


CS21
Decidability
and
Tractability



Lecture 7
January 19, 2024

1

CFG example

- Arithmetic expressions over $\{+, *, (,), a\}$
 - $-(a + a) * a$
 - $-a * a + a + a + a + a$
- A CFG generating this language:
 - $\langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle * \langle \text{expr} \rangle$
 - $\langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle + \langle \text{expr} \rangle$
 - $\langle \text{expr} \rangle \rightarrow (\langle \text{expr} \rangle) \mid a$

January 19, 2024 CS21 Lecture 7 2

2

CFG example

$\langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle * \langle \text{expr} \rangle$
 $\langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle + \langle \text{expr} \rangle$
 $\langle \text{expr} \rangle \rightarrow (\langle \text{expr} \rangle) \mid a$

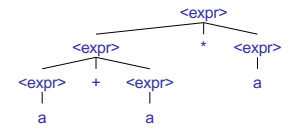
- A derivation of the string: $a+a*a$
 - $\langle \text{expr} \rangle \Rightarrow \langle \text{expr} \rangle * \langle \text{expr} \rangle$
 - $\Rightarrow \langle \text{expr} \rangle + \langle \text{expr} \rangle * \langle \text{expr} \rangle$
 - $\Rightarrow a + \langle \text{expr} \rangle * \langle \text{expr} \rangle$
 - $\Rightarrow a + a * \langle \text{expr} \rangle$
 - $\Rightarrow a + a * a$

January 19, 2024 CS21 Lecture 7 3

3

Parse Trees

- Easier way to picture derivation: **parse tree**



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

January 19, 2024 CS21 Lecture 7 4

4

CFGs and parse trees

$\langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle * \langle \text{expr} \rangle$
 $\langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle + \langle \text{expr} \rangle$
 $\langle \text{expr} \rangle \rightarrow (\langle \text{expr} \rangle) \mid a$

- Is this a good grammar for arithmetic expressions?
 - can group wrong way (+ precedence over *)
 - different grammar for same language can force correct precedence/grouping

January 19, 2024 CS21 Lecture 7 5

5

Some facts about CFLs

- CFLs are closed under
 - union (proof?)
 - concatenation (proof?)
 - star (proof?)
- Every regular language is a CFL
 - proof?

January 19, 2024 CS21 Lecture 7 6

6

NPDA, CFG equivalence

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

Must prove *two* directions: (proof next lecture)

- (\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.

January 19, 2024 CS21 Lecture 7 7

7

NPDA, CFG equivalence

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

an idea:

January 19, 2024 CS21 Lecture 7 8

8

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?

January 19, 2024 CS21 Lecture 7 9

9

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

January 19, 2024 CS21 Lecture 7 10

10

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.

January 19, 2024 CS21 Lecture 7 11

11

NPDA, CFG equivalence

January 19, 2024 CS21 Lecture 7 12

12

NPDA, CFG equivalence

Proof of (\Rightarrow): 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

January 19, 2024 CS21 Lecture 7 13

13

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

January 19, 2024 CS21 Lecture 7 14

14

NPDA, CFG equivalence

- Two possibilities to get from state p to q:

input → abcabbacacbacacabacabbabbaaacabbababaaacaccacccc

string taking NPDA from p to q

January 19, 2024 CS21 Lecture 7 15

15

NPDA, CFG equivalence

- NPDA $P = (Q, \Sigma, \Gamma, \delta, start, \{accept\})$
- CFG G:
 - non-terminals $V = \{A_{p,q} : p, q \in Q\}$
 - start variable $A_{start, accept}$
 - productions:
 - for every $p, r, q \in Q$, add the rule $A_{p,q} \rightarrow A_{p,r}A_{r,q}$

