CS21
Decidability and Tractability

Lecture 7

January 22, 2018

Outline

- non context-free languages (continued)
- deterministic PDAs
- deciding CFLs

- Turing Machines and variants

Pumping Lemma for CFLs

CFL Pumping Lemma: Let L be a CFL.

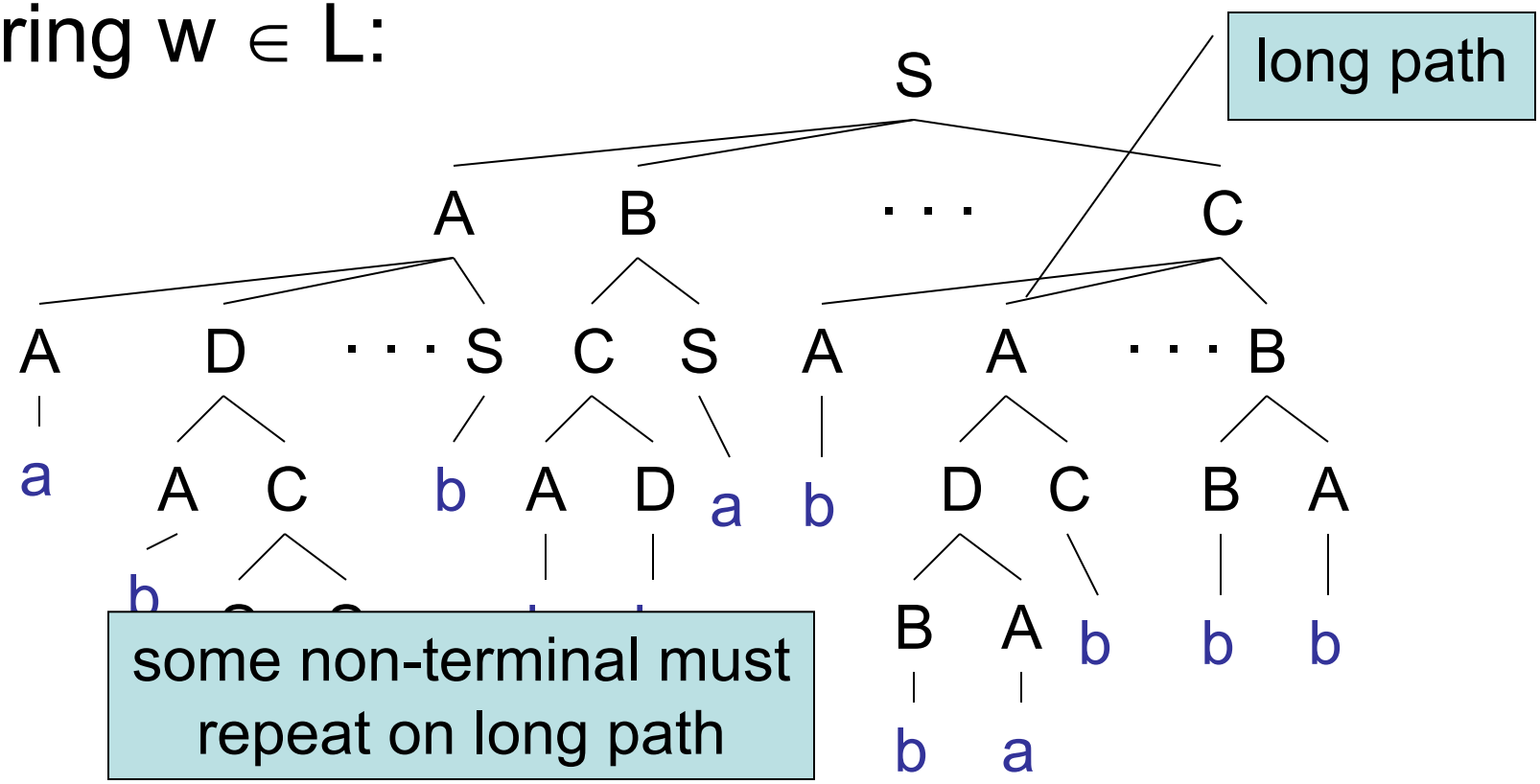
There exists an integer p ("pumping length") for which every $w \in L$ with $|w| \geq p$ can be written as

