Theory of Computation

Lecture 8: Pushdown Automata

Outline

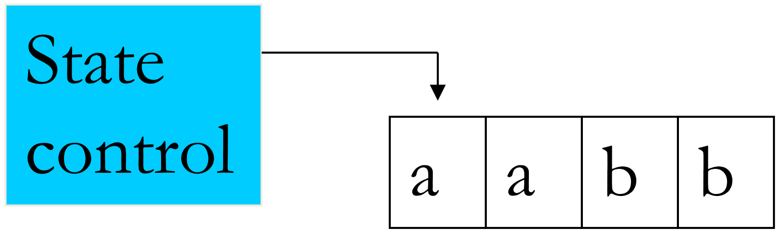
- Pushdown automata
- Formal Definition
- The language of a PDA
- Constructing a PDA
- Equivalence of PDAs and CFGs

From Sipser Chapter 2.2

Pushdown Automata (PDA)

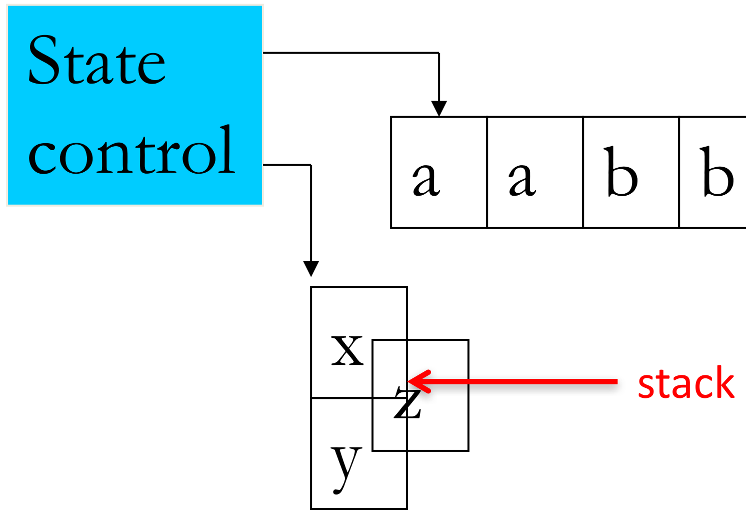
- Similar to NFAs but have an extra component called a *stack*
  - The stack provides extra memory that is separate from the control
- Allows PDA to recognize non-regular languages
- Equivalent in power/expressiveness to a CFG
- Some languages easily described by generators others by recognizers
- Nondeterministic PDA's <u>not</u> equivalent to deterministic ones but NPDA = CFG

Components of a FA



- The state control represents the states and transition function
- Tape contains the input string
- Arrow represents the input head and points to the next symbol to be read from the tape

Components of a PDA



The PDA adds a stack

- Stack acts like a local memory
- Can write to the stack and read them back later
- A stack is a LIFO (Last In First Out) and size is not bounded
- Write to the top (push) and rest “push down” or
- Can remove (read) from the top (pop) and other symbols move up

PDA and Language 0^n1^n

Can a PDA recognize the language 0^n1^n?

- Yes, because size of stack is not bounded
- Describe the PDA that recognizes this language
  - Read symbols from input. Push each 0 onto the stack.
  - As soon as a 1's are seen, starting popping one 0 for each 1
  - If finish reading the input and have no 0's on stack, then accept the input string
  - If stack is empty and 1s remain or if stack becomes empty and still 1's in string, reject
  - If at any time see a 0 after seeing a 1, then reject

Definition of a PDA

- Similar to that of a FA but now we have a stack
  - Stack alphabet may be different from input alphabet
    - Stack alphabet represented by Γ
  - Transition function must include the stack
    - Domain of transition function is $Q \times \Sigma \cup \varepsilon \times \Gamma \cup \varepsilon$
      - The current state, next input symbol and top stack symbol determine the next move
    - Image of transition function is $Q \times \Gamma \cup \varepsilon$
      - The next state and the next symbol on top of the stack

Formal Definition of PDA

A pushdown automata is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are finite sets

1. Q is the set of states
2. Σ is the input alphabet
3. Γ is the stack alphabet
4. $\delta : Q \times \Sigma\varepsilon \times \Gamma\varepsilon \to P(Q \times \Gamma\varepsilon)$ is transition function
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states

– Note that at any step the PDA may enter a new state and possibly write a symbol on top of the stack

  • This definition allows for nondeterminism since δ can return a set

Strings recognized by PDAs

The following 3 conditions must be satisfied for a string to be accepted:

