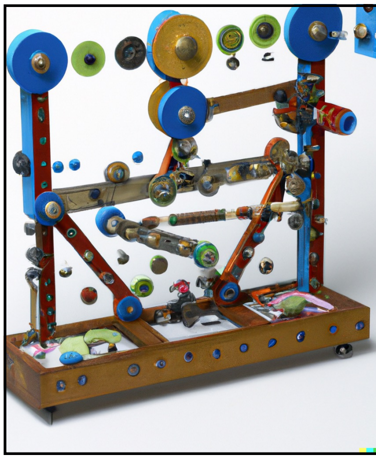


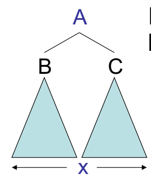
CS21
Decidability
and
Tractability

Lecture 9
January 24, 2024

1

Deciding CFLs

- Convert CFG into Chomsky Normal Form
- parse tree for string x generated by nonterminal A:



If $A \rightarrow^k x$ $(k > 1)$ then there must be a way to split x:

x = yz

- $A \rightarrow BC$ is a production and
- $B \rightarrow^i y$ and $C \Rightarrow^j z$ for $i, j < k$

2

Deciding CFLs

- An algorithm:

IsGenerated(x, A)

```
if |x| = 1, then return YES if A → x is a production,
  else return NO
for all n-1 ways of splitting x = yz
  for all ≤ m productions of form A → BC
    if IsGenerated(y, B) and IsGenerated(z, C),
          return YES
return NO
```

- worst case running time?

3

Deciding CFLs

- worst case running time exp(n)
- Idea: avoid recursive calls
  - build table of YES/NO answers to calls to IsGenerated, in order of length of substring
  - example of general algorithmic strategy called dynamic programming
  - notation: x[i,j] = substring of x from i to j
  - table: T(i, j) contains

    {A: A nonterminal such that $A \rightarrow^* x[i,j]$}

4

Deciding CFLs

IsGenerated($x = x_1x_2x_3\ldots x_n$, G)

```
for i = 1 to n
  T[i, i] = {A: "A → x_i" is a production in G}
for k = 1 to n - 1
  for i = 1 to n - k
        for k splittings x[i, i+k] = x[i,i+j]x[i+j+1, i+k]
              T[i, i+k] = {A: "A → BC" is a production
                    in G and B ∈ T[i,i+j] and
                    C ∈ T[i+j+1,i+k] }
output "YES" if S ∈ T[1, n], else output "NO"
```


5

Deciding CFLs

IsGenerated($x = x_1x_2x_3\ldots x_n$, G)

O(nm) steps

```
for i = 1 to n
  T[i, i] = {A: "A → x_i" is a production in G}
for k = 1 to n - 1
  for i = 1 to n - k
        for k splittings x[i, i+k] = x[i,i+j]x[i+j+1, i+k]
              T[i, i+k] = {A: "A → BC" is a production
                    in G and B ∈ T[i,i+j] and
                    C ∈ T[i+j+1,i+k] }
output "YES" if S ∈ T[1, n], else output "NO"
```

$O(n^3m^3)$ steps

6

Deterministic PDA

- A NPDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where:
  - $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q \times (\Gamma \cup \{\varepsilon\}))$ is a function called the transition function
- A deterministic PDA has only one option at every step:
  - for every state $q \in Q$, $a \in \Sigma$, and $t \in \Gamma$, exactly 1 element in $\delta(q, a, t)$, or
  - exactly 1 element in $\delta(q, \varepsilon, t)$, and $\delta(q, a, t)$ empty for all $a \in \Sigma$

7

Deterministic PDA

- A technical detail:

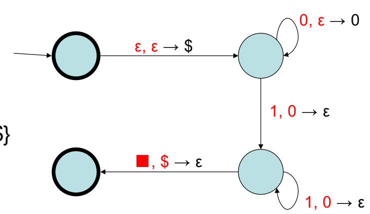
  we will give our deterministic machine the ability to detect end of input string
  - add special symbol ■ to alphabet
  - require input tape to contain x■
- language recognized by a deterministic PDA is called a deterministic CFL (DCFL)

8

Example deterministic PDA

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, \$\}$



$L = \{0^n1^n : n \geq 0\}$

(unpictured transitions go to a "reject" state and stay there)

