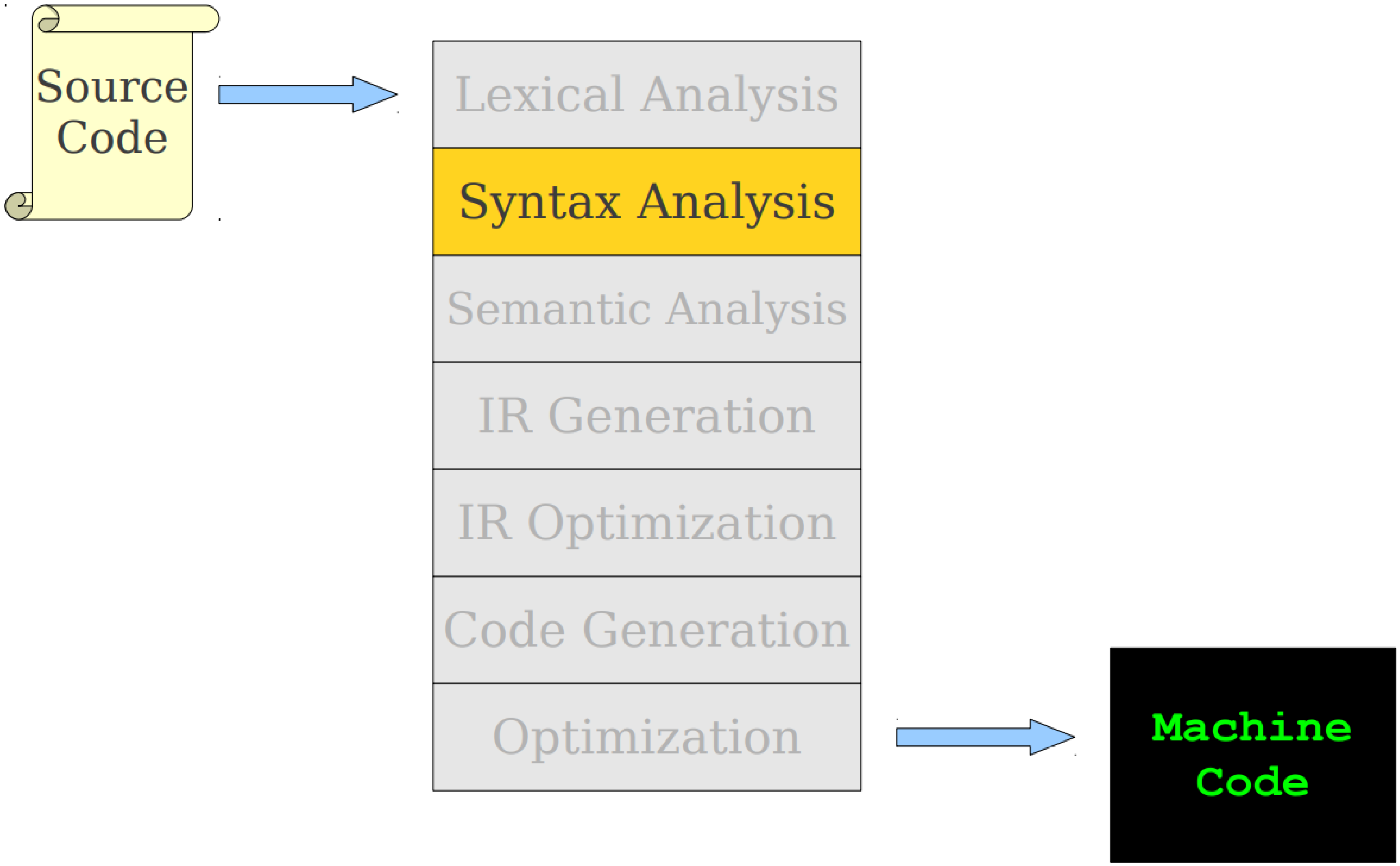
Grammar

Professor: Suman Saha

Where we are?


Source Code
Lexical Analysis
Syntax Analysis
Semantic Analysis
IR Generation
IR Optimization
Code Generation
Optimization
Machine Code

What is Syntax Analysis?

