
1

## 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

$w = xyz$ such that

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



2

## Non-regular languages

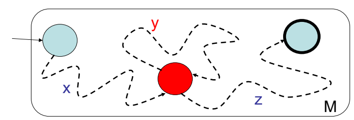
- Using the Pumping Lemma to prove L is not regular:
  - assume L is regular
  - then there exists a pumping length p
  - select a string $w \in L$ of length at least p
  - argue that for every way of writing w = xyz that satisfies (2) and (3) of the Lemma, pumping on y yields a string not in L.
  -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



3

## 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).






4

## 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.



5

## 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.



6

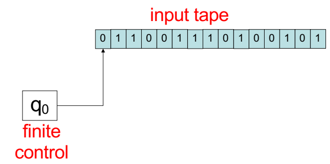
## FA Summary

- 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.



7

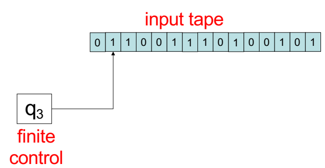
## Machine view of FA






8

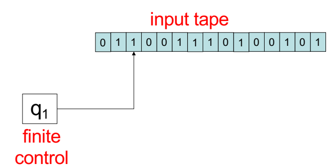
## Machine view of FA






9

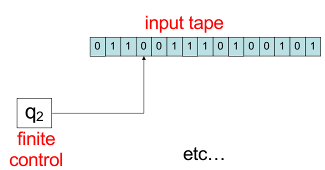
## Machine view of FA






10

## Machine view of FA




etc…



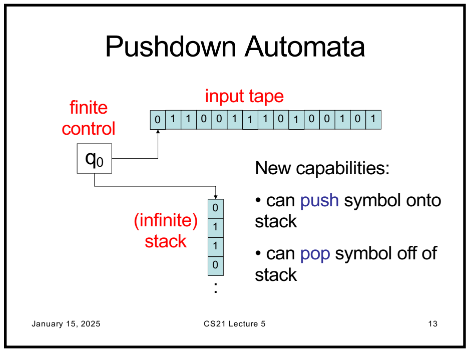
11

## A more powerful machine

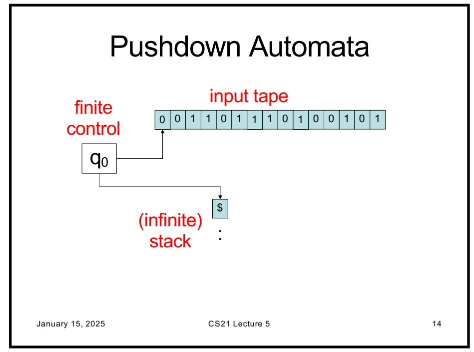
- limitation of FA related to fact that they can only "remember" a bounded amount of information
- What is the simplest alteration that adds unbounded "memory" to our machine?
- Should be able to recognize, e.g., $\{0^n1^n : n \geq 0\}$



