

CS21 Decidability and Tractability

Lecture 4
January 14, 2008

Outline

- Pumping Lemma proof
- Pushdown Automata
- Context-Free Grammars and Languages

Non-regular languages

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

$$w = xyz \quad \text{such that}$$

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

Non-regular languages

- Using the Pumping Lemma to prove L is not regular:
 - assume L is regular
 - then there exists a pumping length p
 - select a string $w \in L$ of length at least p
 - argue that for every way of writing $w = xyz$ that satisfies (2) and (3) of the Lemma, pumping on y yields a string not in L .
 - contradiction.

Pumping Lemma Examples

- Theorem: $L = \{w : w \text{ has an equal \# of 0s and 1s}\}$ is not regular.

- Proof:

- let p be the pumping length for L
- choose $w = 0^p 1^p$

$$w = \underbrace{000000000}_{p} \dots \underbrace{011111111}_{p} \dots 1$$

- $w = xyz$, with $|y| > 0$ and $|xy| \leq p$.

Pumping Lemma Examples

- 3 possibilities:

$$w = \underbrace{000000000}_x \dots \underbrace{011111111}_{y} \dots \underbrace{1}_{z}$$

$$w = \underbrace{000000000}_x \dots \underbrace{011111111}_{y} \dots \underbrace{1}_{z}$$

$$w = \underbrace{000000000}_x \dots \underbrace{011111111}_{y} \dots \underbrace{1}_{z}$$

- first 2 cases, pumping on y gives a string not in language L ; 3rd case a problem!

Pumping Lemma Examples

- recall condition 3: $|xy| \leq p$
- since $w = 0^p 1^p$ we know more about how it can be divided, and this case cannot arise:

$$w = \underbrace{000000000}_{x} \dots \underbrace{011111111}_{y} \dots \underbrace{1}_{z}$$

- so we do get a contradiction.
- conclude that L is not regular.

January 14, 2008

CS21 Lecture 4

7

Pumping Lemma Examples

- Theorem: $L = \{0^i 1^j : i > j\}$ is not regular.
- Proof:
 - let p be the pumping length for L
 - choose $w = 0^{p+1} 1^p$

$$w = \underbrace{000000000}_{p+1} \dots \underbrace{011111111}_{p} \dots 1$$

- $w = xyz$, with $|y| > 0$ and $|xy| \leq p$.

January 14, 2008

CS21 Lecture 4

8

Pumping Lemma Examples

- 1 possibility:

$$w = \underbrace{000000000}_{x} \dots \underbrace{011111111}_{y} \dots \underbrace{1}_{z}$$

- pumping on y gives strings in the language (?)
- this seems like a problem...
- Lemma states that for every $i \geq 0$, $xy^i z \in L$
- $xy^0 z$ not in L. So L not regular.

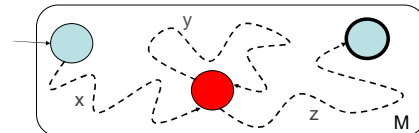
January 14, 2008

CS21 Lecture 4

9

Proof of the Pumping Lemma

- Let M be a FA that recognizes L.
- Set $p =$ number of states of M .
- Consider $w \in L$ with $|w| \geq p$. On input w , M must go through *at least* $p+1$ states. There must be a repeated state (among first $p+1$).



January 14, 2008

CS21 Lecture 4

10

FA Summary

- A "problem" is a language
- A "computation" receives an input and either accepts, rejects, or loops forever.
- A "computation" recognizes a language (it may also decide the language).
- Finite Automata perform simple computations that read the input from left to right and employ a finite memory.

January 14, 2008

CS21 Lecture 4

11

FA Summary

- The languages recognized by FA are the regular languages.
- The regular languages are closed under union, concatenation, and star.
- Nondeterministic Finite Automata may have several choices at each step.
- NFAs recognize exactly the same languages that FAs do.

January 14, 2008

CS21 Lecture 4

12

FA Summary

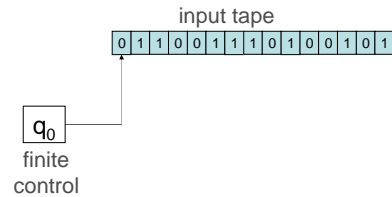
- Regular expressions are languages built up from the operations union, concatenation, and star.
- Regular expressions describe exactly the same languages that FAs (and NFAs) recognize.
- Some languages are not regular. This can be proved using the Pumping Lemma.