$w = uvxyz$ such that

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

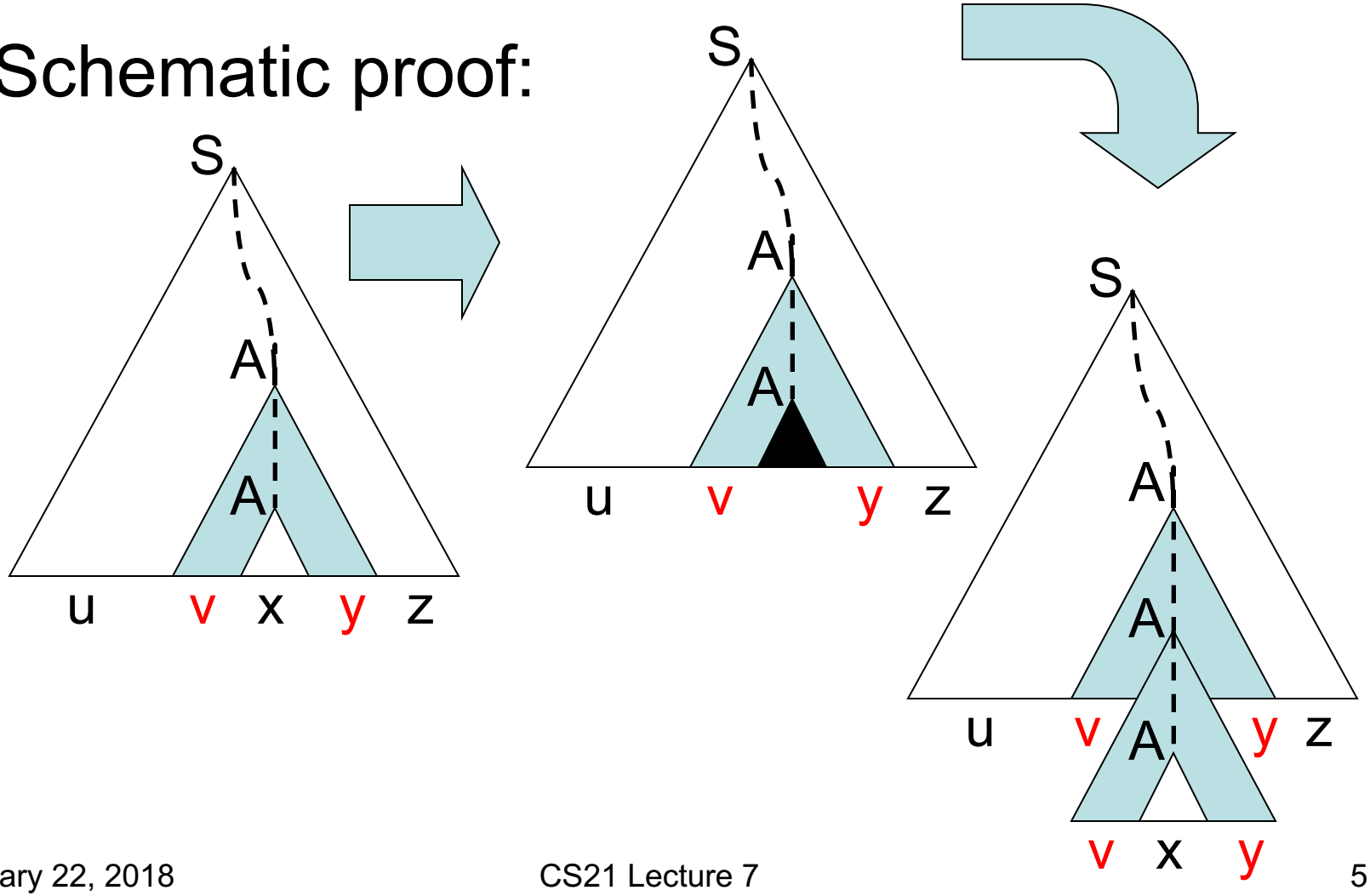
CFL Pumping Lemma

Proof: consider a parse tree for a very long string w ∈ L:



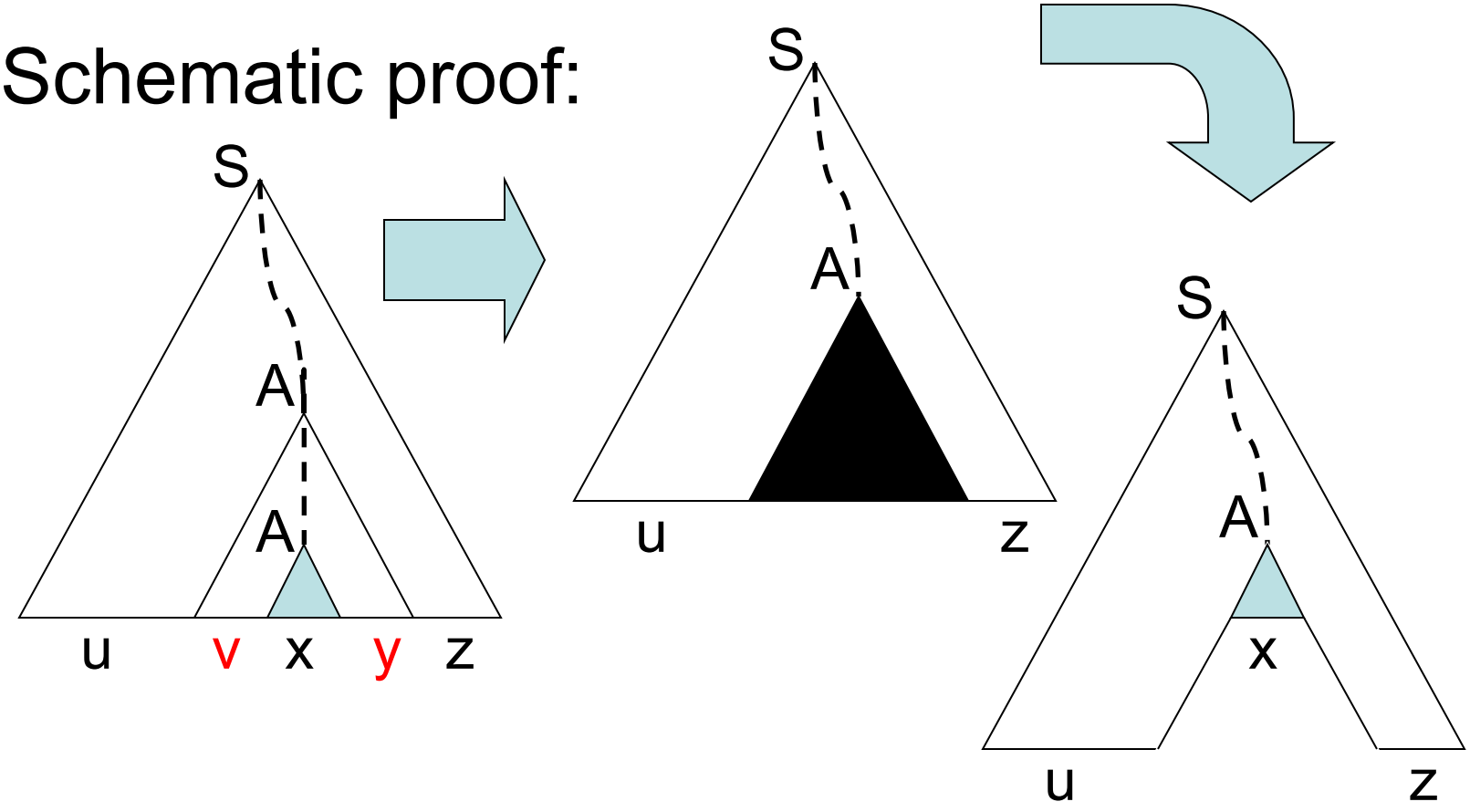
CFL Pumping Lemma

- Schematic proof:



CFL Pumping Lemma

- Schematic proof:



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 ≤ h, then string generated has length $\leq b^h$ (so length $> b^h$ implies height > h)

CFL Pumping Lemma

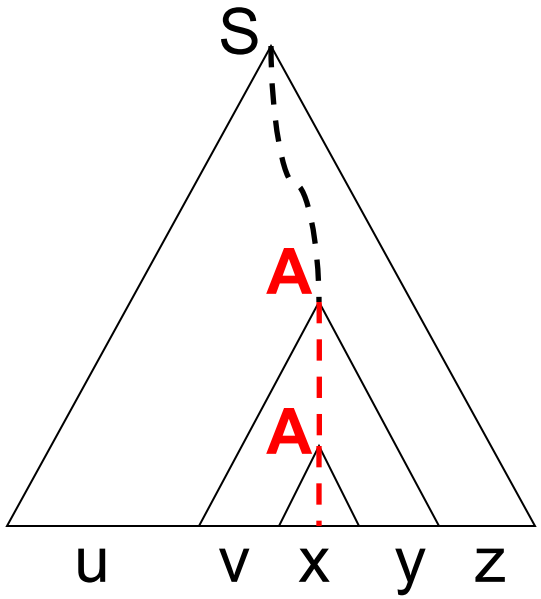
- let m be the # of nonterminals in the grammar
- to ensure path of length at least m+2, require

$$|w| \geq \mathbf{p} = b^{m+2}$$

- 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 ≥ m+1 non-terminals and 1 terminal.

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 ≤ m+2, so ≤ $b^{m+2} = p$ leaves

Deterministic PDA

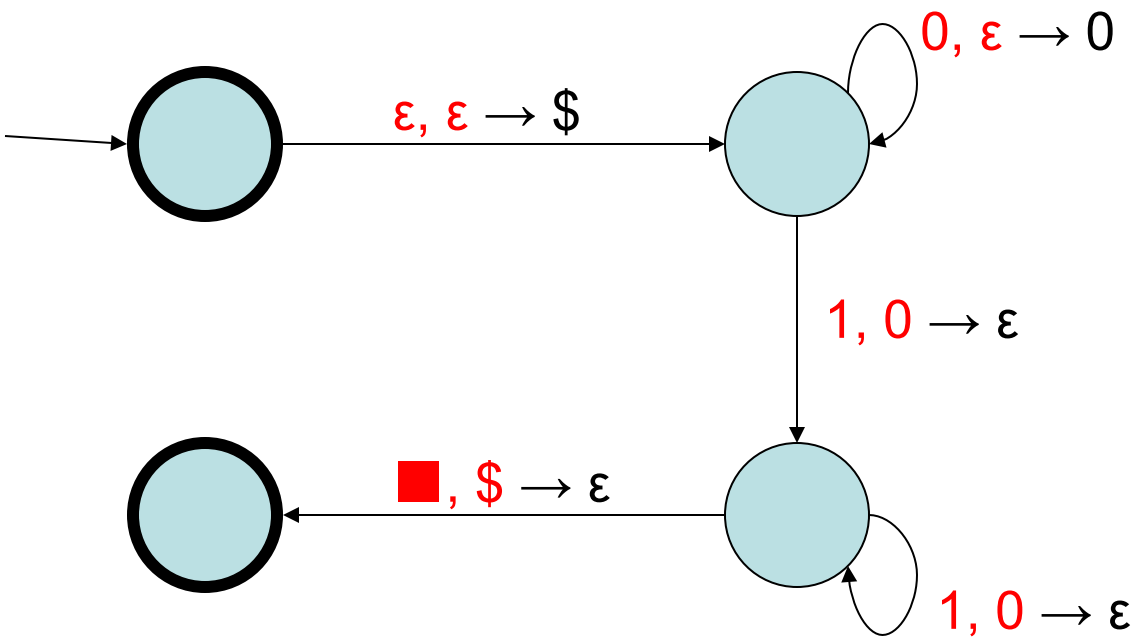
- A NPDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where:
  - $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow \wp(Q \times (\Gamma \cup \{\varepsilon\}))$ is a function called the transition function
- A deterministic PDA has only one option at 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 $a \in \Sigma$

