CS 3313
Foundations of Computing: Part II

Pushdown Automata

http://gw-cs3313.github.io

1

Automaton Models

- Deterministic Finite Automata/ Finite State Machines
  - Finite number of states
  - Each state "summarizes" history of events occurred until current time
  - Reads one input at each step
  - Goes to a next state depending on value of input and current state

- DFAs = Regular Languages
- DFAs cannot accept context free languages

2

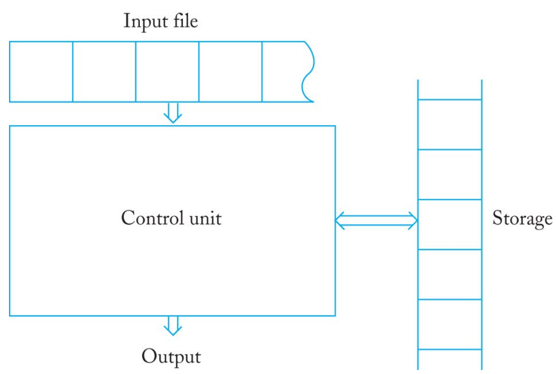
Recall definition: Automata

- An *automaton* is an abstract model of a digital computing device
- An automaton consists of
  - An input mechanism
  - A control unit
  - Possibly, a storage mechanism
  - Possibly, an output mechanism
- Control unit can be in any number of internal *states*, as determined by a *next-state* or *transition* function.
- There are a finite number of states

3

Augmenting the Finite State Machine

- DFAs do not have external memory…
- To increase power of DFAs add external storage
  - Machine in current state can read input, can look up value in memory, and depending on (input + current state + value in memory) goes to next state and can store something in memory.



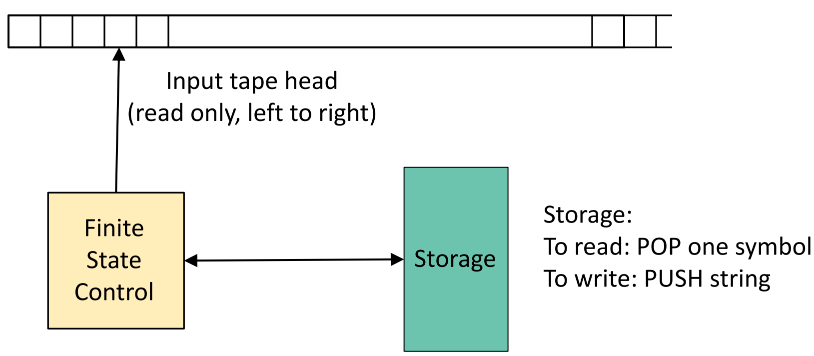
4

Adding a simple memory model to DFAs

- One simple form of memory/storage = a box
  - Simple because we don't need to keep track of "memory address"
  - Throw/Write things into the box – place it on the top of other items in the box
  - Remove/Read the topmost item in the box
- In terms of computational models, box = stack
  - First-in Last-out
- Let's call this machine model M, a "NFA+S" (NFA + Stack)
  - Known formally as a Pushdown Automata (PDA)
- Behavior of machine M:
  1. Reads input, Reads from top of the box/stack, and checks current states
  2. Goes to next state, and store (or not?) something into the box
     - And then reads next input

5

Automata with Stack Storage



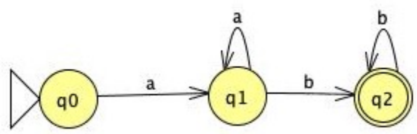
Machine Model: In one step
Current: current state, reads symbol (or empty string) from input, reads top of stack (POPs top of stack)
Next: goes to next state, PUSHes a string (possibly empty) to the stack

Accepts: if machine is in a Final state (or if stack is empty)

6

Recall: Machine design/description

- Each state captures some property of the input processed thus far
- Based on the property and current input we define the next "action"
- Example: a(a*)b(b)*
  - Start in q_0: Not read any input
    - Read an a, go to q_1
    - Read b, go to trap/reject state
  - q_1: have read at least one a.
    - Read a, stay in q_1
    - Read b, go to q2
  - q_2: Have read at least one a, followed by at least one b
    - Read b, stay in q_2
    - Read a, go to trap/reject state