January 14, 2008

CS21 Lecture 4

13

Machine view of FA

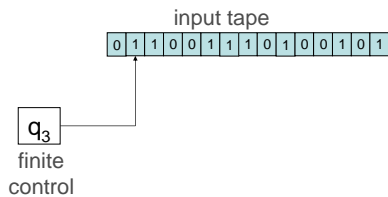


January 14, 2008

CS21 Lecture 4

14

Machine view of FA

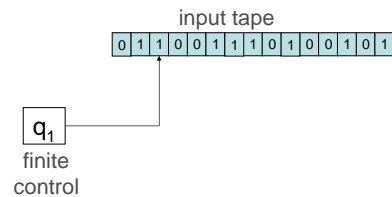


January 14, 2008

CS21 Lecture 4

15

Machine view of FA

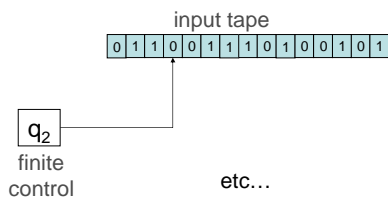


January 14, 2008

CS21 Lecture 4

16

Machine view of FA



January 14, 2008

CS21 Lecture 4

17

A more powerful machine

- limitation of FA related to fact that they can only "remember" a bounded amount of information
- What is the simplest alteration that adds unbounded "memory" to our machine?
- Should be able to recognize, e.g., $\{0^n1^n : n \geq 0\}$

January 14, 2008

CS21 Lecture 4

18

Pushdown Automata

finite control

q_0

input tape

0 1 1 0 0 1 1 1 0 1 0 0 1 0 1

(infinite) stack

0
1
1
0
⋮

New capabilities:

- can push symbol onto stack
- can pop symbol off of stack

January 14, 2008
CS21 Lecture 4
19

Pushdown Automata

finite control

q_0

input tape

0 0 1 1 0 1 1 1 0 1 0 0 1 0 1

(infinite) stack

\$
⋮

January 14, 2008
CS21 Lecture 4
20

Pushdown Automata

finite control

q_1

input tape

0 0 1 1 0 1 1 1 0 1 0 0 1 0 1

(infinite) stack

0
\$
⋮

January 14, 2008
CS21 Lecture 4
21

Pushdown Automata

finite control

q_1

input tape

0 0 1 1 0 1 1 1 0 1 0 0 1 0 1

(infinite) stack

0
0
\$
⋮

January 14, 2008
CS21 Lecture 4
22

Pushdown Automata

finite control

q_2

input tape

0 0 1 1 0 1 1 1 0 1 0 0 1 0 1

(infinite) stack

0
\$
⋮

January 14, 2008
CS21 Lecture 4
23

Pushdown Automata

finite control

q_2

input tape

0 0 1 1 0 1 1 1 0 1 0 0 1 0 1

(infinite) stack

\$
⋮

Note: often start by pushing \$ marker onto stack so that we can detect "empty stack"

January 14, 2008
CS21 Lecture 4
24

Pushdown Automata (PDA)

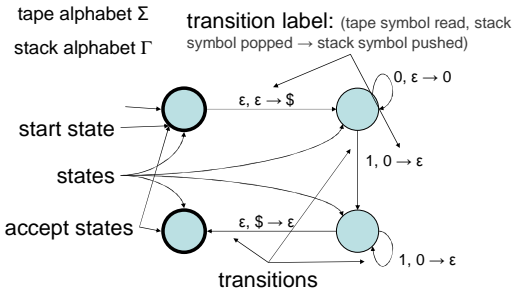
- We will define nondeterministic pushdown automata immediately
 - potentially several choices of “next step”
- Deterministic PDA defined later
 - weaker than NPDA
- Two ways to describe NPDA
 - diagram
 - formal definition

