cup V_n, 1 \leq i \leq m, m \geq 0$
 - Note that $A \rightarrow \epsilon$ is a valid production

CS 5300 - SJAllan

2

Context Free Language

- Starting with S , nonterminals are rewritten using productions until only terminals remain
- The set of string derivable from S comprises the *context-free language* of grammar G
 - Denoted $L(G)$
- If $A \rightarrow \gamma$ is a production, then $\alpha A \beta \Rightarrow \alpha \gamma \beta$, is a *one-step derivation*
 - \Rightarrow^* - derived in zero or more steps
 - \Rightarrow^+ - derived in one or more steps

CS 5300 - SJAllan

3

Sentential Form

- $S \Rightarrow^* \beta$, the is said to be a sentential form
- $SF(G)$ is the set of sentential forms for grammar G
- $L(G) = \{ x \in V_t^* \mid S \Rightarrow^+ x \}$
- $L(G) = SF(G) \cap V_t^*$

CS 5300 - SJAllan

4

Derivations

- Choice
 - Leftmost – leftmost nonterminal always replaced
 - Rightmost – rightmost nonterminal always replaced
- Consider the grammar G_1 with the productions as follows (ignoring superscripts):
 - $E \rightarrow E + T^1 \mid T^2$
 - $T \rightarrow T * F^3 \mid F^4$
 - $F \rightarrow (E)^5 \mid a^6$

CS 5300 - SJAllan

5

Derivations

- Leftmost derivation of $(a+a)*a$
 - $E \Rightarrow T \Rightarrow T * F \Rightarrow F * F \Rightarrow (E) * F \Rightarrow (E+T) * F \Rightarrow (T+T) * F \Rightarrow (F+T) * F \Rightarrow (a+T) * F \Rightarrow (a+F) * F \Rightarrow (a+a) * F \Rightarrow (a+a) * a$
- Rightmost derivation of $(a+a)*a$
 - $E \Rightarrow T \Rightarrow T * F \Rightarrow T * a \Rightarrow F * a \Rightarrow (E) * a \Rightarrow (E+T) * a \Rightarrow (E+F) * a \Rightarrow (E+a) * a \Rightarrow (T+a) * a \Rightarrow (F+a) * a \Rightarrow (a+a) * a$

CS 5300 - SJAllan

6

Other Types of CFGs

- *Regular grammar*
 - A CFG that are limited to productions of the form $A \rightarrow aB$ and $C \rightarrow \epsilon$
- *Context-sensitive grammar*
 - A CFG that has productions of the form $\alpha A \beta \rightarrow \alpha \delta \beta$

CS 5300 - SJAllan

7

Errors in CFGs

- Useless nonterminals
 - Cannot be reached from the start symbol
 - Cannot derive a terminal string
- Ambiguous
 - Grammars that have more than one derivation tree
 - Consider
 - $E \rightarrow E + E$
 - $E \rightarrow a$
- Generates the “wrong language”

CS 5300 - SJAllan

8

Parsing

- Processing of finding if an input string is a valid element of some language
- Two types of parsers:
 - Top-down
 - Bottom-up

CS 5300 - SJAllan

9

Top-Down Parsing

- Given the grammar G_i :
 - Start with start symbol and try and derive the input string
 - Replace left-hand-sides with right-hand-sides
 - Parse is: $E \Rightarrow^2 T \Rightarrow^3 T * F \Rightarrow^4 F * F \Rightarrow^5 (E) * F \Rightarrow^1 (E + T) * F \Rightarrow^2 (T + T) * F \Rightarrow^4 (F + T) * F \Rightarrow^6 (a + T) * F \Rightarrow^4 (a + F) * F \Rightarrow^6 (a + a) * F \Rightarrow^6 (a + a) * a$
 - Thus $(a + a) * a \in L(G_i)$
 - The parse is: 2 3 4 5 1 2 4 6 4 6 6
 - Look at parse tree

CS 5300 - SJAllan

10

Bottom-up Parsing

- Start with input string and try and derive the start symbol using the left-most nonterminal
- Find right-hand-sides and replace them with left-hand-sides
- Parse is: $(a + a) * a \xleftarrow{2} (F + a) * a \xleftarrow{3} (T + a) * a \xleftarrow{6} (E + a) * a \xleftarrow{4} (E + F) * a \xleftarrow{5} (E + T) * a \xleftarrow{1} (E) * a \xleftarrow{4} F * a \xleftarrow{6} T * a \xleftarrow{2} T * F \xleftarrow{4} T \xleftarrow{6} E$
- The parse is: 6 4 2 6 4 1 5 4 6 3 2
- Look at parse tree