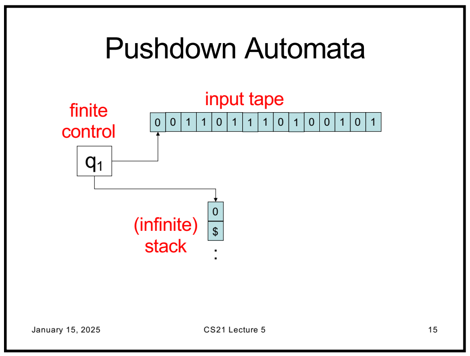
12



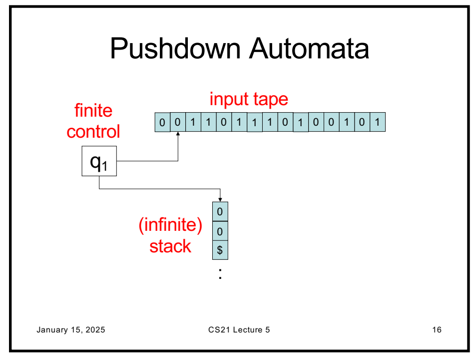

13



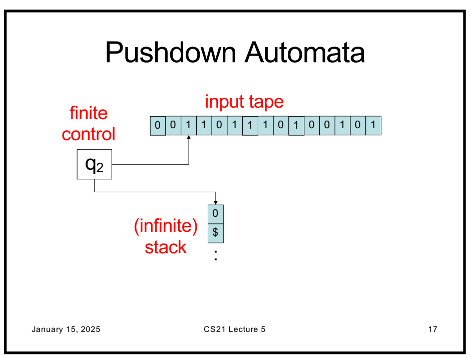

14



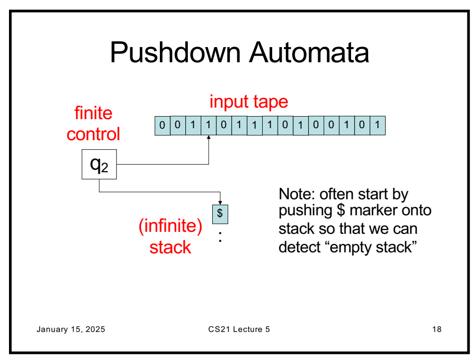

15




16




17




18

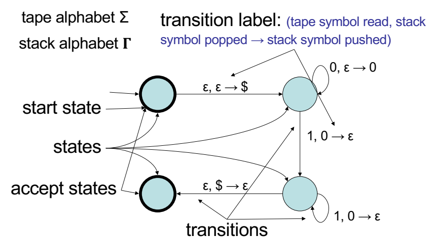
## Pushdown Automata (PDA)

- 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





## NPDA diagram








## NPDA operation

- Taking a transition labeled:

  a, b → c

  - $a \in (\Sigma \cup \{\varepsilon\})$
  - $b,c \in (\Gamma \cup \{\varepsilon\})$

  - read a from tape, or don’t read from tape if $a = \varepsilon$
  - pop b from stack, or don’t pop from stack if $b = \varepsilon$
  - push c onto stack, or don’t push onto stack if $c = \varepsilon$