January 19, 2024 CS21 Lecture 7 16

16

NPDA, CFG equivalence

- Two possibilities to get from state p to q:

input → abcabbacacbacacabacabbabbaaacabbababaaacaccacccc

string taking NPDA from p to q

January 19, 2024 CS21 Lecture 7 17

17

NPDA, CFG equivalence

- NPDA $P = (Q, \Sigma, \Gamma, \delta, s, \{accept\})$
- CFG G:
 - non-terminals $V = \{A_{p,q} : p, q \in Q\}$
 - start variable $A_{start, accept}$
 - productions:
 - for every $p, r, s, q \in Q, d \in \Gamma$ and $a, b \in (\Sigma \cup \{\epsilon\})$ if $(r, d) \in \delta(p, a, \epsilon)$, and $(q, \epsilon) \in \delta(s, b, d)$, add the rule $A_{p,q} \rightarrow aA_{r,s}b$

January 19, 2024 CS21 Lecture 7 18

18

NPDA, CFG equivalence

- NPDA $P = (Q, \Sigma, \Gamma, \delta, \text{start}, \{\text{accept}\})$
- CFG G :
 - non-terminals $V = \{A_{p,q} : p, q \in Q\}$
 - start variable $A_{\text{start}, \text{accept}}$
 - productions:
 - for every $p \in Q$, add the rule
 - $A_{p,p} \rightarrow \epsilon$

January 19, 2024 CS21 Lecture 7 19

19

NPDA, CFG equivalence

- two claims to verify correctness:
 1. if $A_{p,q}$ generates string x , then x can take NPDA P from state p (w/ empty stack) to q (w/ empty stack)
 2. if x can take NPDA P from state p (w/ empty stack) to q (w/ empty stack), then $A_{p,q}$ generates string x

January 19, 2024 CS21 Lecture 7 20

20

NPDA, CFG equivalence

1. if $A_{p,q}$ generates string x , then x can take NPDA P from state p (w/ empty stack) to q (w/ empty stack)
 - induction on length of derivation of x .
 - base case: 1 step derivation. must have only terminals on rhs. In G , must be production of form $A_{p,p} \rightarrow \epsilon$.

January 19, 2024 CS21 Lecture 7 21

21

NPDA, CFG equivalence

1. if $A_{p,q}$ generates string x , then x can take NPDA P from state p (w/ empty stack) to q (w/ empty stack)
 - assume true for derivations of length at most k , prove for length $k+1$.
 - verify case: $A_{p,q} \rightarrow A_{p,r}A_{r,q} \rightarrow^k x = yz$
 - verify case: $A_{p,q} \rightarrow aA_{r,s}b \rightarrow^k x = ayb$

January 19, 2024 CS21 Lecture 7 22

22

NPDA, CFG equivalence

2. if x can take NPDA P from state p (w/ empty stack) to q (w/ empty stack), then $A_{p,q}$ generates string x
 - induction on # of steps in P 's computation
 - base case: 0 steps. starts and ends at same state p . only has time to read empty string ϵ .
 - G contains $A_{p,p} \rightarrow \epsilon$.

January 19, 2024 CS21 Lecture 7 23

23

NPDA, CFG equivalence

2. if x can take NPDA P from state p (w/ empty stack) to q (w/ empty stack), then $A_{p,q}$ generates string x
 - induction step. assume true for computations of length at most k , prove for length $k+1$.
 - if stack becomes empty sometime in the middle of the computation (at state r)
 - y is read going from state p to r $(A_{p,r} \rightarrow^* y)$
 - z is read going from state r to q $(A_{r,q} \rightarrow^* z)$
 - conclude: $A_{p,q} \rightarrow A_{p,r}A_{r,q} \rightarrow^* yz = x$

January 19, 2024 CS21 Lecture 7 24

24

NPDA, CFG equivalence

2. if x can take NPDA P from state p (w/ empty stack) to q (w/ empty stack), then $A_{p,q}$ generates string x