January 14, 2008

CS21 Lecture 4

25

NPDA diagram



January 14, 2008

CS21 Lecture 4

26

NPDA operation

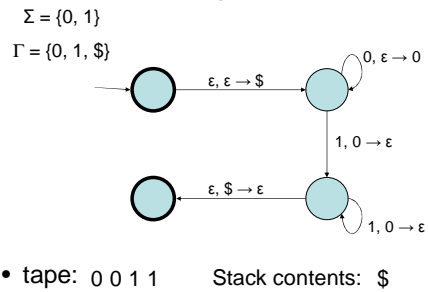
- Taking a transition labeled: $a, b \rightarrow c$
 - $a \in (\Sigma \cup \{\epsilon\})$
 - $b, c \in (\Gamma \cup \{\epsilon\})$
 - read a from tape, or don't read from tape if $a = \epsilon$
 - pop b from stack, or don't pop from stack if $b = \epsilon$
 - push c onto stack, or don't push onto stack if $c = \epsilon$

January 14, 2008

CS21 Lecture 4

27

Example NPDA

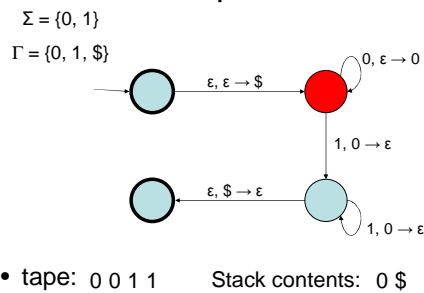


January 14, 2008

CS21 Lecture 4

28

Example NPDA

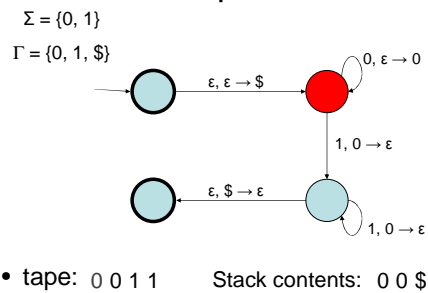


January 14, 2008

CS21 Lecture 4

29

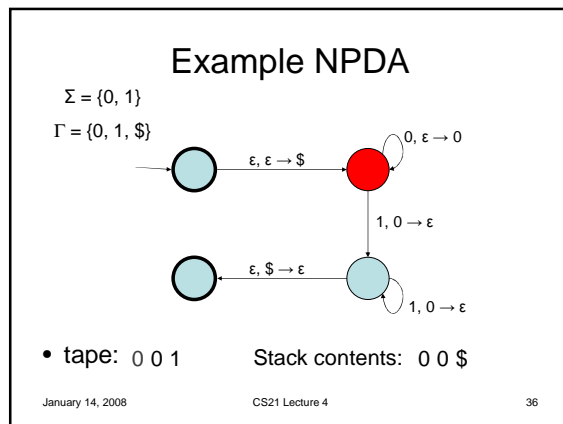
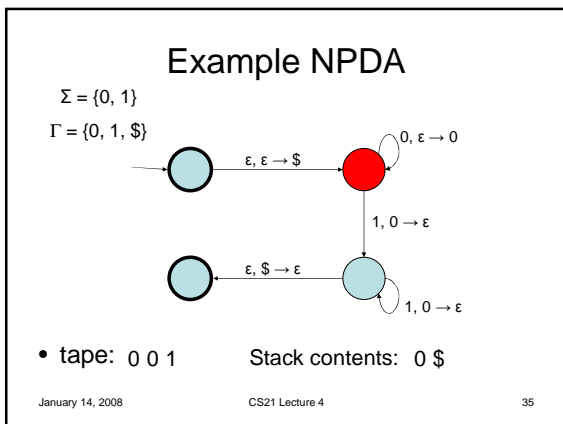
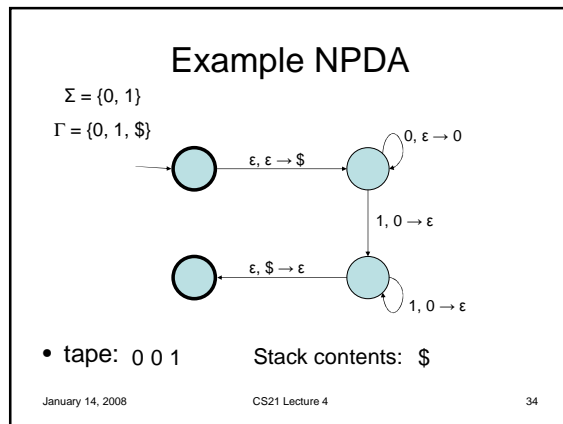
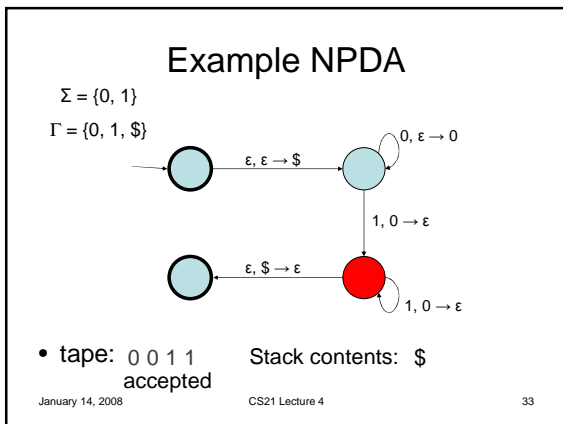
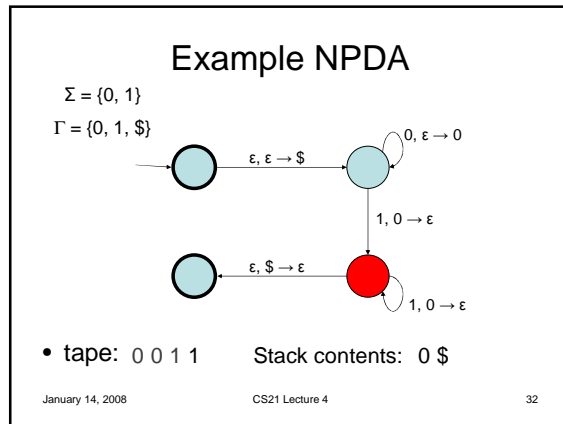
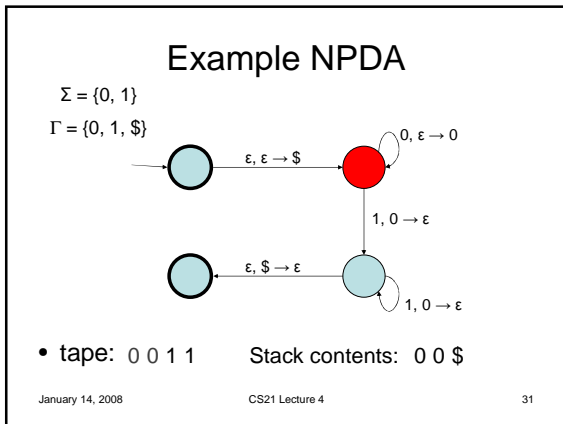
Example NPDA

