CS21
Decidability
and
Tractability

Lecture 6
January 17, 2024

1

Context-free grammars and languages

- languages recognized by a (N)FA are exactly the languages described by regular expressions, and they are called the regular languages
- languages recognized by a NPDA are exactly the languages described by context-free grammars, and they are called the context-free languages

January 17, 2024 CS21 Lecture 6 2

2

Context-Free Grammars

start symbol

terminal symbols

A → 0A1
A → B
B → #

non-terminal symbols

production

January 17, 2024 CS21 Lecture 6 3

3

Context-Free Grammars

- generate strings by repeated replacement of non-terminals with string of terminals and non-terminals
  - write down start symbol (non-terminal)
  - replace a non-terminal with the right-hand-side of a rule that has that non-terminal as its left-hand-side.
  - repeat above until no more non-terminals

January 17, 2024 CS21 Lecture 6 4

4

Context-Free Grammars

A → 0A1
A → B
B → #

Example:
A ⇒ 0A1 ⇒ 00A11 ⇒
000A111 ⇒ 000B111 ⇒
000#111

- a derivation of the string 000#111
- set of all strings generated in this way is the language of the grammar L(G)
- called a Context-Free Language

January 17, 2024 CS21 Lecture 6 5

5

Context-Free Grammars

- Natural languages (e.g. English) structure:

shorthand for multiple rules with same lhs

<sentence> → <noun-phrase><verb-phrase>
<noun-phrase> → <cpx-noun> | <cpx-noun><prep-phrase>
<verb-phrase> → <cpx-verb> | <cpx-verb><prep-phrase>
<prep-phrase> → <prep><cpx-noun>
<cpx-noun> → <article><noun>
<cpx-verb> → <verb>|<verb><noun-phrase>
<article> → a | the
<noun> → dog | cat | flower
<verb> → eats | sees
<prep> → with

Generate a string in the language of this grammar.

January 17, 2024 CS21 Lecture 6 6

6

Context-Free Grammars

- CFGs don't capture natural languages completely
- computer languages often defined by CFG
  - hierarchical structure
  - slightly different notation often used "Backus-Naur form"
  - see next slide for example

Example CFG

<stmt> → <if-stmt> | <while-stmt> | <begin-stmt>
| <asgn-stmt>
<if-stmt> → IF <bool-expr> THEN <stmt> ELSE <stmt>
<while-stmt> → WHILE <bool-expr> DO <stmt>
<begin-stmt> → BEGIN <stmt-list> END
<stmt-list> → <stmt> | <stmt>; <stmt-list>
<asgn-stmt> → <var> := <arith-expr>
<bool-expr> → <arith-expr><compare-op><arith-expr>
<compare-op> → < | > | ≤ | ≥ | =
<arith-expr> → <var> | <const>
| (<arith-expr><arith-op><arith-expr>)
<arith-op> → + | - | * | /
<const> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<var> → a | b | c | … | x | y | z

CFG formal definition

- A context-free grammar is a 4-tuple

(V, Σ, R, S)

where

- V is a finite set called the non-terminals
- Σ is a finite set (disjoint from V) called the terminals
- R is a finite set of productions where each production is a non-terminal and a string of terminals and non-terminals.
- S ∈ V is the start variable (or start non-terminal)

CFG formal definition

- u, v, w are strings of non-terminals and terminals, and A → w is a production:

"uAv yields uwv" notation: $uAv \Rightarrow uwv$

also: "yields in 1 step" notation: $uAv \Rightarrow^1 uwv$

- in general:

"yields in k steps" notation: $u \Rightarrow^k v$

  - meaning: there exists strings $u_1, u_2, \ldots u_{k-1}$ for which $u \Rightarrow^1 u_1 \Rightarrow^1 u_2 \Rightarrow^1 \ldots \Rightarrow^1 u_{k-1} \Rightarrow^1 v$

CFG formal definition

- notation: $u \Rightarrow^* v$
  - meaning: $\exists\, k \geq 0$ and strings $u_1, \ldots, u_{k-1}$ for which $\underbrace{u \Rightarrow^1 u_1 \Rightarrow^1 u_2 \Rightarrow^1 \ldots \Rightarrow^1 u_{k-1} \Rightarrow^1 v}$
- if u = start symbol, this is a derivation of v
- The language of G, denoted L(G) is:

$\{w \in \Sigma^* : S \Rightarrow^* w\}$

CFG example

- Balanced parentheses:
  - ( )
  - (( ) ((( ) ( ))))
- a string w in Σ* = { (,) }* is balanced iff:
  - # "("s equals # ")"s, and
  - for any prefix of w, # "("s ≥ # ")"s

Exercise: design a CFG for balanced parentheses.

CFG example

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

- Proof that w ∈ L(G) implies w is balanced
  - induction on length of derivation
  - base case: length 1: $S \Rightarrow \epsilon$
  - general case: length n
    - $S \Rightarrow (S) \Rightarrow^{n-1} (w') = w$
    - $S \Rightarrow SS \Rightarrow^{n-1} w'w'' = w$

13

CFG example

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

- Proof that w is balanced implies w ∈ L(G)
  - induction on length of w
  - base case: length 0: $w = \epsilon$
  - general case: length n
  - consider shortest prefix in language
  - if whole string then w = (w') and w' balanced
  - if proper prefix then w = w'w'' with w' and w'' balanced

14