Context-Free Languages

Theory of Computation

CISC 603, Spring 2020, Daqing Yun

Recall

- Theorem: For every finite automaton $M=(Q, \Sigma, q_0, A, \delta)$, the language $L(M)$ is regular
- It would be much more convenient if we could just write `(a(|b))*` instead of building the abstract syntax tree manually with **repeat**, **choose**, and **concatenate**
- We wanna build a (programming) language parser that can understand certain patterns/rules and automatically transform raw syntax into Abstract Syntax Trees (ASTs)

Language Grammars

- Generate a language parser that can "understand" certain patterns/rules and automatically transform raw syntax into ASTs
- Language grammar – A set of rules describing accepted languages

Why?

- Context-free grammars (CFGs) are used to describe the syntax of essentially *every* modern programming language
- Every modern complier uses CFG concepts to parse programs
  - Not to forget their important role in describing natural languages
  - Useful for nested structures, e.g., parentheses in programming languages
- And Document Type Definitions are really CFG's

Using Grammar Rules to Define a Language

- Regular languages and FAs are too simple for many purposes
  - Using *context-free grammars* allows us to describe more interesting languages
  - Much high-level programming language syntax can be expressed with context-free grammars
  - Context-free grammars with a very simple form provide another way to describe the regular languages
- We will study how derivations can be related to the structure of the string being derived

Informal Comments

- A *context-free grammar* is a notation for describing languages
- It is more powerful than finite automata or RegEx's, but still cannot define all possible languages
- Useful for nested structures, e.g., parentheses in programming languages

Informal Comments (cont’d.)

- Basic idea is to use “variables” to stand for sets of strings (i.e., languages)
- These variables are defined *recursively*, in terms of one another
- Recursive rules (“productions”) involve only concatenation
- Alternative rules for a variable allow union

Using Grammar Rules to Define a Language (cont'd.)

- A grammar is a set of rules, usually simpler than those of English, by which strings in a language can be generated
- Consider the language $L = \{a^n b^n \mid n \geq 0\}$, defined using the *recursive* definition:
  - $\Lambda \in L$
  - For every $S \in L$, $aSb \in L$
- Think of S as a variable representing an arbitrary element, and write these rules as $S \to \Lambda \quad S \to aSb$

(In the process of obtaining an element of L, S can be replaced by either string)

Using Grammar Rules to Define a Language (cont'd.)

- If α and β are strings, and α contains at least one occurrence of S, then $\alpha => \beta$ means that β is obtained from α in one step, by using one of the two rules to replace a single occurrence of S by either Λ or aSb
- For example, we could write:

$$S => aSb => aaSbb => aaaSbbb => aaabbb$$

to describe a *derivation* of the string *aaabbb*

- We can simplify the rules by using the | symbol to mean "or", so that the rules become $S \rightarrow \Lambda \mid aSb$

Example: CFG for $\{0^n 1^n \mid n \geq 1\}$

- Productions:

  $S \rightarrow 01$

  $S \rightarrow 0S1$

- Basis: 01 is in the language
- Induction: if w is in the language, then so is $0w1$

CFG Formalism

- Terminals = symbols of the alphabet of the language being defined
- Variables = nonterminals = a finite set of other symbols, each of which represents a language
- Start symbol = the variable whose language is the one being defined

Context-Free Grammars: Definitions and More Examples

- Definition: A *context-free grammar* (CFG) is a 4-tuple G=(V, Σ, S, P), where V and Σ are disjoint finite sets, $S \in V$, and P is a finite set of formulas of the form

  A −> α, where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$

  – Elements of Σ are *terminal symbols*, or *terminals*, and elements of V are *variables*, or *nonterminals*

  – S is the *start* variable, and elements of P are *grammar rules,* or *productions*

  – We use −> for productions in a grammar and => for a step in a derivation

  – The notations $\alpha =>^n \beta$ and $\alpha =>^* \beta$ refer to n steps and *zero or more* steps, respectively

Example: Formal CFG

- Here is a formal CFG for $\{0^n1^n \mid n \geq 1\}$
- Terminals = $\{0, 1\}$
- Variables = $\{S\}$
- Start symbol = S
- Productions =

  $S \rightarrow 01$

  $S \rightarrow 0S1$

Context-Free Grammars: Definitions and More Examples (cont'd.)