- if stack becomes empty only at beginning and end of computation.
 - first step: state p to r , read a , push d
 - go from state r to s , read string y ($A_{r,s} \rightarrow^* y$)
 - last step: state s to q , read b , pop d
 - conclude: $A_{p,q} \rightarrow a A_{r,s} b \rightarrow^* ayb = x$

January 19, 2024 CS21 Lecture 7 25

25

NPDA, CFG equivalence

2. if x can take NPDA P from state p (w/ empty stack) to q (w/ empty stack), then $A_{p,q}$ generates string x

- if stack becomes empty only at beginning and end of computation.
 - first step: state p to r , read a , push d
 - go from state r to s , read string y ($A_{r,s} \rightarrow^* y$)
 - last step: state s to q , read b , pop d
 - conclude: $A_{p,q} \rightarrow a A_{r,s} b \rightarrow^* ayb = x$

January 19, 2024 CS21 Lecture 7 26

26

Pumping Lemma for CFLs

CFL Pumping Lemma: Let L be a CFL. There exists an integer p ("pumping length") for which every $w \in L$ with $|w| \geq p$ can be written as $w = uvxyz$ such that

1. for every $i \geq 0$, $uv^i x y^i z \in L$, and
2. $|vy| > 0$, and
3. $|vxy| \leq p$.

January 19, 2024 CS21 Lecture 7 27

27

CFL Pumping Lemma Example

Theorem: the following language is not context-free:

$$L = \{a^n b^n c^n : n \geq 0\}.$$

- Proof:
 - let p be the pumping length for L
 - choose $w = a^p b^p c^p$
 - $w = aaaa\dots abbbb\dots bcccc\dots c$
 - $w = uvxyz$, with $|vy| > 0$ and $|vxy| \leq p$.

January 19, 2024 CS21 Lecture 7 28

28

CFL Pumping Lemma Example

- possibilities:

$$w = \underbrace{aaaa\dots}_u \underbrace{aaabbb\dots}_v \underbrace{bbcccc\dots}_x \underbrace{\dots}_y \underbrace{\dots}_z c$$

(if v , y each contain only one type of symbol, then pumping on them produces a string not in the language)

January 19, 2024 CS21 Lecture 7 29

29

CFL Pumping Lemma Example

- possibilities:

$$w = \underbrace{aaaa\dots}_u \underbrace{abbbb\dots}_v \underbrace{bcccc\dots}_x \underbrace{\dots}_y \underbrace{\dots}_z c$$

(if v or y contain more than one type of symbol, then pumping on them might produce a string with equal numbers of a 's, b 's, and c 's - if vy contains equal numbers of a 's, b 's, and c 's. But they will be out of order.)

January 19, 2024 CS21 Lecture 7 30

30

CFL Pumping Lemma

- how large should pumping length p be?
- need to ensure other conditions:
 $|vy| > 0$ $|vxy| \leq p$
- $b = \max$ # symbols on rhs of any production (assume $b \geq 2$)
- if parse tree has height $\leq h$, then string generated has length $\leq b^h$ (so length $> b^h$ implies height $> h$)

37

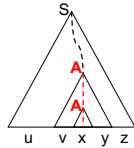
CFL Pumping Lemma

- let m be the # of nonterminals in the grammar
- to ensure path of length at least $m+2$, require
 $|w| \geq p = b^{m+2}$
- since $|w| > b^{m+1}$, any parse tree for w has height $> m+1$
- let T be the **smallest** parse tree for w
- longest root-leaf path must consist of $\geq m+1$ non-terminals and 1 terminal.

38

CFL Pumping Lemma

- must be a repeated non-terminal **A** on long path
- select a repetition among the **lowest** $m+1$ non-terminals on path.
- pictures show that for every $i \geq 0$, $uv^ixy^iz \in L$
- is $|vy| > 0$?
 - smallest parse tree T ensures
- is $|vxy| \leq p$?
 - red path has length $\leq m+2$, so $\leq b^{m+2} = p$ leaves



39