CS 5300 - SJAllan

11

Bottom-up Parsing

- Let's do the parse starting with the start symbol but replacing right-most nonterminal: $E \Rightarrow^2 T \Rightarrow^3 T * F \Rightarrow^6 T * a \Rightarrow^4 F * a \Rightarrow^5 (E) * a \Rightarrow^1 (E + T) * a \Rightarrow^4 (E + F) * a \Rightarrow^6 (E + a) * a \Rightarrow^2 (T + a) * a \Rightarrow^4 (F + a) * a \Rightarrow^6 (a + a) * a$
- The parse is 2 3 6 4 5 1 4 6 2 4 6
- Notice, this is just a bottom up parse reversed

CS 5300 - SJAllan

12

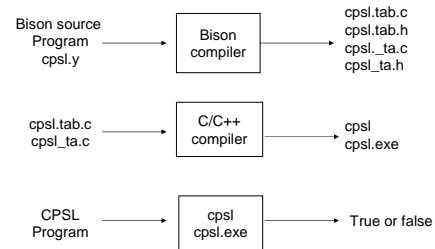
Concepts of Parsing

- Types of parsers:
 - Top-down
 - Recursive descent
 - LL(1)
 - Bottom-up
 - Precedence parsers
 - LR(0)
 - SLR(1)
 - LALR(1)
 - LR(1)
- Most parser generators are LALR(1)

Essentially equivalent

More powerful

Creating a Parser Using Bison



Bison

- Bison produces a function named yyparse that returns true or false
- Bison is executed using the command similar to `bison -vd cpsl.y`
- File produced:
 - `cpsl.tab.c` (`cpsl_ta.c`)
 - Contains the actual parser that calls the lexical analyzer
 - This file needs to be compiled
 - `cpsl.tab.h` (`cpsl_ta.h`)
 - Contains the definition of the tokens
 - This file is input into the lexical analyzer
 - `cpsl.output` (`cpsl.out`)
 - Tells what the parser did
 - If there are conflicts, the file helps you hunt them down

Bison Input File

```

%%      { definitions }
%%      { rules }
%%      { programs }
  
```

Definitions

- Included code as seen in flex
- `%token [<union field>]` terminal
 - As many lines as needed to define the tokens
- `%start` nonterminal
 - Optional line specifying the start symbol
- `%left` nonterminals, `%right` nonterminals, and/or `%nonassoc` nonterminals
 - Used to specify precedence and associativity of nonterminal symbols
 - Precedence is by order, lowest precedence coming first and highest coming last
- `%union { fields }`
 - Use to specify the different types that may be pushed on the semantic stack
- `%type [<union field>]` nonterminals
 - Used to tell the semantic type of the nonterminals

Rules

- Contains the productions for the context free grammar
 - The following notation is used:
 - `:` replaces \rightarrow
 - `|` is used for alternative productions
 - `;` is used to end a production
 - Each production can have an action associated with it
 - The action is just C/C++ code

Conflicts

- Shift-reduce
 - Resolved by shifting
 - Can be acceptable in the grammar
- Reduce-reduce
 - Resolved by using rule that appears first
 - Always need to be resolved

CS 5300 - SJAllan

19

Ambiguous Grammar

- You are required to use an ambiguous grammar for expressions in your project

CS 5300 - SJAllan

20

Example Bison File

```
%{
%}
%token LPAREN$Y RPAREN$Y IDENT$Y ASSIGN$Y REPEAT$Y UNTIL$Y
%token SEMICOLON$Y MULT$Y DIV$Y L$Y PLUS$Y MINUS$Y
%token LBRACKET$Y RBRACKET$Y COMMA$Y PERIOD$Y
%union
{
  int int_val;
  char *name_ptr;
}
%nonassoc L$Y
%left PLUS$Y MINUS$Y
%left MULT$Y DIV$Y
%right UNMINUS$Y
%*
Statement : Variable ASSIGN$Y Expression
          | REPEAT$Y StatementList UNTIL$Y Expression
StatementList : Statement
             | StatementList SEMICOLON$Y Statement
Variable : IDENT$Y
         | IDENT$Y LBRACKET$Y ExpressionList RBRACKET$Y
         | IDENT$Y PERIOD$Y IDENT$Y
Expression : Expression PLUS$Y Expression
          | Expression MINUS$Y Expression
          | Expression MULT$Y Expression
          | Expression DIV$Y Expression
          | MINUS$Y Expression %prec UNMINUS$Y
          | LPAREN$Y Expression RPAREN$Y
          | Expression L$Y Expression
          | Variable
ExpressionList : ExpressionList COMMA$Y Expression
              | Expression
```

CS 5300 - SJAllan

21