1. M must start in the start state with an empty stack
2. M must move according to the transition function
3. At the end of the input, M must be in an accept state

– To make it easy to test for an empty stack, a $ is initially pushed onto the stack

Notation

We write transitions as a,b $\rightarrow$ c to mean:

- When the machine is reads“a” from the input and “b” is on the top of the stack, “b” is replaced with “c”
- Any of a, b, or c can be ε!
  - If a is ε then can make stack change without reading an input symbol
  - If b is ε then no need to pop a symbol (just push c)
  - If c is ε then no new symbol is written (just pop b)

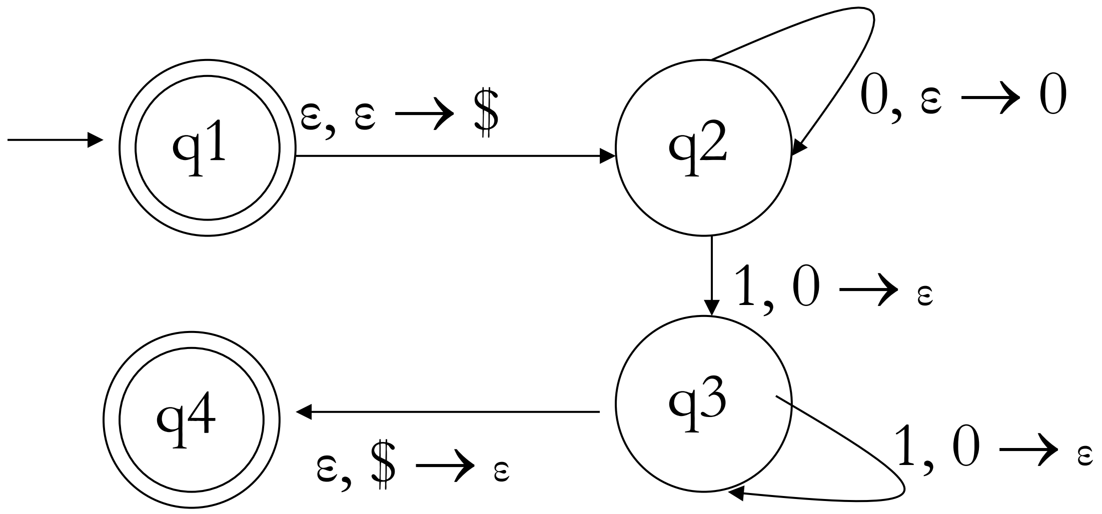
PDA for 0^n1^n

Formally describe PDA that accepts $\{0^n1^n | n \geq 0\}$

Let M1 be $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

$Q = \{q_1, q_2, q_3, q_4\}$ $\Sigma = \{0, 1\}$

$\Gamma = \{0, \$\}$ $F = \{q_1, q_4\}$

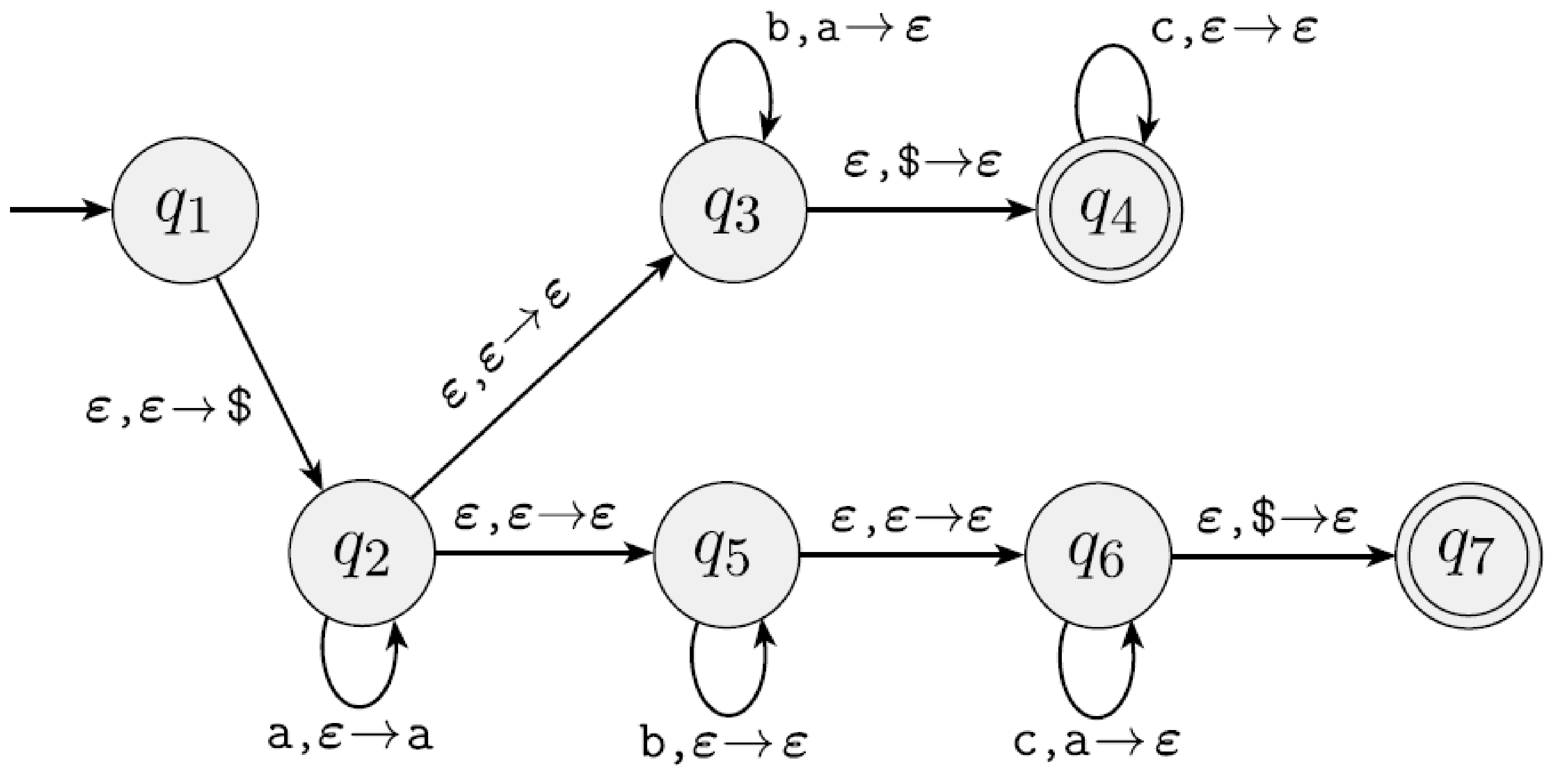


Example : PDA for $a^i b^j c^k$, i=j or i=k

Design a PDA that recognizes the language $\{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i=j \text{ or } i=k\}$

- Think of an informal description
- Can we do it without using non-determinism?
  - No
- With non-determinism?
  - Similar to $0^n 1^n$ except that we need to guess non-deterministically whether to match a's with b's or c's

