

CS21 Decidability and Tractability

Lecture 7
January 23, 2008

Outline

- deterministic PDAs
- deciding CFLs
- on to Turing Machines...

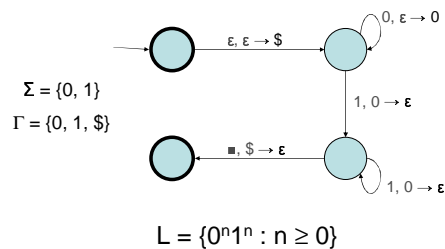
Deterministic PDA

- A NPDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where:
 - $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \rightarrow \wp(Q \times (\Gamma \cup \{\epsilon\}))$ is a function called the transition function
- A deterministic PDA has only one option at every step:
 - for every state $q \in Q$, $a \in (\Sigma \cup \{\epsilon\})$, and $t \in \Gamma$, at most 1 element in $\delta(q, a, t)$
 - if $\delta(q, \epsilon, t)$ not empty, then $\delta(q, a, t)$ empty for all $a \in \Sigma$

Deterministic PDA

- A detail: a NPDA can “guess” the end of input string
- to be fair, we should give our deterministic machine ability to detect end of input string
 - add special symbol \blacksquare to alphabet
 - require input tape to contain $x\blacksquare$
- language recognized by a deterministic PDA is called a deterministic CFL (DCFL)

Example deterministic PDA

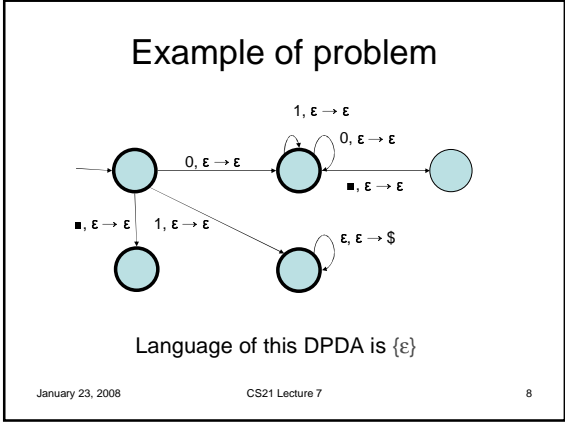
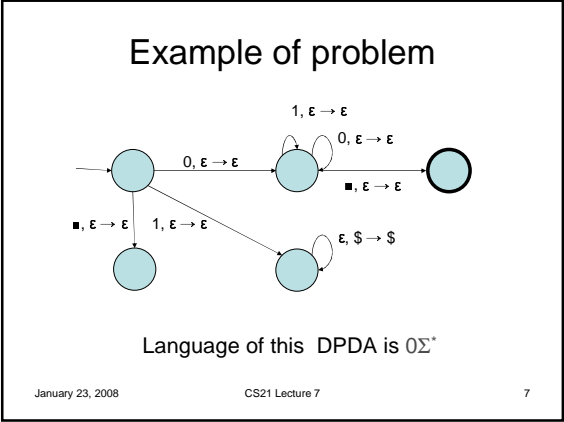


Deterministic PDA

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

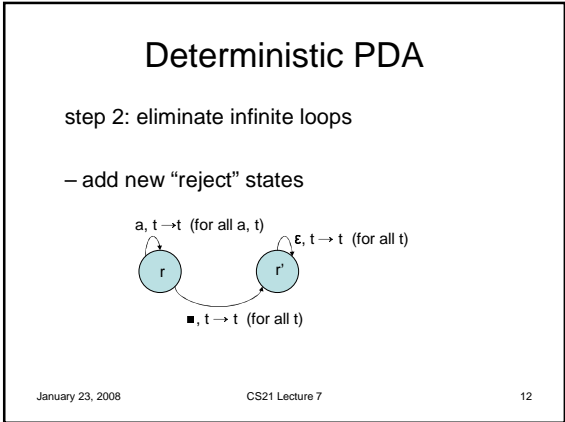
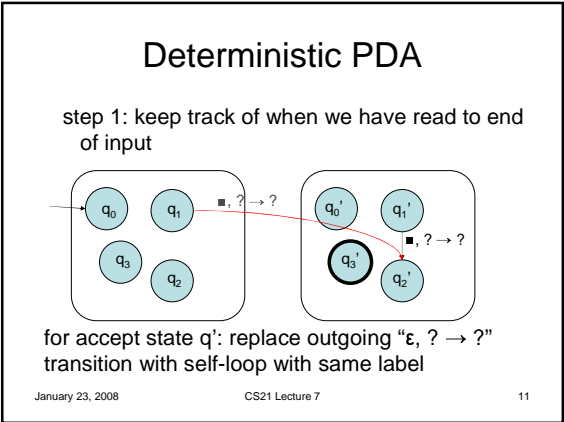
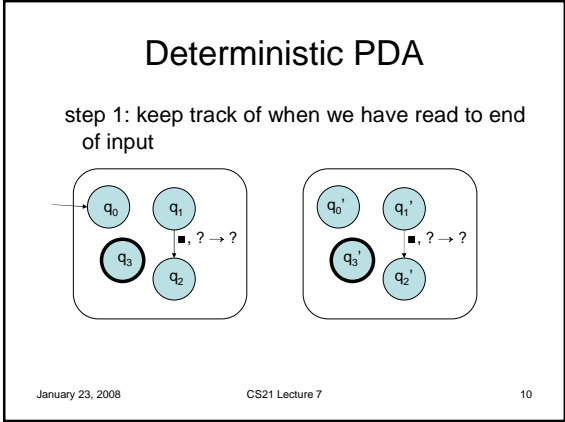