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)

Example deterministic PDA



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

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

$$L = \{0^n 1^n : n \geq 0\}$$

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

Deterministic PDA

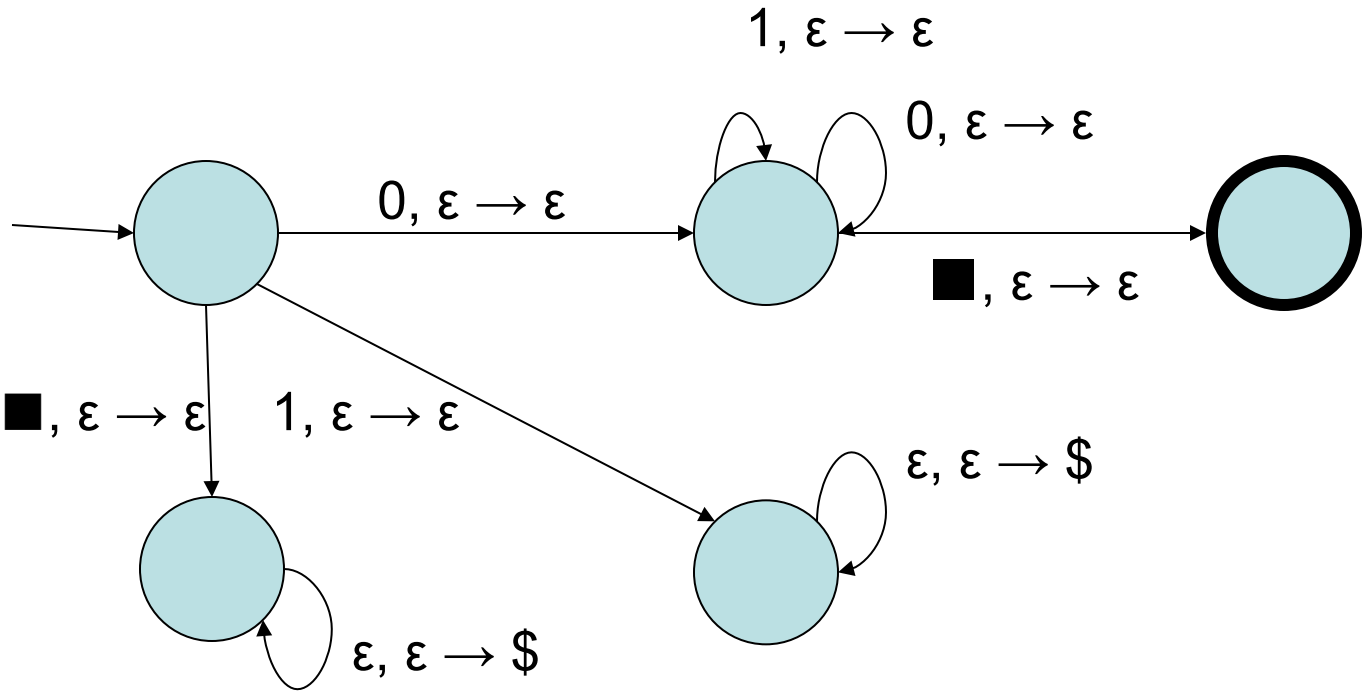
Theorem: DCFLs are closed under complement

(complement of L in Σ^* is (Σ^* - L))

Proof attempt:

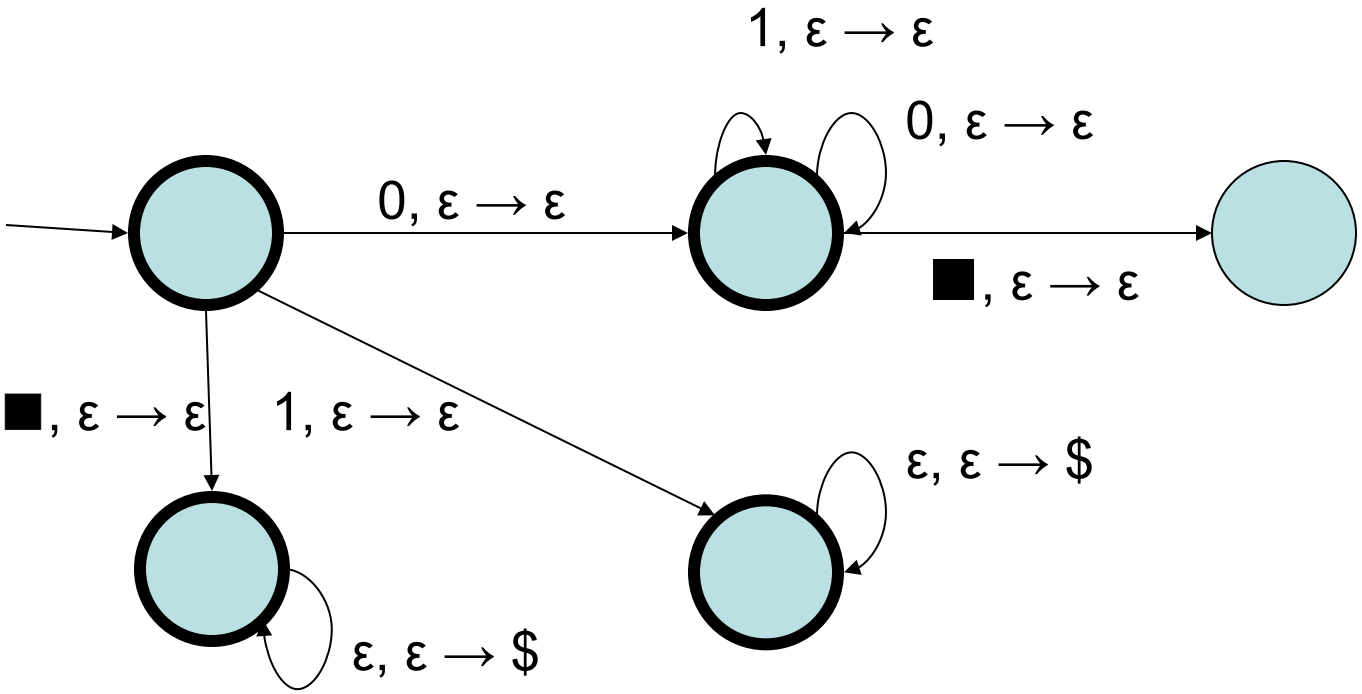
- swap accept/non-accept states
- problem: might enter infinite loop before reading entire string
- machine for complement must accept in these cases, and read to end of string

Example of problem



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

Example of problem



Language of this DPDA is {ε}

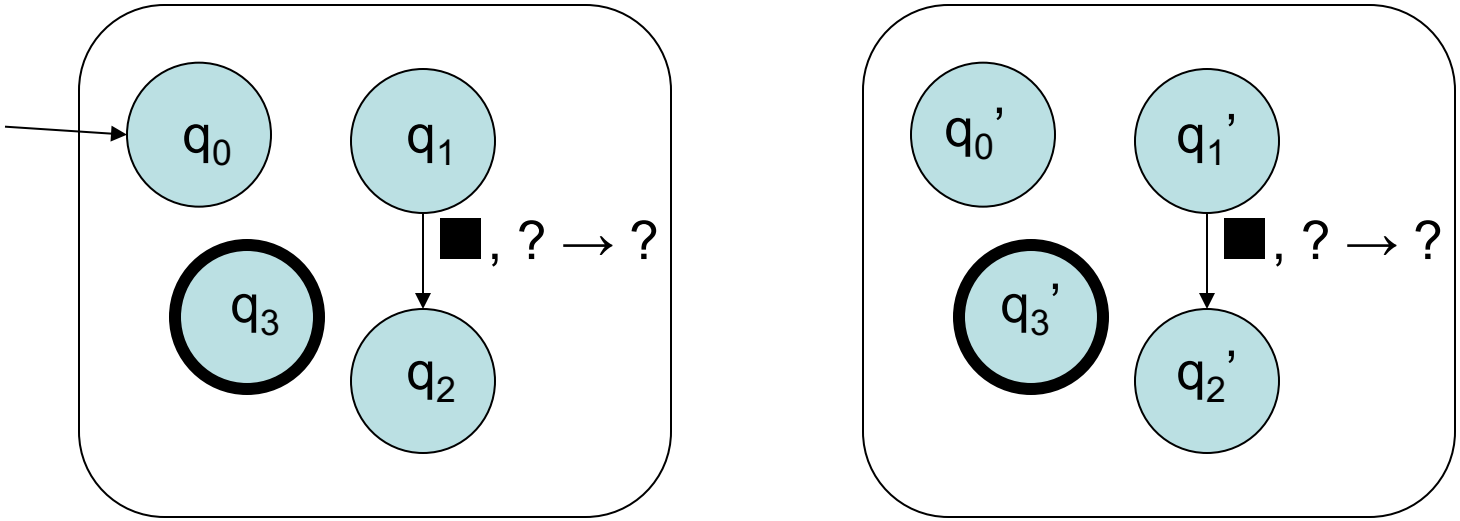
Deterministic PDA

Proof:

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

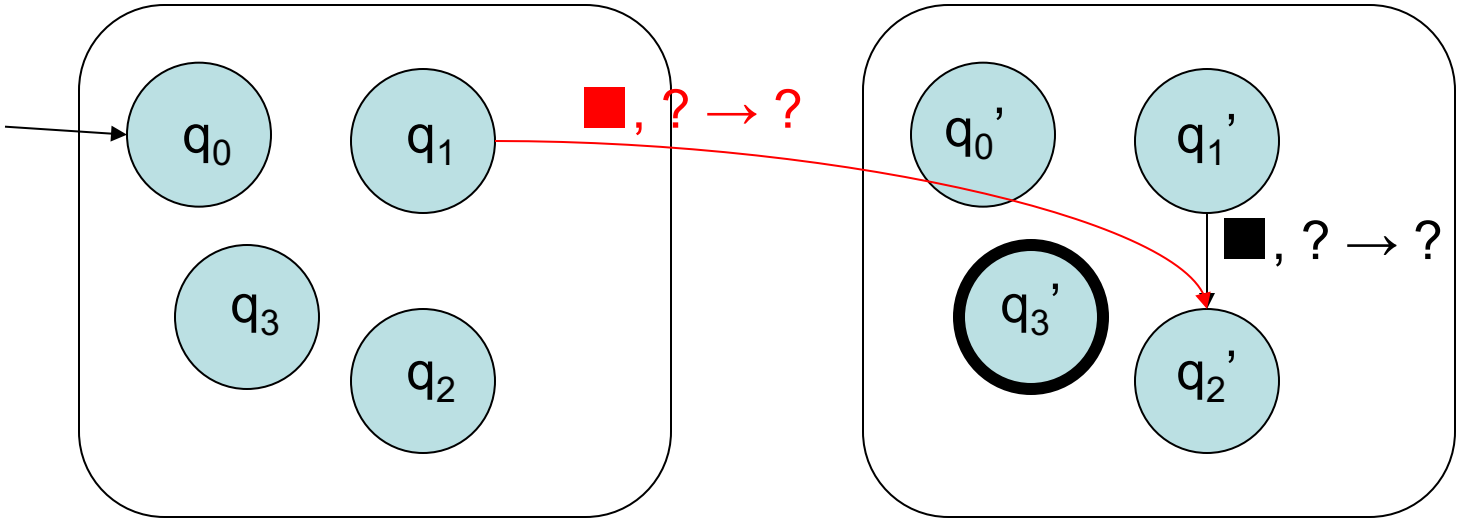
Deterministic PDA

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



Deterministic PDA

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

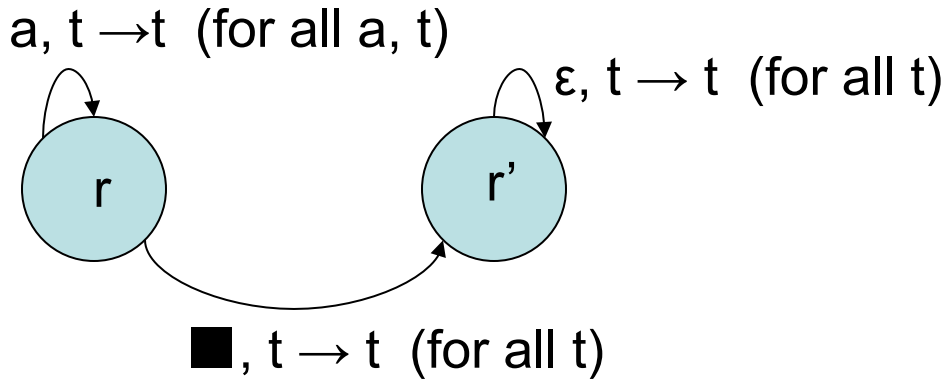


for accept state q': replace outgoing "ε, ? → ?" transition with self-loop with same label