- After lexical analysis (scanning), we have a series of tokens.
- In Syntax analysis (or parsing), we want to interpret what those tokens mean.
- Goal: Recover the *structure* describe by that series of tokens.
- Goal: Report errors if those tokens do not properly encode a structure.

Formal Languages

- An alphabet is a set Σ of symbols that act as letters.
- A language over Σ is a set of strings made from symbols in Σ.
- When scanning, our alphabet was ASCII or Unicode characters. We produced tokens.
- When parsing, our alphabet is the set of tokens produced by the scanner.

The Limits of Regular Languages

- When scanning, we used regular expressions to define each token.
- Unfortunately, regular expressions are (usually) too weak to define programming languages.
  - Cannot define a regular expression matching all expressions with properly balanced parentheses.
  - Cannot define a regular expression matching all functions with properly nested block structure.
- We need a more powerful formalism.

(( )))

((())())

Grammars

- It is written in a metalanguage
- It defines all the legal strings of characters that can form a syntactically valid program

Context-Free Grammars

- Context-Free Grammars
  - Developed by Noam Chomsky in the mid-1950s
  - Describe the syntax of natural languages
  - Define a class of languages called context-free languages
  - Was originally designed for natural languages

Context-Free Grammars

PennState

- Using the notation of Backus-Naur Form (BNF) to describe CFG
- A grammar G <N, T, P, S> consists of the following
  - A finite set N of non-terminal symbols
  - A finite set T of terminal symbols, that is disjoint from N
  - A finite set P of production rules of the form

$$A \to \omega$$

  where ω is a string of nonterminal and terminal

  - Start symbol

Backus-Naur Form (BNF) Grammars

- A rule has a left-hand side (LHS), one or more right-hand side (RHS), and consists of terminal and nonterminal symbols
  - For instance
    - <binaryDigit> → 0
    - <binaryDigit> → 1
  - We can write <binaryDigit> → 0 | 1

$A \rightarrow \omega$

Extended BNF Grammar

- Extended BNF simplifies writing a grammar by introducing metasymbols for iteration; option, and choice
- BNF

$e_1 = e_2 + 3$

$\rightarrow$ <expr> := <expr> + <term>

$e_1 = e_2 - 3$

| <expr> - <term> | <term>

$e_1 = 3$

- EBNF

*

<expr> := <expr> {(+ | -) <term>} | <term>

+|−

$e = e$

$e = e + t$

Extended BNF Grammar

- BNF

<ifStmt> := if (<expr>) <stmt>
| if (<expr>) <stmt> else <stmt>

- EBNF

<ifStmt> := if (<expr>) <stmt> [else <stmt>]

0|1 (?)

Extended BNF Grammar

- However, EBNF is any more powerful than BNF for formally describing language syntax

$$A \rightarrow x\ \{y\}^{*}\ z$$

- Equivalent to

$$S\quad A \rightarrow x\ A'\ z$$
$$A' \rightarrow \varepsilon \mid y\ A'$$

$A \rightarrow xz$

$A \rightarrow xyz$

$A \rightarrow xyyz$

$A \rightarrow xA'z \rightarrow x\varepsilon z \rightarrow xz$

$A \rightarrow xA'z \rightarrow xyA'z \rightarrow xyz$

$A \rightarrow xA'z \rightarrow xyA'z \rightarrow xyyA'z \rightarrow xyyz$

Derivation

- To determine that the given string of symbols belongs to grammar
- A sequence of steps where nonterminals are replaced by the right-hand side of a production is called a derivation.
  - Leftmost derivation
  - Rightmost derivation

- Sentential form vs Sentence
  - A *sentential form* is any string derivable from the start symbol.
  - A *sentence* is a sentential form consisting only of terminals

- If string α derives string ω, we write $\alpha \Rightarrow^* \omega$.

Leftmost Derivation

- Say, we have grammar

N N N

$$\text{Integer} \rightarrow \text{Digit} \mid \text{Integer Digit}$$

$$\text{Digit} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Leftmost Derivation

- Say, we have grammar

$$\text{Integer} \rightarrow \text{Digit} \mid \text{Integer Digit}$$

$$\text{Digit} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

- 352 is an Integer?

