CMSC 141 Automata and Language Theory

Context-Free Languages

Mark Froilan B. Tandoc

October 17, 2014

Closure Properties for CFL

Like regular languages, CFLs are close under union, concatenation, and Kleene star

Closure Under Union

Proof

Given two grammars for two context-free languages, with start symbols S and T. Rename the variables to ensure that the two grammars will not share any variable. Then construct a grammar for the union of the two languages by taking all the rules of both grammars and adding a new start state Z with rules $Z \to S \mid T$

Example

$L_1 \Rightarrow$

$$S \to aSb \mid \varepsilon$$

$L_2 \Rightarrow$

$$T \to aTa \mid bTb \mid a \mid b \mid \varepsilon$$

$L_1 \cup L_2 \Rightarrow$

$$\begin{aligned} Z &\to S \mid T \\ S &\to aSb \mid \varepsilon \\ T &\to aTa \mid bTb \mid a \mid b \mid \varepsilon \end{aligned}$$

Closure Under Concatenation

Proof

Same process as union, but instead, we have the rule for the start state $Z \rightarrow ST$

Example

$L_1 \Rightarrow$

$$S \rightarrow aSb \mid \varepsilon$$

$L_2 \Rightarrow$

$$T \rightarrow aTa \mid bTb \mid a \mid b \mid \varepsilon$$

$L_1 L_2 \Rightarrow$

$$\begin{aligned} Z &\rightarrow ST \\ S &\rightarrow aSb \mid \varepsilon \\ T &\rightarrow aTa \mid bTb \mid a \mid b \mid \varepsilon \end{aligned}$$

Closure Under Kleene Star

Proof
Given a grammar for a context-free language L with start symbol S, the grammar for L^*, with start symbol Z, contains all the rules of the original grammar along with the rules $Z \to ZS \mid \varepsilon$

Example

$L \Rightarrow$
$$S \to aSb \mid \varepsilon$$

$L^* \Rightarrow$
$$\begin{aligned} Z &\to ZS \mid \varepsilon \\ S &\to aSb \mid \varepsilon \end{aligned}$$

Other Closure Properties

Other closure properties for CFLs

- string reversal
- homomorphism (string substitutions)
- inverse homomorphisms

Proofs are left as exercise

Closure Properties

Also note, however, that CFLs are *not* closed under

- intersection
- set complement

Proofs are left as exercise

Pumping Lemma for CFLs

Pumping Lemma

If A is a context-free language that is infinite, then there is a number p (the pumping length) where, if s is any string in A of length at least p, then s may be divided into five parts $s = uvxyz$ satisfying:

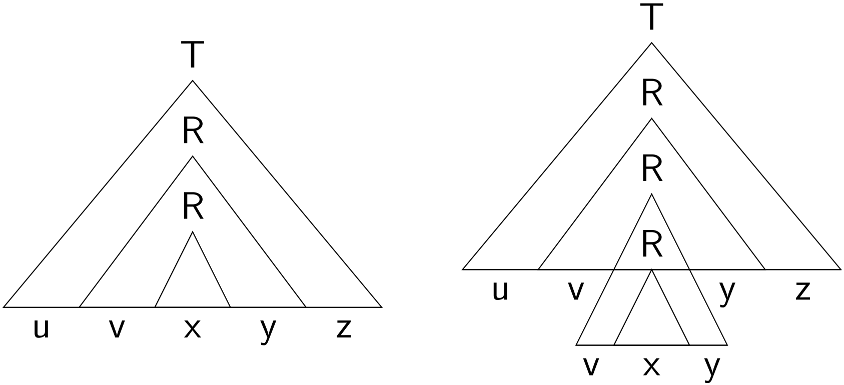
- for each $i \geq 0, uv^ixy^iz \in A$
- $|vy| > 0$ (v and y cannot be both empty)
- $|vxy| \leq p$

Idea of the Proof

- We use the pigeonhole principle on the nodes of the parse tree
- Given a very long string s, its parse tree would be very tall that there must exists some interior node (say R) that must be repeated
  - $R \to^* x \mid vRy$
- "Pumping" translates to expanding R any number of times

Pumping Lemma for CFLs

"Pumping" translates to expanding R any number of times



Non-Context Free Languages

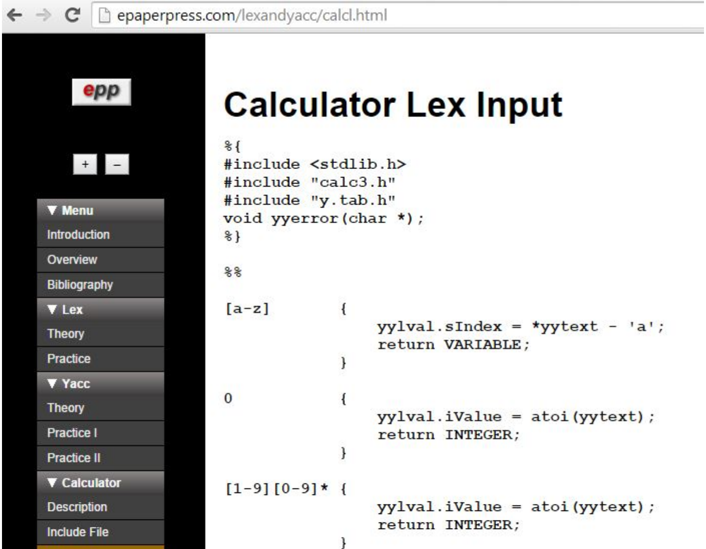
- Some languages cannot be recognized by a PDA or a CFG
- One of the simplest non-CFL is $L = \{a^n b^n c^n : n > 0\} = \{abc, aabbcc, ...\}$
  Can be proven using proof by contradiction and pumping lemma
- How can we still extend PDAs? 2 stacks??