7

Example: $L = \{a^n b^n \mid n > 0\}$

1. $L = \{a^n b^n \mid n > 0\}$
   1. Start q_0: reading a's (bottom of stack marker = Z)
      - Read a with TOS=Z: push AZ to stack stay in Step 1
      - Read a with TOS=A: push AA to stack, stay in Step 1
      - Read b with TOS=A: push nothing (λ) to stack, goto Step 2
   2. q1: Completed reading a's so should only read b's with A on TOS. Match number of A's on stack with number of b's in input
      - Read b with TOS=A: push nothing (λ), stay in Step 2
      - Read λ with TOS=Z: push Z, goto Step 3
   3. q_2: We reach this step if we have equal number of a's and b's and input is empty and stack is empty (with Z on TOS)
      - Accept input.

8

Pushdown Automaton (PDA)

- This machine model (NFA with stack storage) is formally known as a Pushdown Automaton (PDA).
  - The default PDA definition is a non-deterministic machine

    from current configuration, it has number of choices for next move

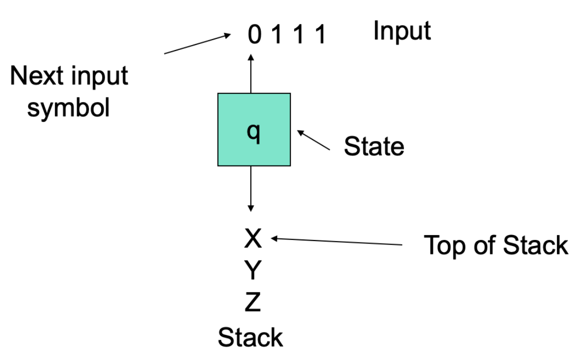
    each choice specifies: next state, push string to stack
- The PDA is an automaton that accepts Context free languages
  - and equivalent to Context Free Grammars in language-defining power.
  - But the deterministic version models parsers.
    - Syntax of most programming languages have deterministic PDA's.

9

Intuition: PDA

- Think of an λ-NFA with the additional power that it can manipulate a stack.
- Its moves are determined by:

1. The current state (of its "NFA"),
2. The current input symbol (or λ), and
3. The current symbol on top of its stack.



10

10

Intuition: PDA – (2)

- Being nondeterministic, the PDA can have a choice of next moves.
- In each choice, the PDA can:
  1. Change state, and also
  2. Replace the top symbol on the stack by a sequence of zero or more symbols.
     - Zero symbols = "pop."
     - Many symbols = sequence of "pushes."

11

11

PDA Formal Definition

- A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is described by:
  1. A finite set of *states* Q (same as before).
  2. An *input alphabet* Σ (same as before).
  3. A *stack alphabet* Γ (typically assume Γ disjoint from Σ).
  4. A *transition function* δ
     - $\delta: (Q \times (\Sigma \cup \lambda) \times \Gamma) \rightarrow 2^{(Q \times \Gamma^*)}$ (subset of $Q \times \Gamma^*$)
       - Number of choices (i.e., non-deterministic)
       - Ex: $\delta(q_1, 0, X) = \{ (q_1, XX), (q_2, \lambda) \}$
  5. A *start state* q_0, in Q (same as before).
  6. A *start symbol* Z_0, in Γ (to indicate bottom of stack).
  7. A set of *final states* $F \subseteq Q$

  Need a few more definitions and notations to define acceptance….