Leftmost Derivation

PennState

- Say, we have grammar

$$\text{Integer} \rightarrow \text{Digit} \mid \text{Integer Digit}$$

$$\text{Digit} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

- 352 is an Integer?

$$\text{Integer} \quad \rightarrow \text{Integer Digit}$$

Leftmost Derivation

- Say, we have grammar

$$\text{Integer} \rightarrow \text{Digit} \mid \text{Integer Digit}$$

$$\text{Digit} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

- 352 is an Integer?

$$\begin{aligned}\text{Integer} \quad & \rightarrow \text{Integer Digit} \\ & \rightarrow \text{Integer Digit Digit}\end{aligned}$$

Leftmost Derivation

- Say, we have grammar

$$\text{Integer} \rightarrow \text{Digit} \mid \text{Integer Digit}$$

$$\text{Digit} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

- 352 is an Integer?

$$\begin{aligned} \text{Integer} \quad & \rightarrow \text{Integer Digit} \\ & \rightarrow \text{Integer Digit Digit} \\ & \rightarrow \text{Digit Digit Digit} \end{aligned}$$

Leftmost Derivation

- Say, we have grammar

$$\text{Integer} \rightarrow \text{Digit} \mid \text{Integer Digit}$$
$$\text{Digit} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

- 352 is an Integer?

$$\begin{aligned}\text{Integer} \quad & \rightarrow \text{Integer Digit} \\ & \rightarrow \text{Integer Digit Digit} \\ & \rightarrow \text{Digit Digit Digit} \\ & \rightarrow 3 \text{ Digit Digit}\end{aligned}$$

Leftmost Derivation

- Say, we have grammar

$$\text{Integer} \rightarrow \text{Digit} \mid \text{Integer Digit}$$

$$\text{Digit} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

- 352 is an Integer?

$$\begin{aligned}\text{Integer} \quad &\rightarrow \text{Integer Digit}\\ &\rightarrow \text{Integer Digit Digit}\\ &\rightarrow \text{Digit Digit Digit}\\ &\rightarrow 3\ \text{Digit Digit}\\ &\rightarrow 3\ 5\ \text{Digit}\end{aligned}$$

Leftmost Derivation

- Say, we have grammar

$$\text{Integer} \rightarrow \text{Digit} \mid \text{Integer Digit}$$

$$\text{Digit} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

- 352 is an Integer?

$$\begin{aligned}\text{Integer} \quad & \rightarrow \text{Integer Digit} \\ & \rightarrow \text{Integer Digit Digit} \\ & \rightarrow \text{Digit Digit Digit} \\ & \rightarrow 3 \text{ Digit Digit} \\ & \rightarrow 3\ 5 \text{ Digit} \rightarrow 352\end{aligned}$$

What if I choose

$$\text{Integer} \rightarrow \text{Digit}$$

Leftmost Derivation

- Say, we have grammar

$$Integer \rightarrow Digit \mid Integer\ Digit$$

$$Digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

- 352 is an Integer?

$$\begin{aligned} Integer \quad & \rightarrow Integer\ Digit \\ & \rightarrow Integer\ Digit\ Digit \\ & \rightarrow Digit\ Digit\ Digit \\ & \rightarrow 3\ Digit\ Digit \\ & \rightarrow 3\ 5\ Digit \rightarrow 352 \end{aligned}$$

$$Integer \Rightarrow^* 352$$

$A \Rightarrow^*$

Rightmost Derivation

- Say, we have grammar

$$\text{Integer} \rightarrow \text{Digit} \mid \text{Integer Digit}$$

$$\text{Digit} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Rightmost Derivation

- Say, we have grammar

$$\text{Integer} \rightarrow \text{Digit} \mid \text{Integer Digit}$$

$$\text{Digit} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

- 352 is an Integer?