- We will sometimes write $=>_G$ to indicate a derivation in a particular grammar G
- $\alpha => \beta$ means that there are strings α_1, α_2, and γ in $(V \cup \Sigma)^*$ and a production $A \rightarrow \gamma$ in P such that $\alpha = \alpha_1 A \alpha_2$ and $\beta = \alpha_1 \gamma \alpha_2$
  - This is a single step in a derivation
- What makes the grammar *context-free* is that the *production* above, with left side A, can be applied wherever A occurs in the string (irrespective of the context; i.e., regardless of what α_1 and α_2 are)

Derivations – Intuition

- We *derive* strings in the language of a CFG by starting with the start symbol, and repeatedly replacing some variable A by the right side of one of its productions
  - That is, the "productions for A" are those that have A on the left side of the $->$

Derivations – Formalism

- We say $\alpha A \beta$ => $\alpha\gamma\beta$ if A –> γ is a production
- Example: S –> 01; S –> 0S1
- S => 0S1 => 00S11 => 000111

Iterated Derivation

- =>* means "zero or more derivation steps"
- Basis: α =>* α for any string α
- Induction: if α =>* β and β => γ, then α =>* γ

Example: Iterated Derivation

- S -> 01; S -> 0S1
- S => 0S1 => 00S11 => 000111
- So S =>* S; S =>* 0S1; S =>* 00S11; S =>* 000111

Sentential Forms

- Any string of variables and/or terminals derived from the start symbol is called a *sentential form*
- Formally, α is a sentential form iff $S =>^* \alpha$

Language of a Grammar

- If G is a CFG, then $L(G)$, the *language of G*, is $\{w \mid S \Rightarrow^* w\}$
  - Note: w must be a terminal string, S is the start symbol
- Example: G has productions $S \rightarrow \varepsilon$ and $S \rightarrow 0S1$
- $L(G) = \{0^n 1^n \mid n \geq 0\}$

Note: ε is a legitimate right side

Context-Free Languages

- A language that is defined by some CFG is called a *context-free language*
- There are CFL's that are not regular languages, such as the example just given
- But not all languages are CFL's
- Intuitively: CFL's can count two things, not three

BNF Notation

- Grammars for programming languages are often written in BNF (*Backus-Naur Form*)
- Variables are words in `<...>`
  - Example: `<statement>`
- Terminals are often multicharacter strings indicated by boldface or underline
  - Example: `while` or `WHILE`

BNF Notation (cont'd.)

- Symbol : : = is often used for –>
- Symbol | is used for "or"
  - A shorthand for a list of productions with the same left side
- Example:
  - S –> $0S1$ | 01 is shorthand for S –> $0S1$ and S –> 01

BNF Notation – Kleene Closure

- Symbol `...` is used for "one or more"
- Example:
  - `<digit> ::= 0|1|2|3|4|5|6|7|8|9`
  - `<unsigned integer>::=<digit>...`
  - Note: that's not exactly the `*` of RegEx's
- Translation: replace α... with a new variable A and productions $A \rightarrow A\alpha \mid \alpha$

Example: Kleene Closure

- Grammar for unsigned integers can be replaced by   Note that `::=` is often used for `->`
  - `U ::= D...`
  - `U -> UD | D`
  - `D -> 0|1|2|3|4|5|6|7|8|9`