12

12

Pushdown Automaton: Definitions

- There is a specific stack alphabet Γ
  - You could always make it equal to Σ
  - Better to keep it separate but can have a 1-1 mapping
    - Ex: $\Sigma = \{a,b\}$ $\Gamma = \{X,Y\}$ where X corresponds to a and Y to b.
- PDA by default is non-deterministic
  - $\delta(q,a,x)$ has a number of choices of (p,y) where p is a state and y is a stack symbol
  - A deterministic PDA is known as a DPDA (less powerful than PDA)
  - λ-transitions are allowed as the default
- Can also push/pop λ onto stack = push/pop nothing
- Can define a transition graph for a pda
  - each edge is labeled with the input symbol, the stack top, and the string that replaces the top of the stack
  - But cumbersome to model as a graph….so use Parse trees formalism

13

Some notational conventions

- *a, b,* … are input symbols.
  - But sometimes we allow λ as a possible value.
- …, *X, Y, Z* are stack symbols.
- …, *w, x, y, z* are strings of input symbols.
- α, β,… are strings of stack symbols.

14

14

The Transition Function δ

- Takes three arguments:
  1. A state, in Q.
  2. An input, which is either a symbol in Σ or λ
  3. A stack symbol in Γ.
- $\delta(q, a, Z)$ is a set of zero or more actions of the form (p, α).
  - p is a state; α is a string of stack symbols.

15

15

Actions of the PDA

- If $\delta(q, a, Z)$ contains (p, α) among its actions, then one thing the PDA can do in state q, with a at the front of the input, and Z on top of the stack is:
  1. Change the state to p.
  2. Remove a from the front of the input (but a may be λ).
  3. Replace Z on the top of the stack by α.
     - Pop Z and Push α
- Note: (3) above implies that you always pop from TOS therefore to push onto TOS, you have to push the original TOS followed by the new stack symbol

16

16

Example: PDA for { $a^n b^n$ | n >= 1 }

- States:
  - q_0: start state. We are in state q_0 if we have only seen a's so far.
  - q_1: we've seen at least one b and may now proceed only if the inputs are b's
  - q_2: final state – accept
- Stack symbols:
  - Z_0 = start symbol, marks bottom of the stack.
    - If this is top of stack, we know we have counted the same number of *a*'s and *b*'s
  - A = marker used to count the number of a's seen in the input

17

Example 1 – Transition Function

L = {$a^n b^n$ | n >0} $M = (\{q_0,q_1,q_2\}, \{a,b\}, \{A,Z\}, \boldsymbol{\delta}, q_0, Z, \{q_2\})$

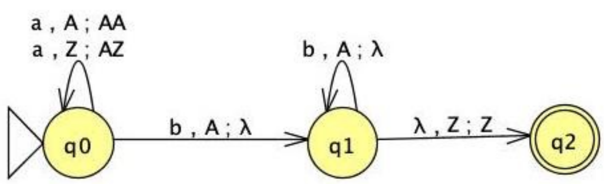
1. Start q_0: (bottom of stack marker = Z)
   - $\delta(q_0,a,Z) = \{(q_0, AZ)\}$ Read a with TOS=Z: push AZ to stack stay in Step 1
   - $\delta(q_0,a,A) = \{(q_0, AA)\}$ Read a with TOS=A: push AA to stack, stay in Step 1
   - $\delta(q_0,b,A) = \{(q_0, \lambda)\}$ Read b with TOS=A: push λ to stack, goto Step 2
2. q1: read only b's with A on TOS. Match #A's on stack with # b's in input
   - $\delta(q_1,b,A) = \{(q_1, \lambda)\}$ Read b with TOS=A: push λ, stay in Step 2
   - $\delta(q_1, \lambda, Z) = \{(q_2, Z)\}$ Read λ with TOS=Z: push Z, goto Step 3
3. q_2: Final State - Accept. We reach this step if we have equal number of a's and b's and input is empty and stack is empty (with Z on TOS)

18

Transition Graph representation for PDAs….

