CS21
Decidability and Tractability
Lecture 7
January 19, 2022

Outline
• Context-Free Grammars and Languages
• equivalence of NPDAs and CFGs
• non context-free languages via CFL Pumping Lemma

Context-Free Grammars

start symbol

A → 0A1
A → B
B → #

production
terminal symbols

non-terminal symbols

CFG example
• Balanced parentheses:
  – ( )
  – ( ( ( ) ( ) ) )

  a string \( w \) in \( \Sigma^* = \{ (, ) \}^* \) is balanced iff:
  – \# "(" equals \# ")"s, and
  – for any prefix of \( w \), \# "(" equals \# ")"s

Exercise: design a CFG for balanced parentheses.

CFG example

\[ S \rightarrow (S) \mid SS \mid \epsilon \]

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

CFG example

\[ S \rightarrow (S) \mid SS \mid \epsilon \]

• Proof that \( w \) is balanced implies \( w \in L(G) \)
  – 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
CFG example

• Arithmetic expressions over \{+,\ast,\cdot,a\}
  - \((a + a) \ast a\)
  - \(a \ast a + a + a + a\)

• A CFG generating this language:
  \[<\text{expr}> \rightarrow <\text{expr}> \ast <\text{expr}>]
  \[<\text{expr}> \rightarrow <\text{expr}> + <\text{expr}>]
  \[<\text{expr}> \rightarrow (<\text{expr}>) | a\]

A derivation of the string: \(a + a \ast a\)

\[<\text{expr}> \Rightarrow <\text{expr}> \ast <\text{expr}>\]
\[<\text{expr}> + <\text{expr}> \ast <\text{expr}>\]
\[a + <\text{expr}> \ast <\text{expr}>\]
\[a + a \ast <\text{expr}>\]
\[a + a \ast a\]

Parse Trees

• Easier way to picture derivation: parse tree

  \[<\text{expr}> \rightarrow <\text{expr}> \ast <\text{expr}>\]
  \[<\text{expr}> \rightarrow <\text{expr}> + <\text{expr}>\]
  \[<\text{expr}> \rightarrow (<\text{expr}>) | a\]

  grammar encodes grouping information; this is captured in the parse tree.

Solution to problem

\[<\text{expr}> \rightarrow <\text{expr}> + <\text{term}> | <\text{term}>\]
\[<\text{term}> \rightarrow <\text{term}> \ast <\text{factor}> | <\text{factor}>\]
\[<\text{factor}> \rightarrow <\text{term}> \ast <\text{factor}>\]
\[<\text{factor}> \rightarrow (<\text{expr}>) | a\]

• forces correct precedence in parse tree grouping
  – within parentheses, \(\ast\) cannot occur as ancestor of \(+\) in the parse tree.
Some facts about CFLs

- CFLs are closed under:
  - union (proof?)
  - concatenation (proof?)
  - star (proof?)

- Every regular language is a CFL – proof?

NPDA, CFG equivalence

Theorem: a language $L$ is recognized by a NPDA iff $L$ is described by a CFG.

Must prove two directions:

$(\Rightarrow)$ $L$ is recognized by a NPDA implies $L$ is described by a CFG.

$(\Leftarrow)$ $L$ is described by a CFG implies $L$ is recognized by a NPDA.

NPDA, CFG equivalence

Proof of $(\Leftarrow)$: $L$ is described by a CFG implies $L$ is recognized by a NPDA.

NPDA, CFG equivalence

1. we’d like to non-deterministically guess the derivation, forming it on the stack
2. then scan the input, popping matching symbol off the stack at each step
3. accept if we get to the bottom of the stack at the end of the input.

what is wrong with this approach?

NPDA, CFG equivalence

- only have access to top of stack
- combine steps 1 and 2:
  - allow to match stack terminals with tape during the process of producing the derivation on the stack

NPDA, CFG equivalence

- informal description of construction:
  - place $\$$ and start symbol $S$ on the stack
  - repeat:
    - if the top of the stack is a non-terminal $A$, pick a production with $A$ on the lhs and substitute the rhs for $A$ on the stack
    - if the top of the stack is a terminal $b$, read $b$ from the tape, and pop $b$ from the stack.
    - if the top of the stack is $\$$, enter the accept state.
NPDA, CFG equivalence

Proof of (⇒): L is recognized by a NPDA implies L is described by a CFG.

– harder direction
– first step: convert NPDA into “normal form”:
  • single accept state
  • empties stack before accepting
  • each transition either pushes or pops a symbol

NPDA, CFG equivalence

– main idea: non-terminal $A_{p,q}$ generates exactly the strings that take the NPDA from state $p$ (w/ empty stack) to state $q$ (w/ empty stack)

– then $A_{\text{start, accept}}$ generates all of the strings in the language recognized by the NPDA.