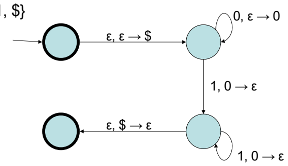




## Example NPDA

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

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




- tape: 0 0 1 1 Stack contents: $

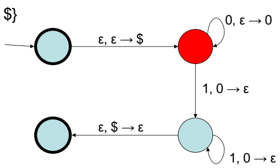




## Example NPDA

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

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




- tape: 0 0 1 1 Stack contents: 0 $

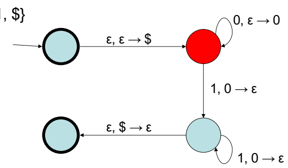




## Example NPDA

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

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

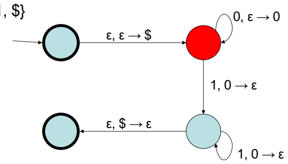



- tape: 0 0 1 1 Stack contents: 0 0 $

## Example NPDA

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

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




- tape: 0 0 1 1 Stack contents: 0 0 $

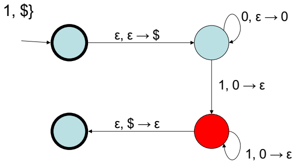


25

## Example NPDA

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

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




- tape: 0 0 1 1 Stack contents: 0 $

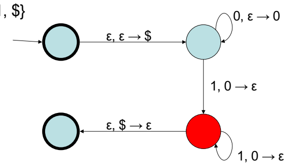


26

## Example NPDA

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

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




- tape: 0 0 1 1 Stack contents: $
  accepted

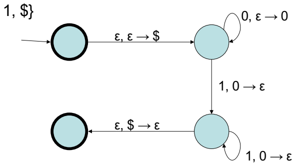


27

## Example NPDA

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

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




- tape: 0 0 1 Stack contents: $

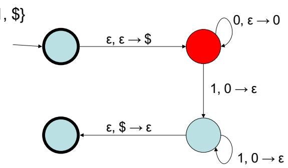


28

## Example NPDA

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

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




- tape: 0 0 1 Stack contents: 0 $

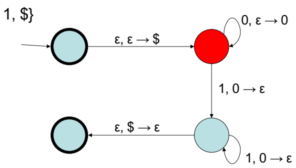


29

## Example NPDA

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

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




- tape: 0 0 1 Stack contents: 0 0 $

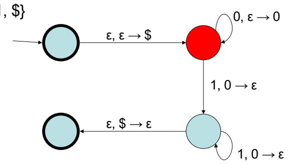


30

## Example NPDA

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

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




- tape: 0 0 1      Stack contents: 0 0 $

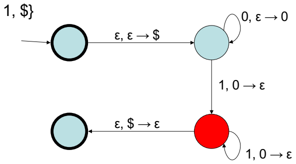


31

## Example NPDA

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

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




- tape: 0 0 1      Stack contents: 0 $

  not accepted

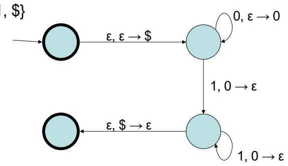


32

## Example NPDA

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

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




- What language does this NPDA accept?



33

## 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
  - $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \to \mathcal{P}(Q \times (\Gamma \cup \{\varepsilon\}))$ 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



34

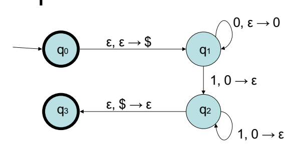
## Formal definition of NPDA

- NPDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ accepts string $w \in \Sigma^*$ if w can be written as
  $w_1 w_2 w_3 \ldots w_m \in (\Sigma \cup \{\varepsilon\})^*$, and
- there exist states $r_0, r_1, r_2, \ldots, r_m$, and
- there exist strings $s_0, s_1, \ldots, s_m$ in $(\Gamma \cup \{\varepsilon\})^*$
  - $r_0 = q_0$ and $s_0 = \varepsilon$
  - $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where $s_i = at$, $s_{i+1} = bt$ for some $t \in \Gamma^*$
  - $r_m \in F$



35

## Example of formal definition




- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{0,1\}$
- $\Gamma = \{0, 1, \$\}$
- $F = \{q_0, q_3\}$

- $\delta(q_0, \varepsilon, \varepsilon) = \{(q_1, \$)\}$
- $\delta(q_1, 0, \varepsilon) = \{(q_1, 0)\}$
- $\delta(q_1, 1, 0) = \{(q_2, \varepsilon)\}$
- $\delta(q_2, 1, 0) = \{(q_2, \varepsilon)\}$
- $\delta(q_2, \varepsilon, \$) = \{(q_3, \varepsilon)\}$

other values of $\delta(\bullet, \bullet, \bullet)$ equal {}



36

## Exercise

Design a NPDA for the language

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





## 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





## Context-Free Grammars

start symbol

terminal symbols

$A \rightarrow 0A1$

$A \rightarrow B$

$B \rightarrow \#$

non-terminal symbols

production





## 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





## 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





## Context-Free Grammars

- Natural languages (e.g. English) structure:

shorthand for multiple rules with same lhs

<sentence> → <noun-phrase><verb-phrase>
<noun-phrase> → <cpx-noun> | <cpx-noun><prep-phrase>
<verb-phrase> → <cpx-verb> | <cpx-verb><prep-phrase>
<prep-phrase> → <prep><cpx-noun>
<cpx-noun> → <article><noun>
<cpx-verb> → <verb>|<verb><noun-phrase>
<article> → a | the
<noun> → dog | cat | flower
<verb> → eats | sees
<prep> → with

Generate a string in the language of this grammar.

## 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





## 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





## CFG formal definition

- A context-free grammar is a 4-tuple

$$(V, \Sigma, R, S)$$

where
- 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)





## CFG formal definition

- u, v, w are strings of non-terminals and terminals, and $A \to w$ 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 $u_1, u_2, \ldots u_{k-1}$ for which $u \Rightarrow^1 u_1 \Rightarrow^1 u_2 \Rightarrow^1 \ldots \Rightarrow^1 u_{k-1} \Rightarrow^1 v$





## CFG formal definition

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

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