9

Deterministic PDA

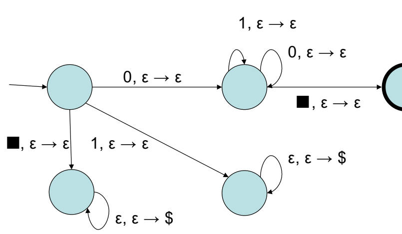
Theorem: DCFLs are closed under complement

(complement of L in Σ^* is $(\Sigma^* - L)$)

Proof attempt:
- swap accept/non-accept states
- problem: might enter infinite loop before reading entire string
- machine for complement must accept in these cases, and read to end of string

10

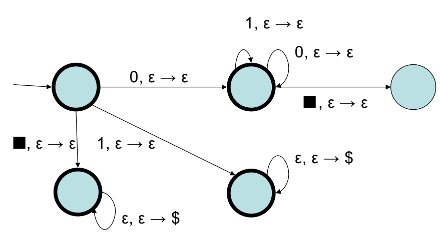
Example of problem



Language of this DPDA is $0\Sigma^*$

11

Example of problem



Language of this DPDA is $\{\epsilon\}$

12

Deterministic PDA

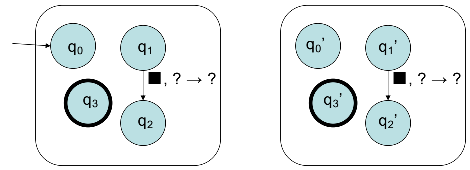
Proof:

- convert machine into "normal form"
  - always reads to end of input
  - always enters either an accept state or single distinguished "reject" state, and stay there
- step 1: keep track of when we have read to end of input
- step 2: eliminate infinite loops

13

Deterministic PDA

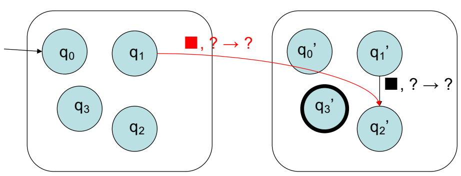
step 1: keep track of when we have read to end of input



14

Deterministic PDA

step 1: keep track of when we have read to end of input



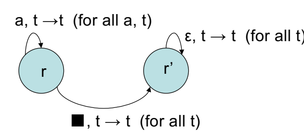
for accept state q': replace outgoing "ε, ? → ?" transition with self-loop with same label

15

Deterministic PDA

step 2: eliminate infinite loops

- add new "reject" states

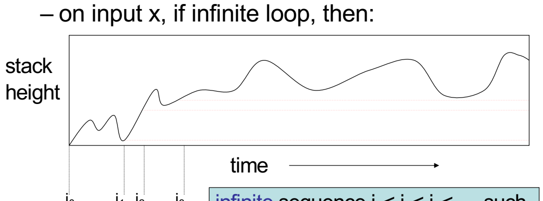


16

Deterministic PDA

step 2: eliminate infinite loops

- on input x, if infinite loop, then:



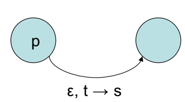
infinite sequence $i_0 < i_1 < i_2 < \ldots$ such that for all k, stack height never decreases below $ht(i_k)$ after time i_k

17

Deterministic PDA

step 2: eliminate infinite loops

- infinite seq. $i_0 < i_1 < \ldots$ such that for all k, stack height never decreases below $ht(i_k)$ after time i_k
- infinite subsequence $j_0 < j_1 < j_2 < \ldots$ such that same transition is applied at each time j_k



- never see any stack symbol below t from j_k on
- we are in a periodic, deterministic sequence of stack operations independent of the input

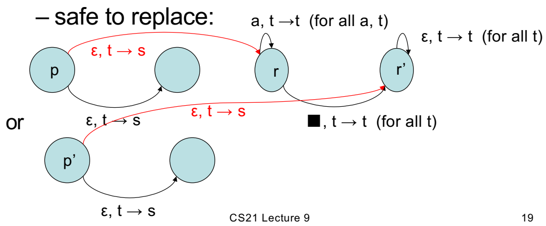
18

Deterministic PDA

step 2: eliminate infinite loops

– infinite subsequence $j_0 < j_1 < j_2 < \ldots$ such that same transition is applied at each time j_k

– safe to replace:



19

Deterministic PDA

– finishing up…

– have a machine M with no infinite loops

– therefore it always reads to end of input

– either enters an accept state q', or enters "reject" state r'

– now, can swap: make r' unique accept state to get a machine recognizing complement of L