Deterministic PDA

Proof:

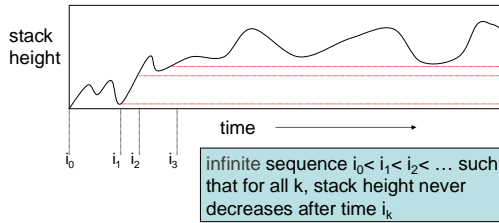
- convert machine into "normal form"
 - always reads to end of input
 - always enters either an accept state or single distinguished "reject" state
- step 1: keep track of when we have read to end of input
- step 2: eliminate infinite loops

January 23, 2008 CS21 Lecture 7 9



Deterministic PDA

- step 2: eliminate infinite loops
 – on input x , if infinite loop, then:



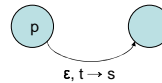
January 23, 2008

CS21 Lecture 7

13

Deterministic PDA

- step 2: eliminate infinite loops
 – infinite sequence $i_0 < i_1 < i_2 < \dots$ such that for all k , stack height never decreases after time i_k
 – infinite subsequence $j_0 < j_1 < j_2 < \dots$ 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

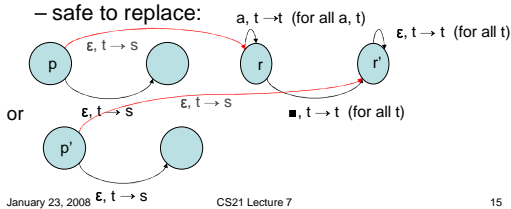
January 23, 2008

CS21 Lecture 7

14

Deterministic PDA

- step 2: eliminate infinite loops
 – infinite subsequence $j_0 < j_1 < j_2 < \dots$ such that same transition is applied at each time j_k
 – safe to replace:



January 23, 2008

CS21 Lecture 7

15

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

January 23, 2008

CS21 Lecture 7

16

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.

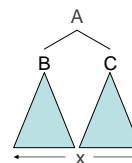
January 23, 2008

CS21 Lecture 7

17

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 $i, j < k$

January 23, 2008

CS21 Lecture 7

18

Deciding CFLs

- An algorithm:
IsGenerated(x, A)
 if $|x| = 1$, then return YES if $A \rightarrow x$ is a production,
 else return NO
 for all $n-1$ ways of splitting $x = yz$
 for all $\leq m$ productions of form $A \rightarrow BC$
 if $\text{IsGenerated}(y, B)$ and $\text{IsGenerated}(z, C)$,
 return YES
 return NO
- worst case running time?

January 23, 2008

CS21 Lecture 7

19

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

January 23, 2008

CS21 Lecture 7

20

Deciding CFLs

IsGenerated(x = $x_1x_2x_3\dots x_n$, G)
 for $i = 1$ to n
 $T[i, i] = \{A: "A \rightarrow x_i" \text{ is a production in } G\}$
 for $k = 1$ to $n - 1$
 for $i = 1$ to $n - k$
 for k splittings $x[i, i+k] = x[i, i+j]x[i+j+1, i+k]$
 $T[i, i+k] = \{A: "A \rightarrow BC" \text{ is a production in } G \text{ and } B \in T[i, i+j] \text{ and } C \in T[i+j+1, i+k]\}$
 output "YES" if $S \in T[1, n]$, else output "NO"

January 23, 2008

CS21 Lecture 7

21

Deciding CFLs

IsGenerated(x = $x_1x_2x_3\dots x_n$, G) $O(nm)$ steps
 for $i = 1$ to n
 $T[i, i] = \{A: "A \rightarrow x_i" \text{ is a production in } G\}$
 for $k = 1$ to $n - 1$
 for $i = 1$ to $n - k$
 for k splittings $x[i, i+k] = x[i, i+j]x[i+j+1, i+k]$
 $T[i, i+k] = \{A: "A \rightarrow BC" \text{ is a production in } G \text{ and } B \in T[i, i+j] \text{ and } C \in T[i+j+1, i+k]\}$
 output "YES" if $S \in T[1, n]$, else output "NO"
 $O(n^3m^2)$ steps

January 23, 2008

CS21 Lecture 7

22

Summary

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

January 23, 2008

CS21 Lecture 7

23

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

January 23, 2008

CS21 Lecture 7

24

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

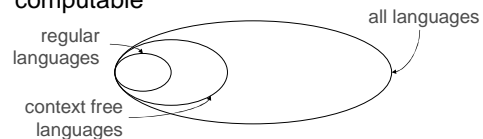
January 23, 2008

CS21 Lecture 7

25

So far...

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



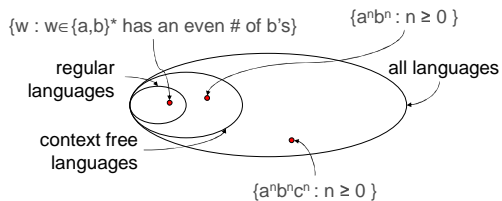
January 23, 2008

CS21 Lecture 7

26

So far...

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



January 23, 2008

CS21 Lecture 7

27

A more powerful machine

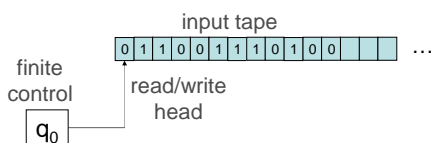
- limitation of NPDA related to fact that their memory is stack-based (last in, first out)
- What is the simplest alteration that adds general-purpose “memory” to our machine?
- Should be able to recognize, e.g., $\{a^n b^n c^n : n \geq 0\}$

January 23, 2008

CS21 Lecture 7

28

Turing Machines



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

January 23, 2008

CS21 Lecture 7

29