Rightmost Derivation

- Say, we have grammar

$$\text{Integer} \rightarrow \text{Digit} \mid \text{Integer Digit}$$

$$\text{Digit} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

- 352 is an Integer?

$$\text{Integer} \quad \rightarrow \text{Integer Digit}$$

Rightmost Derivation

- Say, we have grammar

$$\text{Integer} \rightarrow \text{Digit} \mid \text{Integer Digit}$$

$$\text{Digit} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

- 352 is an Integer?

$$\begin{aligned} \text{Integer} \quad & \rightarrow \text{Integer Digit} \\ & \rightarrow \text{Integer } 2 \end{aligned}$$

Rightmost Derivation

- Say, we have grammar

$$\text{Integer} \rightarrow \text{Digit} \mid \text{Integer Digit}$$

$$\text{Digit} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

- 352 is an Integer?

$$\begin{aligned}\text{Integer} \quad & \rightarrow \text{Integer Digit} \\ & \rightarrow \text{Integer } 2 \\ & \rightarrow \text{Integer Digit } 2\end{aligned}$$

Rightmost Derivation

- Say, we have grammar

$$\text{Integer} \rightarrow \text{Digit} \mid \text{Integer Digit}$$

$$\text{Digit} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

- 352 is an Integer?

$$\begin{aligned} \text{Integer} \quad & \rightarrow \text{Integer Digit} \\ & \rightarrow \text{Integer } 2 \\ & \rightarrow \text{Integer } {\color{red}\text{Digit}}\ 2 \\ & \rightarrow \text{Integer } {\color{cyan}5}\ 2 \end{aligned}$$

Rightmost Derivation

- Say, we have grammar

$$\text{Integer} \rightarrow \text{Digit} \mid \text{Integer Digit}$$
$$\text{Digit} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

- 352 is an Integer?

$$\begin{aligned}\text{Integer} \quad & \rightarrow \text{Integer Digit} \\ & \rightarrow \text{Integer } 2 \\ & \rightarrow \text{Integer Digit } 2 \\ & \rightarrow \text{Integer } 5\ 2 \\ & \rightarrow \text{Digit } 5\ 2\end{aligned}$$

Rightmost Derivation

- Say, we have grammar

$$\text{Integer} \rightarrow \text{Digit} \mid \text{Integer Digit}$$
$$\text{Digit} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

- 352 is an Integer?

$$\begin{aligned}
\text{Integer} \quad & \rightarrow \text{Integer Digit} \\
& \rightarrow \text{Integer } 2 \\
& \rightarrow \text{Integer Digit } 2 \\
& \rightarrow \text{Integer } 5\ 2 \\
& \rightarrow {\color{red}\text{Digit}}\ 5\ 2 \\
& \rightarrow {\color{cyan}3}\ 5\ 2
\end{aligned}$$

Rightmost Derivation

- Say, we have grammar

$$\text{Integer} \rightarrow \text{Digit} \mid \text{Integer Digit}$$
$$\text{Digit} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

- 352 is an Integer?

$$\begin{aligned}\text{Integer} \quad & \rightarrow \text{Integer Digit} \\ & \rightarrow \text{Integer } 2 \\ & \rightarrow \text{Integer Digit } 2 \\ & \rightarrow \text{Integer } 5\ 2 \\ & \rightarrow \text{Digit } 5\ 2 \\ & \rightarrow 3\ 5\ 2\end{aligned}$$

$$\text{Integer} \Rightarrow^* 352$$

Top Hat

The Language of a Grammar

- If G is a CFG with alphabet Σ and start symbol S, then the *language of G* is the set

$$L(G) = \{\omega \in \Sigma^* \mid S \Rightarrow^* \omega\}$$

$\Sigma = \{ \quad \}$

- That is, L(G) is the set of strings derivable from the start symbol.

- Note: ω must be in Σ^*, the set of strings made from terminals. String involving nonterminals aren't in the language.

Context-Free Languages

- A language L is called a **context-free language** (or CFL) if there is a CFG *G* such that L = L(G).