Example : PDA for $a^ib^jc^k$, i=j or i=k

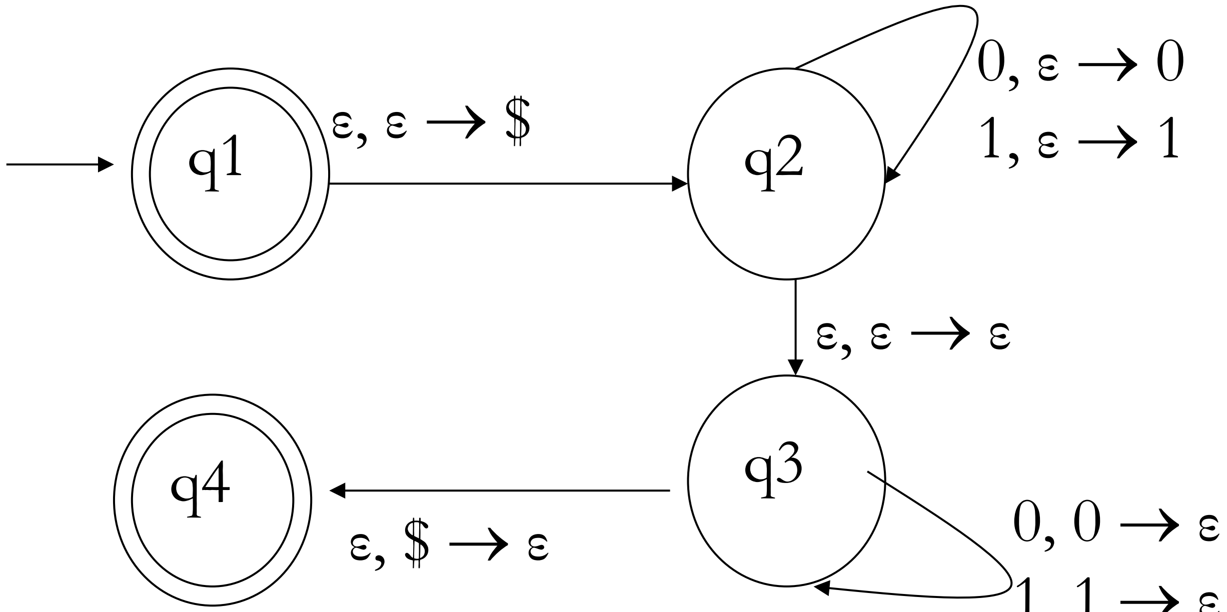


Example 2: PDA for $\{ww^R\}$

Design a PDA for the language $\{ww^R \mid w \in \{0,1\}^*\}$

- w^R is the reverse of w so this is the language of palindromes
- Can you informally describe the PDA?
- Can you come up with a non-deterministic one?
  - Yes, push symbols that are read onto the stack and at some point nondeterministically guess that you are in the middle of the string and then pop off stack value as they match the input (if no match, then reject)

PDA for $\{ww^R\}$



Equivalence of PDAs and CFGs

Theorem: A language is context free if and only if some pushdown automaton recognizes it

- First direction : if a language L is context free then some PDA recognizes it
  Proof idea: We show how to take a CFG that generates L and convert it into an equivalent PDA P
  - Thus P accepts a string only if the CFG can derive it
  - Each main step of the PDA involves an application of one rule in the CFG
  - The stack contains the intermediate strings generated by the CFG
  - Since the CFG may have a choice of rules to apply, the PDA must use its non-determinism
  - One issue: since the PDA can only access the top of the stack, any terminal symbols pushed onto the top of the stack must be checked against the input string immediately.
    - If the terminal symbol matches the next input character, then advance input string
    - If the terminal symbol does not match, then terminate that path