Some applications of CFL

- Compiler Design/Programming Language Design
- Lindenmayer Systems (L-Systems)

Can be used in tools like Lex/Flex and Yacc/Bison

From http://epaperpress.com/lexandyacc



epaperpress.com/lexandyacc/calcl.html

epp

+ –

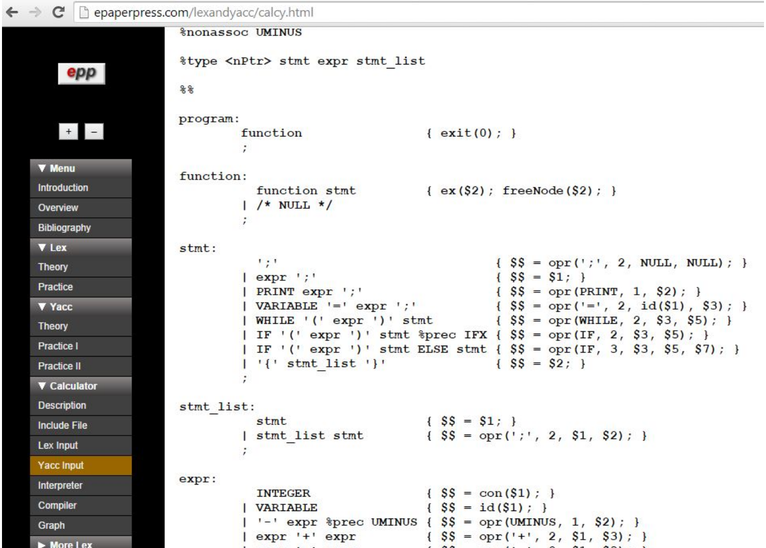
▼ Menu
Introduction
Overview
Bibliography
▼ Lex
Theory
Practice
▼ Yacc
Theory
Practice I
Practice II
▼ Calculator
Description
Include File

Calculator Lex Input

```
%{
#include <stdlib.h>
#include "calc3.h"
#include "y.tab.h"
void yyerror(char *);
%}

%%

[a-z]       {
                yylval.sIndex = *yytext - 'a';
                return VARIABLE;
            }

0           {
                yylval.iValue = atoi(yytext);
                return INTEGER;
            }

[1-9][0-9]* {
                yylval.iValue = atoi(yytext);
                return INTEGER;
            }
```

Can be used in tools like Lex/Flex and Yacc/Bison

From http://epaperpress.com/lexandyacc




```
%nonassoc UMINUS

%type <nPtr> stmt expr stmt_list

%%

program:
        function                { exit(0); }
        ;

function:
          function stmt         { ex($2); freeNode($2); }
        | /* NULL */
        ;

stmt:
          ';'                            { $$ = opr(';', 2, NULL, NULL); }
        | expr ';'                       { $$ = $1; }
        | PRINT expr ';'                 { $$ = opr(PRINT, 1, $2); }
        | VARIABLE '=' expr ';'          { $$ = opr('=', 2, id($1), $3); }
        | WHILE '(' expr ')' stmt        { $$ = opr(WHILE, 2, $3, $5); }
        | IF '(' expr ')' stmt %prec IFX { $$ = opr(IF, 2, $3, $5); }
        | IF '(' expr ')' stmt ELSE stmt { $$ = opr(IF, 3, $3, $5, $7); }
        | '{' stmt_list '}'              { $$ = $2; }
        ;

stmt_list:
          stmt                  { $$ = $1; }
        | stmt_list stmt        { $$ = opr(';', 2, $1, $2); }
        ;

expr:
          INTEGER               { $$ = con($1); }
        | VARIABLE              { $$ = id($1); }
        | '-' expr %prec UMINUS { $$ = opr(UMINUS, 1, $2); }
        | expr '+' expr         { $$ = opr('+', 2, $1, $3); }
```

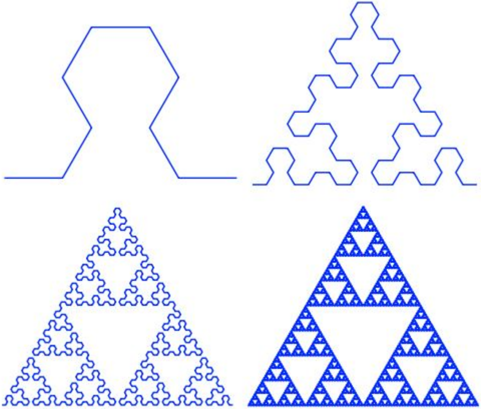
Lindenmayer Systems

grammar-like structures for drawing fractals

Example 5: Sierpinski triangle [edit]

The Sierpinski triangle drawn using an L-system.

- **variables** : A B
- **constants** : + −
- **start** : A
- **rules** : (A → B−A−B), (B → A+B+A)
- **angle** : 60°



Evolution for $n = 2$, $n = 4$, $n = 6$, $n = 8$

From Wikipedia

References

- Previous slides on CMSC 141
- M. Sipser. Introduction to the Theory of Computation. Thomson, 2007.
- J.E. Hopcroft, R. Motwani and J.D. Ullman. Introduction to Automata Theory, Languages and Computation. 2nd ed, Addison-Wesley, 2001.
- E.A. Albacea. Automata, Formal Languages and Computations, UPLB Foundation, Inc. 2005
- JFLAP, www.jflap.org
- Various online LaTeX and Beamer tutorials