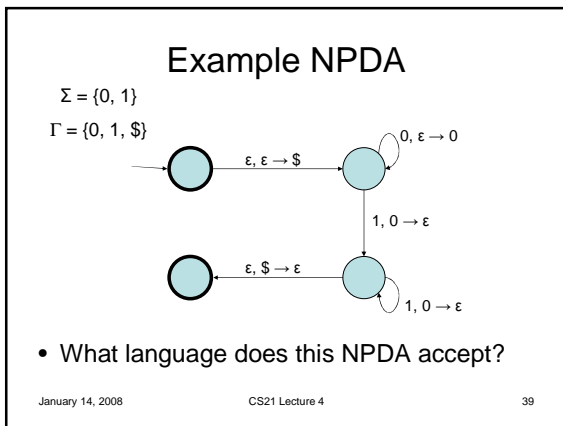
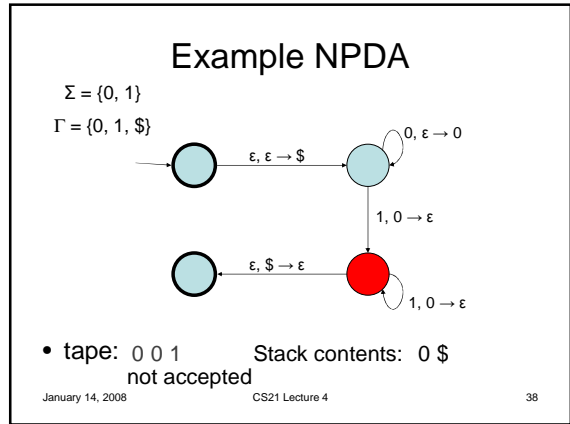
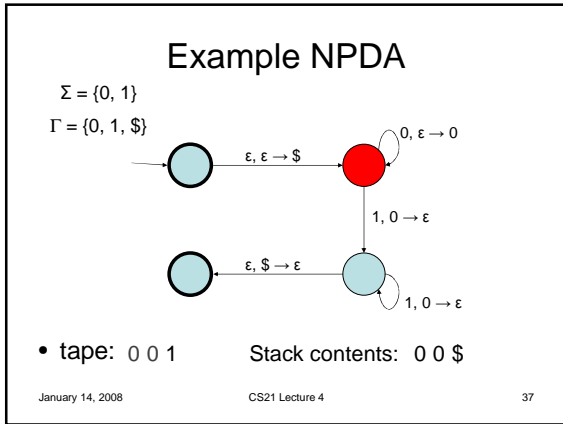