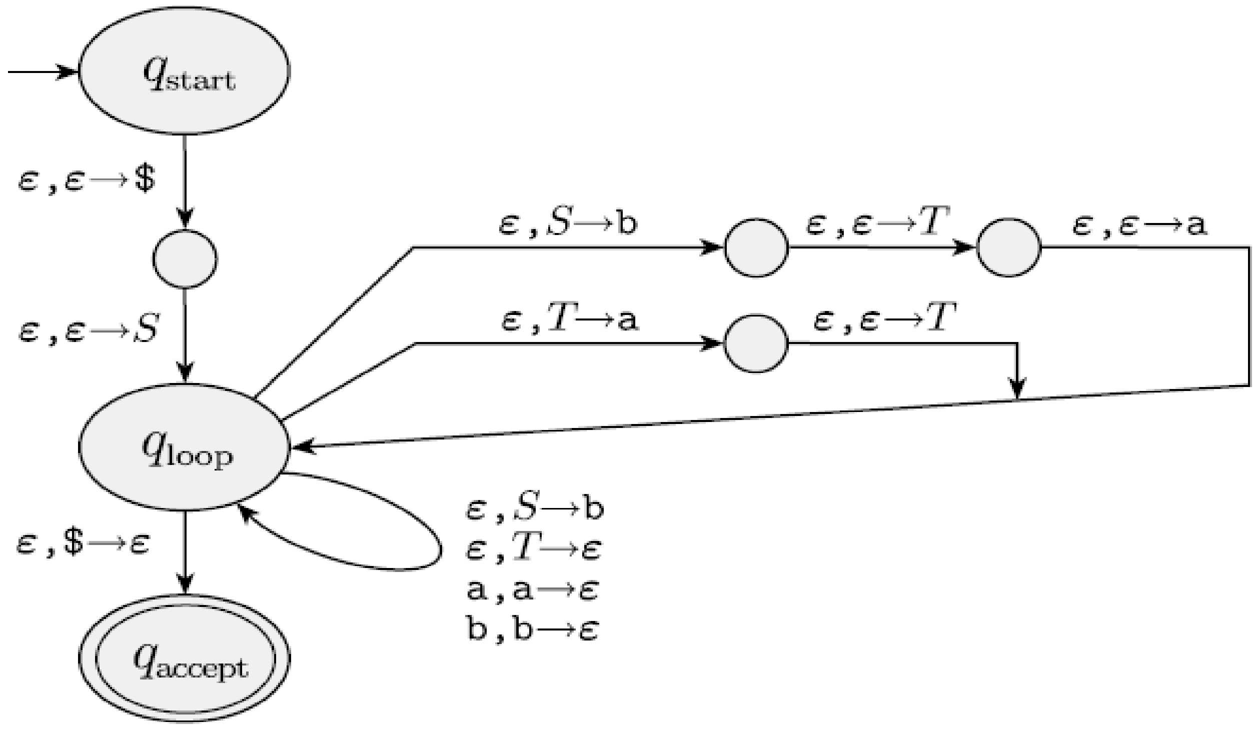
Informal Description of P

- Place marker symbol $ and the start variable on the stack
- Repeat forever
  - If the top of the stack is a variable A, nondeterministically select one of the rules for A and substitute A by the string on the right-hand side of the rule
  - If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a. If they match, repeat. If they do not match, reject this branch.
  - If the top of stack is the symbol $, enter the accept state. Doing so accepts the input if it has all been read.

Example 2.25: construct a PDA P1 from a CFG G

$$\begin{aligned} S &\to \mathtt{a}T\mathtt{b} \mid \mathtt{b} \\ T &\to T\mathtt{a} \mid \varepsilon \end{aligned}$$

The transition function is shown in the following diagram.



q_{start}
$\varepsilon,\varepsilon\to\$$
$\varepsilon,\varepsilon\to S$
$\varepsilon,S\to\mathtt{b}$
$\varepsilon,\varepsilon\to T$
$\varepsilon,\varepsilon\to\mathtt{a}$
$\varepsilon,T\to\mathtt{a}$
$\varepsilon,\varepsilon\to T$
q_{loop}
$\varepsilon,S\to\mathtt{b}$
$\varepsilon,T\to\varepsilon$
$\mathtt{a},\mathtt{a}\to\varepsilon$
$\mathtt{b},\mathtt{b}\to\varepsilon$
$\varepsilon,\$\to\varepsilon$
q_{accept}

Example

- Note that the top path in q_{loop} branches to the right and replaces S with aTb
  - It first pushes b, then T, then a (a is then at top of stack)
