CFLs and Regular Languages

- We can show that every RL is also a CFL
  - Since a regular grammar is certainly context free.
- We can also show by only using Regular Expressions and Context Free Grammars
  - That is what we will do in this half.
- Note: Much of this lecture is not in the text!

Union, Concatenation, and Kleene Star of CFLs

- Formally, Let $L_1$ and $L_2$ be CFLs. Then there exists CFGs:
  - $G_1 = (V_1, T, S_1, P_1)$
  - $G_2 = (V_2, T, S_2, P_2)$ such that
    - $L(G_1) = L_1$ and $L(G_2) = L_2$
    - Assume that $V_1 \cap V_2 = \emptyset$
  - We will define:
    - $G_u = (V_u, T, S_u, P_u)$ such that $L(G_u) = L_1 \cup L_2$
    - $G_c = (V_c, T, S_c, P_c)$ such that $L(G_c) = L_1 L_2$
    - $G_k = (V_k, T, S_k, P_k)$ such that $L(G_k) = L_1^*$

Union

- Basic Idea
  - Define the new CFG so that we can either
    - start with the start variable of $G_1$ and follow the production rules of $G_1$
    - start with the start variable of $G_2$ and follow the production rules of $G_2$
  - The first case will derive a string in $L_1$
  - The second case will derive a string in $L_2$
Union, Concatenation, and Kleene Star of CFLs

Concatenation

General Idea
- Define the new CFG so that
  - We force a derivation staring from the start variable of $G_1$ using the rules of $G_1$
  - After that...
  - We force a derivation staring from the start variable of $G_2$ using the rules of $G_2$.

Formally
- $G_c = (V_c, T, S_c, P_c)$
- $V_c = V_1 \cup V_2 \cup \{S_c\}$
- $S_c = S_c$
- $P_c = P_1 \cup P_2 \cup \{S_c \rightarrow S_1S_2\}$

Kleene Star

General Idea
- Define the new CFG so that
  - We can repeatedly concatenate derivations of strings in $L_1$
  - Since $L_1^*$ contains $\lambda$, we must be careful to assure that there are productions in our new CFG such that $\lambda$ can be derived from the start variable.

Formally
- $G_k = (V_k, T, S_k, P_k)$
- $V_k = V_1 \cup \{S_k\}$
- $S_k = S_k$
- $P_k = P_1 \cup \{S_k \rightarrow S_1S_k \mid \lambda\}$

CFLs and Regular Languages

Now we can complete the proof
- Use an inductive proof

Regular Expression

Recursive definition of regular languages / expression over $\Sigma$
- $\emptyset$ is a regular language and its regular expression is $\emptyset$
- $\{\lambda\}$ is a regular language and $\lambda$ is its regular expression
- For each $a \in \Sigma$, $\{a\}$ is a regular language and its regular expression is $a$
Regular Expression

4. If $L_1$ and $L_2$ are regular languages with regular expressions $r_1$ and $r_2$ then
   - $L_1 \cup L_2$ is a regular language with regular expression $(r_1 + r_2)$
   - $L_1L_2$ is a regular language with regular expression $(r_1r_2)$
   - $L_1^*$ is a regular language with regular expression $(r_1)^*$

Only languages obtainable by using rules 1-4 are regular languages.

CFLs and Regular Languages

- RE $\rightarrow$ CFG
  - Base cases
    1. $\emptyset$ can be expressed as a CFG with no productions
    2. $\{\lambda\}$ can be expressed by a CFG with the single production $S \rightarrow \lambda$
    3. For each $a \in \Sigma$, $\{a\}$ can be expressed by a CFG with the single production $S \rightarrow a$

Union, Concatenation, and Kleene Star of CFLs

- RE $\rightarrow$ CFG
  - Assume $R_1$ and $R_2$ are regular expressions that describe languages $L_1$ and $L_2$. Then, by the induction hypothesis, $L_1$ and $L_2$ are CFLs and as such there are CFGs that describe $L_1$ and $L_2$.
  - Create CFGs that describe the the languages:
    - $L_1 \cup L_2$
    - $L_1L_2$
    - $L_1^*$
  - Which we just did...We are done!