20

Summary

- Nondeterministic Pushdown Automata (NPDA)
- Context-Free Grammars (CFGs) describe Context-Free Languages (CFLs)
  – terminals, non-terminals
  – productions
  – yields, derivations
  – parse trees

21

Summary

– grouping determined by grammar

– Chomsky Normal Form (CNF)

- NDPAs and CFGs are equivalent
- CFL Pumping Lemma is used to show certain languages are not CFLs

22

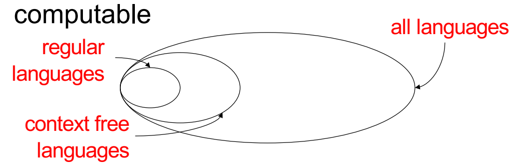
Summary

- deterministic PDAs recognize DCFLs
- DCFLs are closed under complement
- there is an efficient algorithm (based on dynamic programming) to determine if a string x is generated by a given grammar G

23

So far…

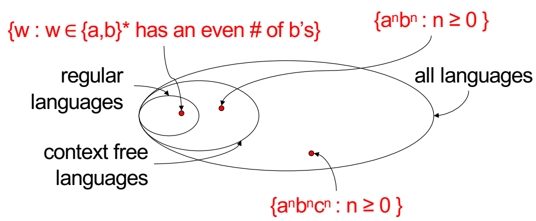
- several models of computation
  – finite automata
  – pushdown automata
- fail to capture our intuitive notion of what is computable



24

So far…

- We proved (using constructions of FA and NPDAs and the two pumping lemmas):



January 24, 2024 CS21 Lecture 9 25

25

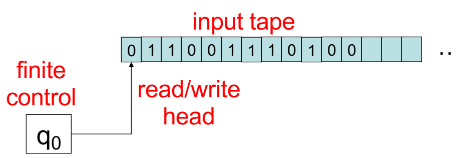
A more powerful machine

- limitation of NPDA related to fact that their memory is stack-based (last in, first out)
- What is the simplest alteration that adds general-purpose "memory" to our machine?
- Should be able to recognize, e.g., $\{a^nb^nc^n : n \geq 0\}$

January 24, 2024 CS21 Lecture 9 26

26

Turing Machines



- New capabilities:
  - infinite tape
  - can read OR write to tape
  - read/write head can move left and right

January 24, 2024 CS21 Lecture 9 27

27

Turing Machine

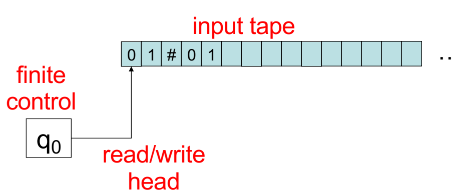
- Informal description:
  - input written on left-most squares of tape
  - rest of squares are blank
  - at each point, take a step determined by
    - current symbol being read
    - current state of finite control
  - a step consists of
    - writing new symbol
    - moving read/write head left or right
    - changing state

January 24, 2024 CS21 Lecture 9 28

28

Example Turing Machine

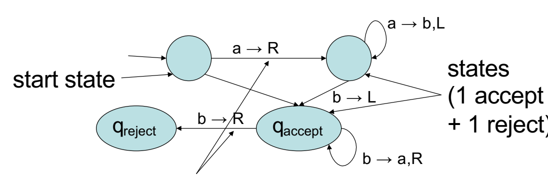
language L = {w#w : w ∈ {0,1}*}



January 24, 2024 CS21 Lecture 9 29

29

Turing Machine diagrams



transition label: (tape symbol read → tape symbol written, direction moved)

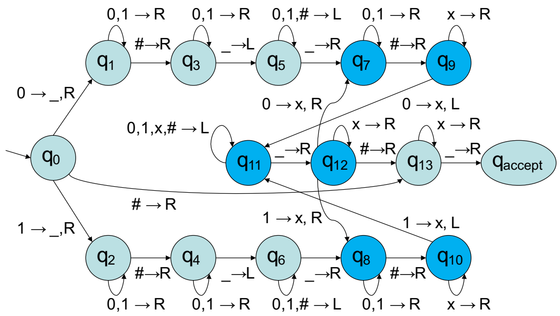
- a → R means "read a, move right"
- a → L means "read a, move left"
- a → b, R means "read a, write b, move right

"_" means blank tape square

January 24, 2024 CS21 Lecture 9 30

30

Example TM diagram



31