- Note the path below that replaces T with Ta
  - It replaces T with a then pops T on top of that
- Your task:
  - Show how this PDA accepts the string aab, which has the following derivation:
    - $S \rightarrow aTb \rightarrow aTab \rightarrow aab$

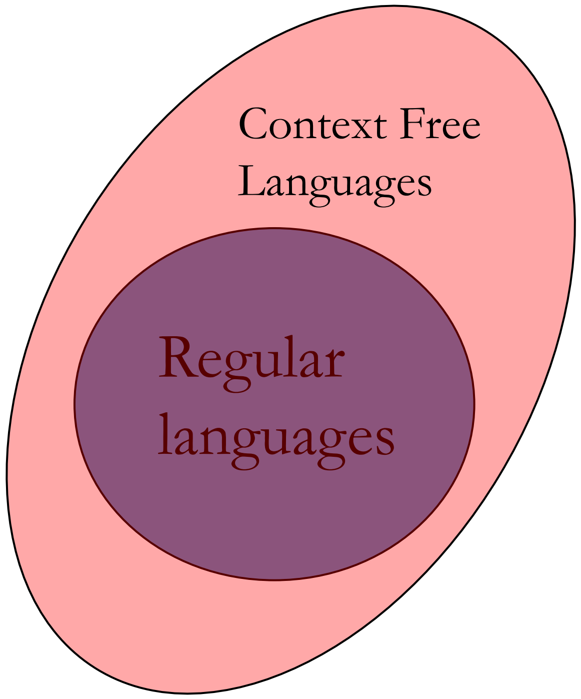
Example continued

- In the following, the left is the top of stack
  - We start with S$
    - We take the top branch to the right and we get the following as we go thru each state:
      - S$ $\rightarrow$ b$ $\rightarrow$ Tb$ $\rightarrow$ aTb$
    - We read a and use rule a,a $\rightarrow$ ε to pop it to get Tb$
    - We next take the 2nd branch going to right:
      - Tb$ $\rightarrow$ ab$ $\rightarrow$ Tab$
    - We next use rule ε,T $\rightarrow$ ε to pop T to get ab$
    - Then we pop a then pop b at which point we have $
    - Everything read so accept

Relationship of Regular Languages & CFLs

- We know that CFGs define CFLs
- We now know that a PDA recognizes the same class of languages and hence recognizes CFLs
- We know that every PDA is a FA that just ignores the stack
- Thus PDAs recognize regular languages
- Thus the class of CFLs contains regular languages
- But since we know that a FA is not as powerful as a PDA (e.g., 0^n1^n) we can say that CFLs and regular languages are not equivalent

Relationship of Regular Languages & CFLs


Context Free Languages
Regular languages