CFLs and Regular Languages

- What have we learned?
  - CFLs are closed under union, concatenation, and Kleene Star
  - Every Regular Language is also a CFL
  - We now have an algorithm, given a Regular Expression, to construct a CGF that describes the same language

Example

- Find a CFG for the $L = (011 + 1)^*(01)^*$
  - $(011 + 1)$ can be described by the CFG with productions:
    - $A \rightarrow 011 \mid 1$
  - $(011 + 1)^*$ can be described by the CFG with productions:
    - $B \rightarrow AB \mid \lambda$
    - $A \rightarrow 011 \mid 1$
Example

Find a CFG for the \( L = (011 + 1)(01)^* \)

- \((01)\) can be described by the CFG with productions:
  - \( D \rightarrow 01 \)

- \((01)^*\) can be described by the CFG with productions:
  - \( C \rightarrow DC | \lambda \)
  - \( D \rightarrow 01 \)

Putting it all together

\((011 + 1)(01)^*\) can be described by the CFG with productions:

- \( S \rightarrow BC \)
- \( B \rightarrow AB | \lambda \)
- \( A \rightarrow 011 | 1 \)
- \( C \rightarrow DC | \lambda \)
- \( D \rightarrow 01 \)

Questions?

Union, Concatenation, and Kleene Star of CFLs

You can use proof of closure properties in building CFLs:

Example:

- Find a CFL for \( L = \{0^i 1^j 0^k \mid j > i + k \} \)
  - Number of 1s is greater than the combined number of 0s
  - This language can be expressed as
    \[ L = \{0^i 1^i 1^m \mid m > 0 \} \]

Union, Concatenation, and Kleene Star of CFLs

Example:

Formally

- \( G = (V, \Sigma, S, P) \) where
  - \( V = \{S, A, B, C\} \)
  - \( \Sigma = \{0, 1\} \)
  - \( P = \{S \rightarrow ABC\} \)
    - \( A \rightarrow 0A1 | \lambda \)
    - \( B \rightarrow 1B | 1 \)
    - \( C \rightarrow 1C0 | \lambda \)
CFLs and Regular Languages

Questions?

Practical uses for grammars

How a compiler works

Stream of tokens

Parse Tree

Object code

Source file

The Bell Labs Gang

Ken Thompson
Regular expressions in UNIX / grep / vi

Eric E. Schmidt

Mike Lesk
lex

Stephen C Johnson
yacc

A real practical example

Grammars for programming languages

Keywords and punctuation are terminals
Program constructs are variables
Production rules define the syntax of the language

This is really the second step in building a compiler!

A real practical example

Grammars for programming languages

<stmt> → ... | <for-stmt> | <if-stmt> | ...
<stmt> → { <stmt> <stmt> } | ε
<if-stmt> → if (<expr>) then <stmt>
<for-stmt> → for(<expr>; <expr>; <expr>) <stmt>

Famous programming language ambiguity

Dangling else

<stmt> → if (<expr>) <stmt> | if (<expr>) <stmt> else <stmt> | ...
<stmt> → if (expr1) if (expr2)f(); else g();
To which if does the else belong?
In this derivation, the else belongs to the 1st if

\[
\text{if } (\text{expr1}) \text{ if } (\text{expr2}) f(); \text{ else } g();
\]

Famous programming language ambiguity

\[
\text{if } (\text{expr1}) \text{ if } (\text{expr2}) f(); \text{ else } g();
\]

Famous programming language ambiguity

\[
\text{if } (\text{expr1}) \text{ if } (\text{expr2}) f(); \text{ else } g();
\]

Famous programming language ambiguity

\[
\text{if } (\text{expr1}) \text{ if } (\text{expr2}) f(); \text{ else } g();
\]

Famous programming language ambiguity

\[
\text{if } (\text{expr1}) \text{ if } (\text{expr2}) f(); \text{ else } g();
\]

Famous programming language ambiguity

\[
\text{if } (\text{expr1}) \text{ if } (\text{expr2}) f(); \text{ else } g();
\]
Summary

- All Regular Languages are CFLs
- Use regular language operations in constructing CFGs
- CFGs in compiler design