Warm-Up Problem

- Write an ε-NFA over $\Sigma = \{a, b, c\}$ that accepts $L = \{ab\} \cup \{ab^n c : n \in \mathbb{N}\}^*$.

CS 241 Lecture 9

Context-Free Grammars

With thanks to Brad Lushman, Troy Vasiga, Kevin Lanctot, and Carmen Bruni

Come Up for Air

Where are we now?

1. Identify tokens (Scanning) [Complete!]
2. Check order of tokens (Syntactic Analysis) [Now]
3. Type Checking (Semantic Analysis) [Later]
4. Code Generation [Also later]

- Syntax: Is the order of the tokens correct? Do parentheses balance?
- Semantics: Does what is written make sense (right type of variables in functions etc.)

Our Motivating Example

- Consider $\Sigma = \{(,)\}$ and $L = \{w : w$ is a balanced string of parentheses$\}$.
- Is this language regular? Can we build a DFA for L?

- Try to convince yourself of why this is impossible: once you have arbitrarily large levels of open parentheses, it is tough to be able to know how many (symbols you have processed, so that you can process the right number of)s, without extra memory.

Balanced Parentheses

Consider this regular attempt:

(*???*) | ε

- The *???* is the problem: what goes inside the parentheses is the entire language of matched parentheses!
- What if we could recurse in regular expressions?

L = (L) | ε

Note: This just covers (((…))), not, e.g., ()()(()).

Context-Free Languages

In terms of power, context-free languages are exactly regular languages plus recursion.

In terms of expression, rather than extend regular expressions, we have a different form, called *grammars*.

Definitions

- *Grammar* is the language of languages (Matt Might).
- In some sense, grammars help us to describe what we are allowed and not allowed to say.
- Context-free grammars are a set of rewrite rules that we can use to describe a language.

Grammar Example

The following is a CFG for C++ compound statements:

- <compound stmt> → { <stmt list> }
- <stmt list> → <stmt> <stmt list> | epsilon
- <stmt> → <compound stmt>
- <stmt> → if (<expr>) <stmt>
- <stmt> → if (<expr>) <stmt> else <stmt>
- <stmt> → while (<expr>) <stmt>
- <stmt> → do <stmt> while (<expr>) ;
- <stmt> → for (<stmt> <expr> ; <expr>) <stmt>
- <stmt> → break ; | continue ;
- <stmt> → return <expr> ; | goto <id> ;
- Source: https://www.cs.rochester.edu/~nelson/courses/csc_173/grammars/cfg.html

No need for Kleene star!

Formal Definition

Definition

A **Context Free Grammar (CFG)** is a 4 tuple (N, Σ, P, S) where

- N is a finite non-empty set of *non-terminal symbols*.
- Σ is an alphabet; a set of non-empty *terminal symbols*.
- P is a finite set of productions, each of the form $A \rightarrow \beta$ where
  $A \in N$ and $\beta \in (N \cup \Sigma)^*$
- $S \in N$ is a starting symbol

Note: We set $V = N \cup \Sigma$ to denote the *vocabulary*, that is, the set of all symbols in our language.

Conventions

- Lower case letters from the start of the alphabet, i.e., a, b, c, ..., are elements of Σ
- Lower case letters from the end of the alphabet, i.e., w, x, y, z, are elements of Σ^* (words)
- Upper-case letters, i.e., A, B, C, ..., are elements of N (non-terminals)
- S is always our start symbol.
- Greek letters, i.e., α, β, γ, ..., are elements of V^* (recall this is $(N \cup \Sigma)^*$)

Stacking

- In most programming languages, the terminals (alphabet) of the context-free language are the *tokens*, which are the *words* in the regular language
- This is why scanners categorize tokens (e.g. all infinity IDs are “ID”): so that the CFL’s alphabet is finite!
- It is possible to define CFGs directly over the input characters: this is called *scannerless*

Example

Let's revisit $\Sigma = \{(,)\}$ and $L = \{w : w \text{ is a balanced string of parentheses}\}$.

- $S \rightarrow \varepsilon$
- $S \rightarrow (S)$
- $S \rightarrow SS$

We can also write this using a shorthand:

- $S \rightarrow \varepsilon \mid (S) \mid SS$

Example

Find a derivation of (()()). Recall our CFG: $S \to \varepsilon \mid (S) \mid SS$

Definition

Over a CFG (N, Σ, P, S), we say that...

- A *derives* γ and we write $A \Rightarrow \gamma$ if and only if there is a rule $A \to \gamma$ in P.
- $\alpha A \beta \Rightarrow \alpha \gamma \beta$ if and only if there is a rule $A \to \gamma$ in P.
- $\alpha \Rightarrow^* \gamma$ if and only if a *derivation* exists, that is, there exists $\delta_i \in V^*$ for $0 \le i \le k$ such that $\alpha = \delta_0 \Rightarrow \delta_1 \Rightarrow \ldots \Rightarrow \delta_k = \gamma$. Note that k can be 0.

Solution:

$S \Rightarrow (S) \Rightarrow (SS) \Rightarrow ((S)S) \Rightarrow (()S) \Rightarrow (()(S)) \Rightarrow (()())$.

Hence $S \Rightarrow^* (()())$.

Aside: Why “Context-Free”?

- Context-free languages actually add a sort of context to regular languages, so why are they “context free”?
- They’re free of a different sort of context. For instance, a context-free language can’t catch this:

```
int a;
(*a)+12;
```

Need the *context* that a is an int to know this isn’t allowed

Final Definitions

Definition

Define the *language of a CFG* (N, Σ, P, S) to be $L(G) = \{w \in \Sigma^* : S \Rightarrow^* w\}$.

Definition

A language is *context-free* if and only if there exists a CFG G such that $L = L(G)$.

Example: Every regular language is context-free! (Why?)

Regular Languages are Context-Free (Very Informally)

1. $\emptyset$: ({S}, {a}, $\emptyset$, S)
2. {ε}: ({S}, {a}, S → ε, S).
3. {a}: ({S}, {a}, S → a, S).
4. Union: {a} ∪ {b}: ({S}, {a, b}, S → a|b, S).
5. Concatenation: {ab}:
   ({S}, {a, b}, S → ab, S).
6. Kleene Star: {a}∗: ({S}, {a}, S → Sa|e, S).

Practice

Let $\Sigma = \{a, b\}$. Find a CFG for each of the following:

- $a(a|b)*b$
- $\{a^n b^n : n \in N\}$
  - Further, find a derivation in your grammar of aaabbb.
- Palindromes over {a, b, c}

Solution to Second

CFG:

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

Derivation:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb.$$

A Fundamental Example

Let's consider arithmetic operations over $\Sigma = \{a, b, c, +, -, *, /, (,)\}$. Find

- A CFG for L_1: arithmetic expressions from Σ without parentheses, and a derivation for
  a - b
- A CFG for L_2: Well-formed arithmetic expressions from Σ with balanced parentheses, and a derivation for ((a) - b)