CS21
Decidability and Tractability

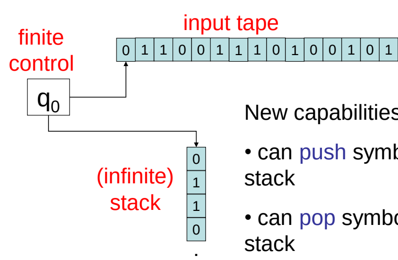
Lecture 5

January 14, 2015

Outline

- Pushdown Automata
- Context-Free Grammars and Languages
  - parse trees
  - ambiguity
  - normal form
- equivalence of NPDAs and CFGs

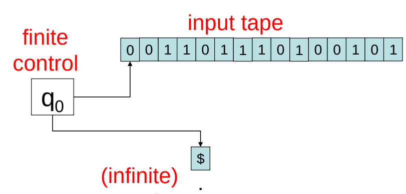
Pushdown Automata



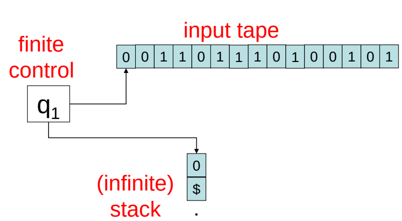
New capabilities:

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

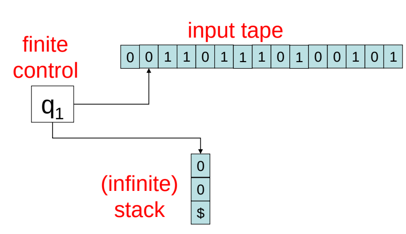
Pushdown Automata



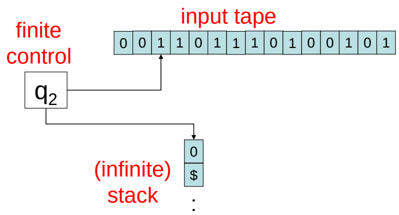
Pushdown Automata



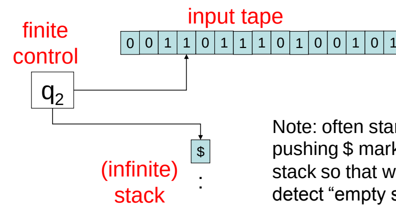
Pushdown Automata



Pushdown Automata



Pushdown Automata



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

Pushdown Automata (PDA)

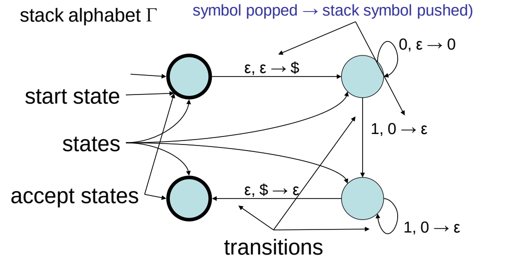
- We will define nondeterministic pushdown automata immediately
  - potentially several choices of "next step"
- Deterministic PDA defined later
  - weaker than NPDA
- Two ways to describe NPDA
  - diagram
  - formal definition

NPDA diagram

tape alphabet Σ

stack alphabet Γ

transition label: (tape symbol read, stack symbol popped $\rightarrow$ stack symbol pushed)



NPDA operation

- Taking a transition labeled:

  a, b $\rightarrow$ c

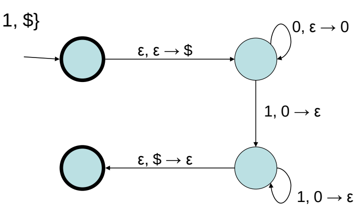
  - a $\in$ ($\Sigma \cup \{\varepsilon\}$)
  - b,c $\in$ ($\Gamma \cup \{\varepsilon\}$)

  - read a from tape, or don't read from tape if a = ε
  - pop b from stack, or don't pop from stack if b = ε
  - push c onto stack, or don't push onto stack if c = ε