January 14, 2008

CS21 Lecture 4

30





Formal definition of NPDA

• A NPDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where:

- Q is a finite set called the states
- Σ is a finite set called the tape alphabet
- Γ is a finite set called the stack alphabet
- $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \rightarrow \wp(Q \times (\Gamma \cup \{\epsilon\}))$ is a function called the transition function
- q_0 is an element of Q called the start state
- F is a subset of Q called the accept states

January 14, 2008 CS21 Lecture 4 40

Formal definition of NPDA

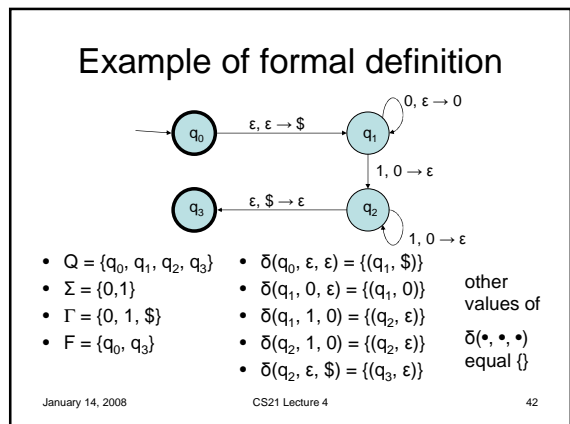
• NPDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ accepts string $w \in \Sigma^*$ if w can be written as $w_1 w_2 w_3 \dots w_m \in (\Sigma \cup \{\epsilon\})^*$, and

• there exist states $r_0, r_1, r_2, \dots, r_m$, and

• there exist strings s_0, s_1, \dots, s_m in $(\Gamma \cup \{\epsilon\})^*$

- $r_0 = q_0$ and $s_0 = \epsilon$
- $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where $s_i = at$, $s_{i+1} = bt$ for some $t \in \Gamma^*$
- $r_m \in F$

January 14, 2008 CS21 Lecture 4 41



Exercise

Design a NPDA for the language

$\{a^i b^j c^k : i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$

January 14, 2008

CS21 Lecture 4

43

Context-free grammars and languages

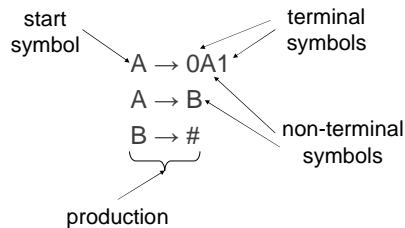
- languages recognized by a (N)FA are exactly the languages described by regular expressions, and they are called the regular languages
- languages recognized by a NPDA are exactly the languages described by context-free grammars, and they are called the context-free languages

January 14, 2008

CS21 Lecture 4

44

Context-Free Grammars



January 14, 2008

CS21 Lecture 4

45

Context-Free Grammars

- generate strings by repeated replacement of non-terminals with string of terminals and non-terminals
 - write down start symbol (non-terminal)
 - replace a non-terminal with the right-hand-side of a rule that has that non-terminal as its left-hand-side.
 - repeat above until no more non-terminals

January 14, 2008

CS21 Lecture 4

46

Context-Free Grammars

Example:

$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$

$A \rightarrow 0A1$
 $A \rightarrow B$
 $B \rightarrow \#$

- a derivation of the string 000#111
- set of all strings generated in this way is the language of the grammar $L(G)$
- called a Context-Free Language