From Regexes to CFGs

- CFGs consist purely of production rules of the form $A \rightarrow \omega$. They do not have the regular expression operators * or ∪.
- However, we can convert regular expressions to CFGs as follows:

$$S \rightarrow a^*b$$

aab

b

ab

From Regexes to CFGs

- CFGs consist purely of production rules of the form $A \rightarrow \omega$. They do not have the regular expression operators * or ∪.
- However, we can convert regular expressions to CFGs as follows:

$$S \rightarrow Ab$$
$$A \rightarrow Aa \mid \varepsilon$$

$S \rightarrow Ab$
ε
$\rightarrow b$

$S \rightarrow \underline{A}b$
$\rightarrow \underline{A}ab$
$\rightarrow ab$

$S \rightarrow Ab$
$\rightarrow Aab$
$\rightarrow Aaab$
$\rightarrow aab$

From Regexes to CFGs

- CFGs consist purely of production rules of the form $A \rightarrow \omega$. They do not have the regular expression operators * or ∪.
- However, we can convert regular expressions to CFGs as follows:

$$S \rightarrow \underline{a(b \cup c^*)} \quad [a(b|c)]$$

$\rightarrow ab$

$\rightarrow ac$

$\rightarrow acc$

From Regexes to CFGs

- CFGs consist purely of production rules of the form $A \rightarrow \omega$. They do not have the regular expression operators * or ∪.
- However, we can convert regular expressions to CFGs as follows:

$$S \rightarrow aX$$
$$X \rightarrow b \mid C$$
$$C \rightarrow Cc \mid \varepsilon$$

$S \Rightarrow aX$
$\Rightarrow ab$

$S \Rightarrow aX$
$\Rightarrow aC$
$\Rightarrow aCc$
$\Rightarrow ac$

Regular Languages and CFLs

- **Theorem:** Every regular language is context-free.
- **Proof Idea:** Use the construction from the previous slides to convert a regular expression for L into a CFG for L.
- **Problem Set Exercise:** Instead, show how to convert a DFA/NFA into a CFG

The Language of a Grammar

PennState

- Consider the following CFG G:

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

$$S \rightarrow aSb \rightarrow aaSbb \rightarrow aaaSbbb \rightarrow aaabbb$$

- What strings can this generate?

a	a	a	a	b	b	b	b

$$L(G) = \{\, a^n b^n \mid n \in \mathbb{N} \,\}$$

Designing CFGs

- Like designing DFAs, NFAs, and regular expressions, designing CFGs is a craft.
- When thinking about CFGs:
  - Think recursively: Build up bigger structures from smaller ones.
  - Have a construction plan: Know in what order you will build up the string.
  - Store information in nonterminals: Have each nonterminal correspond to some useful piece of information.

Designing CFGs

- Let $\Sigma = \{a, b\}$ and let $L = \{w \in \Sigma^* \mid w \text{ is a palindrome}\}$
- We can design a CFG for L by thinking inductively:
- Base case: ε, a, and b are palindromes.
- If ω is a palindrome, then $a\omega a$ and $b\omega b$ are palindromes.

$$S \rightarrow \varepsilon \mid a \mid b \mid aSa \mid bSb$$

a b a a b a

$S \rightarrow aSa$
$\Rightarrow abSba$
$\Rightarrow abaSaba$
$\Rightarrow abaaba$

a
b
ε

aSa
aa

Designing CFGs

- Let $\Sigma = \{ (,) \}$ and let $L = \{w \in \Sigma^* \mid w \text{ is a string of balanced parentheses} \}$
- Some sample string in L

((()))

(())()

(()())(()())

((((()))(())))

ε

()()

Designing CFGs

- Let $\Sigma = \{\ (,\ )\ \}$ and let $L = \{w \in \Sigma^* \mid \text{w is a string of balanced parentheses}\}$
- Let's think about this recursively.
  - Base case: the empty string is a string of balanced parentheses.
  - Recursive step: Look at the closing parenthesis that matches the first open parenthesis.