Deterministic PDA

step 2: eliminate infinite loops

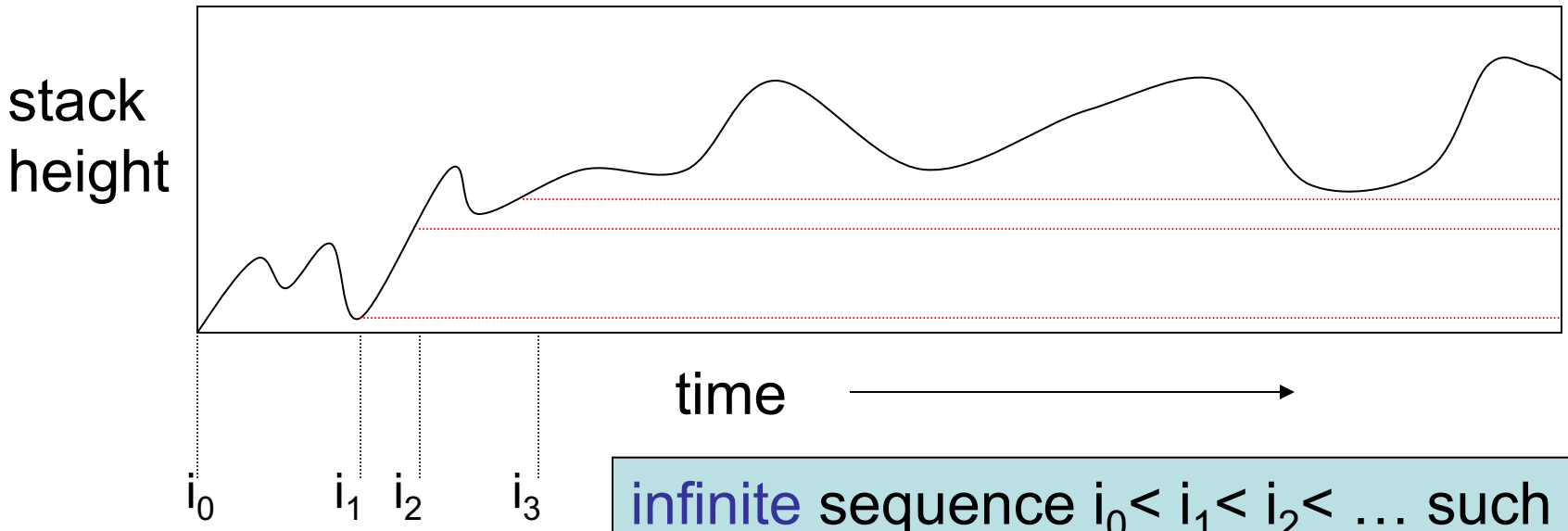
– add new “reject” states



Deterministic PDA

step 2: eliminate infinite loops

– on input x, if infinite loop, then:



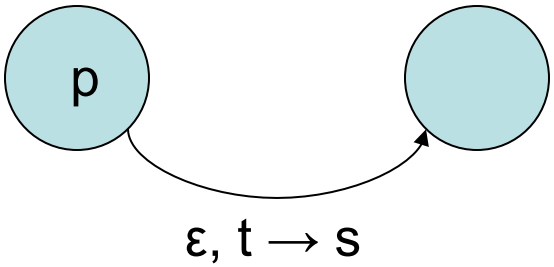
infinite sequence $i_0 < i_1 < i_2 < \ldots$ such that for all k, stack height never decreases below $ht(i_k)$ after time i_k

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

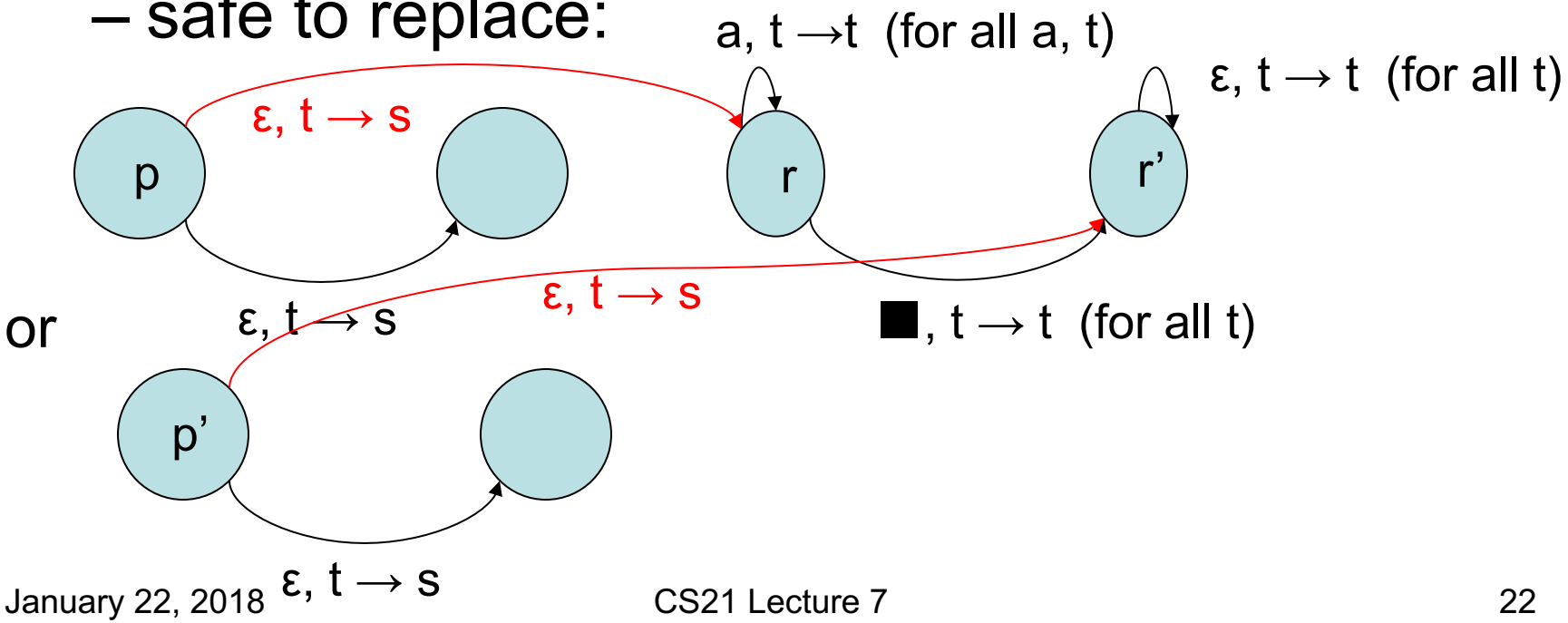
• we are in a periodic, deterministic sequence of stack operations independent of the input