Example NPDA

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

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

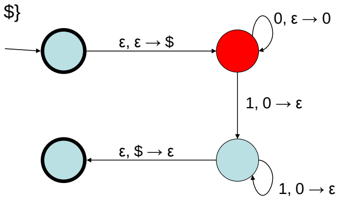


- tape: 0 0 1 1   Stack contents: $

Example NPDA

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

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

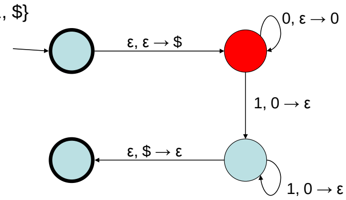


- tape: 0 0 1 1 Stack contents: 0 $

Example NPDA

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

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

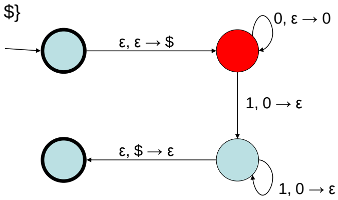


- tape: 0 0 1 1 Stack contents: 0 0 $

Example NPDA

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

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

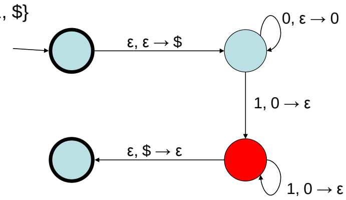


- tape: 0 0 1 1 Stack contents: 0 0 $

Example NPDA

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

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

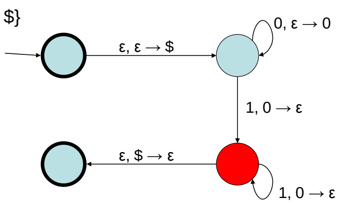


- tape: 0 0 1 1 Stack contents: 0 $

Example NPDA

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

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

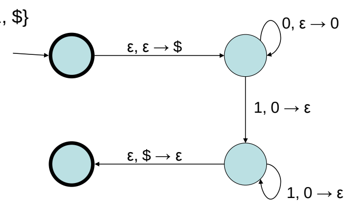


- tape: 0 0 1 1 Stack contents: $
  accepted

Example NPDA

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

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

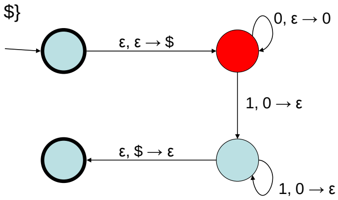


- tape: 0 0 1 Stack contents: $

Example NPDA

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

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

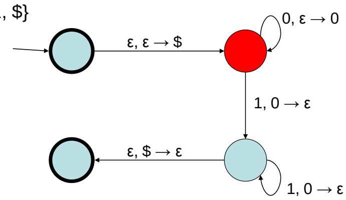


- tape: 0 0 1   Stack contents: 0 $

Example NPDA

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

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

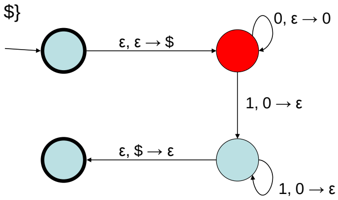


- tape: 0 0 1   Stack contents: 0 0 $

Example NPDA

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

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

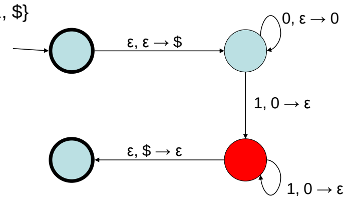


- tape: 0 0 1   Stack contents: 0 0 $

Example NPDA

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

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



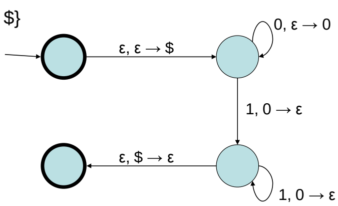
- tape: 0 0 1   Stack contents: 0 $

not accepted

Example NPDA

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

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



- What language does this NPDA accept?

Formal definition of NPDA

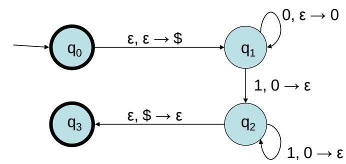
- A NPDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where:
  - Q is a finite set called the states
  - Σ is a finite set called the tape alphabet
  - Γ is a finite set called the stack alphabet
  - $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow \wp(Q \times (\Gamma \cup \{\varepsilon\}))$ 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

Formal definition of NPDA

- NPDA M = (Q, Σ, Γ, δ, q_0, F) accepts string w ∈ Σ* 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 $t \in \Gamma^*$
  - $r_m \in F$

Example of formal definition



- Q = {q_0, q_1, q_2, q_3}
- Σ = {0,1}
- Γ = {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 {}

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 ⇒ 0A1 ⇒ 00A11 ⇒ 000A111 ⇒ 000B111 ⇒ 000#111

A → 0A1
A → B
B → #

- 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, Σ, 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 → 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 $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_{k-1} \Rightarrow v$

CFG formal definition

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

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

CFG example

- Balanced parentheses:
  - ( )
  - (( ) ((( ) ( ))))
- a string w in Σ* = { (,) }* is balanced iff:
  - # “(”s equals # “)”s, and
  - for any prefix of w, # “(”s ≥ # “)”s

Exercise: design a CFG for balanced parentheses.