((()(()))(()))(())((()))

Designing CFGs

PennState

- Let $\Sigma = \{\ (,\ )\ \}$ and let $L = \{w \in \Sigma^* \mid w \text{ is a string of balanced parentheses}\}$
- Let's think about this recursively.
  - Base case: the empty string is a string of balanced parentheses.
  - Recursive step: Look at the closing parenthesis that matches the first open parenthesis. Removing the first parenthesis and the matching parenthesis forms two new strings of balanced parentheses.

$$S \to (S)S \mid \varepsilon$$

$S \to (S)S$

$((S)S)S$

$((S)S)(S)S \Rightarrow (())()$

Designing CFGs: A Caveat

- Let $\Sigma = \{ a, b \}$ and let $L = \{w \in \Sigma^* \mid w$ has the same number of a's and b's $\}$
- Is this a CFG for L?

$S \Rightarrow aSa$

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

$S \Rightarrow aSb$

$\Rightarrow abSba$

$= aabb$

- Can you derive the string abba?

Designing CFGs: A Caveat

- When designing a CFG for a language, make sure that it
  - generates all the strings in the language and
  - never generates a string outside the language.
- The first of these can be tricky – make sure to test your grammars!

CFG Caveats II

- Is the following grammar a CFG for the language $\{ a^n b^n \mid n \in \mathbb{N} \}$?

$$S \to aSb$$

$S \to aSb$
$aaSbb$
$aaaSbbb$

- What strings can you derive?
  - Answer: None!
- What is the language of the grammar?
  - Answer: Ø
- When designing CFGs, make sure your recursion terminates!

CFG Caveats III

- When designing CFGs, remember that each nonterminal can be expanded out independently of the others.
- Let $\Sigma = \{a, \overset{?}{=}\}$ and let $L = \{a^n \overset{?}{=} a^n \mid n \in \mathbb{N}\}$.
- Is the following a CFG for L?

$$S \rightarrow X \overset{?}{=} X$$
$$X \rightarrow aX \mid \varepsilon$$

$S \Rightarrow X \overset{?}{=} X$

$$
\begin{aligned}
S &\Rightarrow X \overset{?}{=} X \\
&\Rightarrow aX \overset{?}{=} X \\
&\Rightarrow aaX \overset{?}{=} X \\
&\Rightarrow aa \overset{?}{=} X \\
&\Rightarrow aa \overset{?}{=} aX \\
&\Rightarrow aa \overset{?}{=} a
\end{aligned}
$$

Finding a Build Order

- Let $\Sigma = \{a, \overset{?}{=}\}$ and let $L = \{a^n \overset{?}{=} a^n \mid n \in \mathbb{N}\}$.
- To build a CFG for L, we need to be more clever with how we construct the string.
  - If we build the strings of a's independently of one another, then we can't enforce that they have the same length.
  - Idea: Build both strings of a's at the same time.
- Here's one possible grammar based on that idea:

$$S \to \overset{?}{=} \mid aSa$$

$$\begin{aligned} S &\Rightarrow aSa \\ &\Rightarrow aaSaa \\ &\Rightarrow aaaSaaa \\ &\Rightarrow aaa\overset{?}{=}aaa \end{aligned}$$

Function Prototypes

- Let $\Sigma = \{$void, int, double, name, (,), ,, ;$\}$.
- Let's write a CFG for C-style function prototypes!
- Examples:
  - void name(int name, double name);
  - int name();
  - int name(double name);
  - int name(int, int name, int);
  - void name(void);

Function Prototypes

- Here's one possible grammar:
  - S → Ret name (Args);
  - Ret → Type | void
  - Type → int | double
  - Args → ε | void | ArgList
  - ArgList → OneArg | ArgList, OneArg
  - OneArg → Type | Type name

- Fun question to think about: what changes would you need to make to support pointer types?

CFGs for Programming Languages

BLOCK → STMT
 | { STMTS }
STMTS → ε
 | STMT STMTS
STMT → EXPR;
 | if (EXPR) BLOCK
 | while (EXPR) BLOCK
 | do BLOCK while (EXPR);
 | BLOCK
 | …

EXPR → identifier
 | constant
 | EXPR + EXPR
 | EXPR – EXPR
 | EXPR * EXPR
 | …

Reading and Exercises

Reading

- Chapter: 2.2 (Michael Scott Book)

References

Lecture Materials of CS 103, Stanford University