January 14, 2008

CS21 Lecture 4

47

Context-Free Grammars

- Natural languages (e.g. English) shorthand for multiple rules with same lhs
 - $\langle \text{sentence} \rangle \rightarrow \langle \text{noun-phrase} \rangle \langle \text{verb-phrase} \rangle$
 - $\langle \text{noun-phrase} \rangle \rightarrow \langle \text{cpx-noun} \rangle \mid \langle \text{cpx-noun} \rangle \langle \text{prep-phrase} \rangle$
 - $\langle \text{verb-phrase} \rangle \rightarrow \langle \text{cpx-verb} \rangle \mid \langle \text{cpx-verb} \rangle \langle \text{prep-phrase} \rangle$
 - $\langle \text{prep-phrase} \rangle \rightarrow \langle \text{prep} \rangle \langle \text{cpx-noun} \rangle$
 - $\langle \text{cpx-noun} \rangle \rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle$
 - $\langle \text{cpx-verb} \rangle \rightarrow \langle \text{verb} \rangle \mid \langle \text{verb} \rangle \langle \text{noun-phrase} \rangle$
 - $\langle \text{article} \rangle \rightarrow a \mid the$
 - $\langle \text{noun} \rangle \rightarrow dog \mid cat \mid flower$
 - $\langle \text{verb} \rangle \rightarrow eats \mid sees$
 - $\langle \text{prep} \rangle \rightarrow with$
- Generate a string in the language of this grammar.

January 14, 2008

CS21 Lecture 4

48

Context-Free Grammars

- CFGs don't capture natural languages completely
- computer languages often defined by CFG
 - hierarchical structure
 - slightly different notation often used “Backus-Naur form”
 - see next slide for example

January 14, 2008

CS21 Lecture 4

49

Example CFG

```

<stmt> → <if-stmt> | <while-stmt> | <begin-stmt>
                                     | <asgn-stmt>
<if-stmt> → IF <bool-expr> THEN <stmt> ELSE <stmt>
<while-stmt> → WHILE <bool-expr> DO <stmt>
<begin-stmt> → BEGIN <stmt-list> END
<stmt-list> → <stmt> | <stmt>; <stmt-list>
<asgn-stmt> → <var> := <arith-expr>
<bool-expr> → <arith-expr><compare-op><arith-expr>
<compare-op> → <|> | <|≤|> | <|≥|> | <|=|>
<arith-expr> → <var> | <const>
                                     | (<arith-expr><arith-op><arith-expr>)
<arith-op> → + | - | * | /
<const> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<var> → a | b | c | ... | x | y | z
    
```

January 14, 2008

CS21 Lecture 4

50

CFG formal definition

- A context-free grammar is a 4-tuple (V, Σ, R, S)
 - V is a finite set called the non-terminals
 - Σ is a finite set (disjoint from V) called the terminals
 - R is a finite set of productions where each production is a non-terminal and a string of terminals and non-terminals.
 - $S \in V$ is the start variable (or start non-terminal)
- where

January 14, 2008

CS21 Lecture 4

51

CFG formal definition

- u, v, w are strings of non-terminals and terminals, and $A \rightarrow w$ is a production:
 - “ uAv yields uwv ” notation: $uAv \Rightarrow uwv$
 - also: “yields in 1 step” notation: $uAv \Rightarrow^1 uwv$
- in general:
 - “yields in k steps” notation: $u \Rightarrow^k v$
 - meaning: there exists strings u_1, u_2, \dots, u_{k-1} for which $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \dots \Rightarrow u_{k-1} \Rightarrow v$

January 14, 2008

CS21 Lecture 4

52

CFG formal definition

- notation: $u \Rightarrow^* v$
 - meaning: $\exists k \geq 0$ and strings u_1, \dots, u_{k-1} for which $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \dots \Rightarrow u_{k-1} \Rightarrow v$
- if u = start symbol, this is a derivation of v
- The language of G , denoted $L(G)$ is:

$$\{w \in \Sigma^* : S \Rightarrow^* w\}$$

January 14, 2008

CS21 Lecture 4

53

CFG example

- Balanced parentheses:
 - $()$
 - $(((((())())))$
- a string w in $\Sigma^* = \{ (,) \}^*$ is balanced iff:
 - #“(“s equals #“)“s, and
 - for any prefix of w , #“(“s \geq #“)“s

Exercise: design a CFG for balanced parentheses.

January 14, 2008

CS21 Lecture 4

54