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:



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'

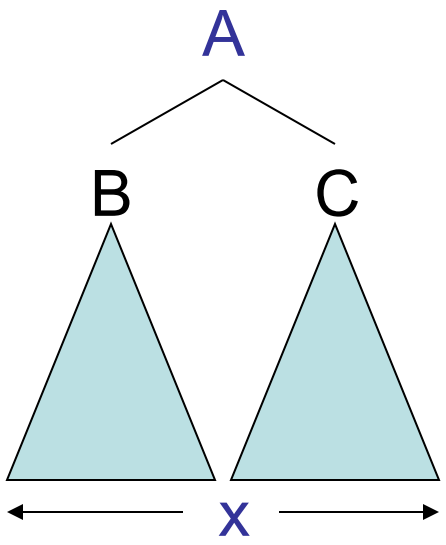
– now, can swap: make r' unique accept state to get a machine recognizing complement of L

Deciding CFLs

- Useful to have an efficient algorithm to decide whether string x is in given CFL
  - e.g. programming language often described by CFG. Determine if string is valid program.
- If CFL recognized by deterministic PDA, just simulate the PDA.
  - but not all CFLs are (homework)…
- Can simulate NPDA, but this takes exponential time in the worst case.

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 $i, j < k$

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?

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 $A \Rightarrow^* x[i,j]$}

Deciding CFLs

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

for i = 1 to n

T[i, i] = {A: "A $\rightarrow x_i$" 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 $\rightarrow$ BC" is a production in G and B $\in$ T[i,i+j] and C $\in$ T[i+j+1,i+k] }

output "YES" if S $\in$ T[1, n], else output "NO"

Deciding CFLs

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

for i = 1 to n

T[i, i] = {A: "$A \rightarrow x_i$" 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: "$A \rightarrow BC$" is a production in G and $B \in T[i,i+j]$ and $C \in T[i+j+1,i+k]$ }

output "YES" if $S \in T[1, n]$, else output "NO"

O(nm) steps

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

Summary

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

Summary

- grouping determined by grammar
- ambiguity
- Chomsky Normal Form (CNF)

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

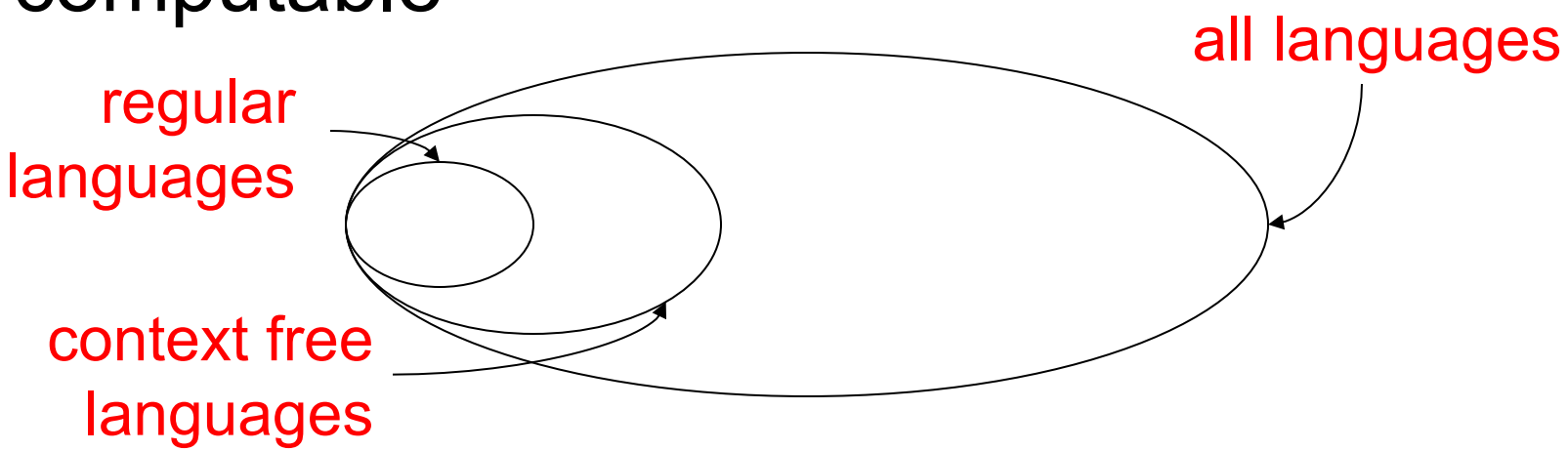
Summary

- 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

So far…

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



So far…

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