BNF Notation: Optional Elements

- Surround one or more symbols by [...] to make them optional
- Example:
  - `<statement>::=if<condition>then <statement>[;else<statement>]`
- Translation: replace [α] by a new variable A with productions $A \rightarrow \alpha \mid \varepsilon$
- Example: grammar for if-then-else can be replaced by

$$S \rightarrow iCtSA$$

$$A \rightarrow ;eS \mid \varepsilon$$

BNF Notation – Grouping

- Use {…} to surround a sequence of symbols that need to be treated as a unit
  - Typically, they are followed by a ... for "one or more"
- Example:
  - `<statement list>::=<statement> [{;<statement>}...]`

Translation: Grouping

- You may, if you wish, create a new variable A for $\{\alpha\}$
- One production for A is $A \rightarrow \alpha$
- Use A in place of $\{\alpha\}$

Example: Grouping

```
L->S[{;S}...]
```

A stands for {;S}

```
L->S[A...]
A->;S
```

B stands for [A…]
(zero or more A's)

```
L->SB
B->A...|ε
A->;S
```

C stands for A...

```
L->SB
B->C|ε
C->AC|A
A->;S
```

Derivation Trees and Ambiguity

- So far we've been interested in *what* strings a CFG generates
- It is also useful to consider *how* a string is generated by a CFG
- A *derivation* may provide information about the structure of a string, and if a string has several possible derivations, one may be more appropriate than another
- We can draw trees to represent derivations

Leftmost and Rightmost Derivations

- Derivations allow us to replace any of the variables in a string
- Leads to many different derivations of the same string
- By forcing the leftmost variable (or alternatively, the rightmost variable) to be replaced, we avoid these “distinctions without a difference”

Leftmost Derivations

- Say $wA\alpha =>_{lm} w\beta\alpha$ if w is a string of terminals only and $A -> \beta$ is a production
- Also, $\alpha =>^*_{lm} \beta$ if α becomes β by a sequence of 0 or more $=>_{lm}$ steps

A derivation in a context-free grammar is a leftmost derivation (LMD) if, at each step, a production is applied to the leftmost variable-occurrence in the current string. A rightmost derivation (RMD) is defined similarly.

Example: Leftmost Derivations

- Balanced-parentheses grammar:

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

- $S \Rightarrow_{lm} SS \Rightarrow_{lm} (S)S \Rightarrow_{lm} (())S \Rightarrow_{lm} (())()$
- Thus, $S \Rightarrow^*_{lm} (())()$
- $S \Rightarrow SS \Rightarrow S() \Rightarrow (S)() \Rightarrow (())()$ is a derivation, but not a leftmost derivation

Rightmost Derivations

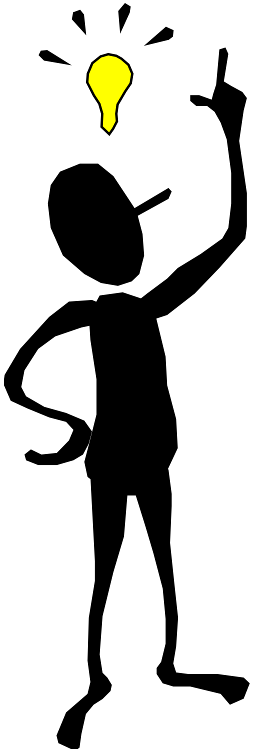
- Say $\alpha A w \Rightarrow_{rm} \alpha\beta w$ if w is a string of terminals only and $A \rightarrow \beta$ is a production
- Also, $\alpha \Rightarrow^*_{rm} \beta$ if α becomes β by a sequence of 0 or more $\Rightarrow_{rm}$ steps

Example: Rightmost Derivations

- Balanced-parentheses grammar:

  $S \to SS \mid (S) \mid ()$

- $S \Rightarrow_{rm} SS \Rightarrow_{rm} S() \Rightarrow_{rm} (S)() \Rightarrow_{rm} (())()$
- Thus, $S \Rightarrow^*_{rm} (())()$
- $S \Rightarrow SS \Rightarrow SSS \Rightarrow S()S \Rightarrow ()()S \Rightarrow ()()()$ is neither a rightmost nor a leftmost derivation



Thanks ! ☺

Questions ?