Edge labeled $(a,X,\boldsymbol{\alpha})$ from state p to state q if
$\delta(p,a,X)$ contains $(q, \boldsymbol{\alpha})$

Ex: $\delta(q_0,b,A)$ contains (q_1, λ)
$\delta(q_0,a,A)$ contains (q_0, AA)



19

Deterministic PDA's (DPDA)

- To be deterministic, there must be at most one choice of move for any state q, input symbol a, and stack symbol X.
- In addition, there must not be a choice between using input λ or real input.
  - Formally, $\delta(q, a, X)$ and $\delta(q, \lambda, X)$ cannot both be nonempty.
  - Example for $\{a^n b^n\}$ is a DPDA

20

20

Instantaneous Descriptions

- To trace the actions of a PDA, we must keep track of the current state of the control unit, the stack contents, and the unread part of the input string
  - Note: This was easy to do in a DFA – the extended δ
- We can formalize the concept of a current configuration of the PDA with an ***instantaneous description*** (ID) that describes state, unread input symbols, and stack contents (with the top as the leftmost symbol)
- An ID is a triple (q, w, α), where:
  1. q is the current state.
  2. w is the remaining input.
  3. α is the stack contents, top at the left.

21

Moves in a PDA

- In one "move" (step), a PDA goes from one ID to another
- We say that ID I_1 can become ID I_2 in one move of the PDA, we write $I_1 \vdash I_2$
  - A move is denoted by the symbol $\vdash$ ("yields")
- Formally: $(q, aw, X\alpha) \vdash (p, w, \beta\alpha)$ for any w and α,

  iff $\delta(q, a, X)$ contains (p, β).
- Extend $\vdash$ to $\vdash^*$, meaning "zero or more moves," by:
  - Basis: $I \vdash^* I$.
  - Induction: If $I \vdash^* J$ and $J \vdash K$, then $I \vdash^* K$.

22

Example: Moves

- Using the previous example PDA for $\{a^n b^n\}$, we can describe the sequence of moves by:
  1. $(q_0, aabb, Z_0) \vdash (q_0, abb, AZ_0)$

23

23

Language of a PDA

- The common way to define the language of a PDA is by *final state*.
  - the set of all strings that cause the PDA to halt in a final state, after starting in q_0 with an empty stack.
  - The final contents of the stack are irrelevant
  - As was the case with nondeterministic automata, the string is accepted if any of the computations cause it to halt in a final state

- If M is a PDA, then L(M) is the set of strings w such that

  $(q_0, w, Z_0) \vdash^* \ (f, \lambda\ , \alpha)$ for final state f and any $\alpha \epsilon\ \Gamma^*$

  Important: note that there has to be no input remaining to be processed/read; ex if $(q_0, x, Z_0) \vdash^* \ (f, y\ , \alpha)$ *then y is not accepted by the PDA*

24

24

Language of a PDA – Alternate Definition: Acceptance by Empty stack

- Another way to define acceptance of a language by a PDA is by *empty stack*.
- If M is a PDA, then N(M) is the set of strings w such that

  $(q_0, w, Z_0) \vdash^* (q, \lambda, \lambda)$ for any state q.

Note: stack has to be empty (machine stops) and input remaining is empty.

25

25

Equivalence of PDA Language Definitions

1. If L = L(P), then there is another PDA P' such that L = N(P').
2. If L = N(P), then there is another PDA P'' such that L = L(P'').

Either type of PDA acceptance works!

26

26

Example 1: Transition Function for acceptance on empty stack

L = $\{a^n b^n \mid n > 0\}$ $M = (\{q_0, q_1\}, \{a,b\}, \{A,Z\}, \boldsymbol{\delta}, q_0, Z, \{\})$

1. Start q_0: (bottom of stack marker = Z)
   - $\delta(q_0,a,Z) = \{(q_0, AZ)\}$ Read a with TOS=Z: push AZ to stack stay in Step 1
   - $\delta(q_0,a,A) = \{(q_0, AA)\}$ Read a with TOS=A: push AA to stack, stay in Step 1
   - $\delta(q_0,b,A) = \{(q_0, \lambda)\}$ Read b with TOS=A: push λ to stack, goto Step 2
2. q1: read only b's with A on TOS. Match #A's on stack with # b's in input
   - $\delta(q_1,b,A) = \{(q_1, \lambda)\}$ Read b with TOS=A: push λ, stay in Step 2
   - $\delta(q_1, \lambda, Z) = \{(q_2, \lambda)\}$ Read λ with TOS=Z: push λ stay in Step 2
   - ***String is accepted iff the input remaining is empty and the stack is empty***

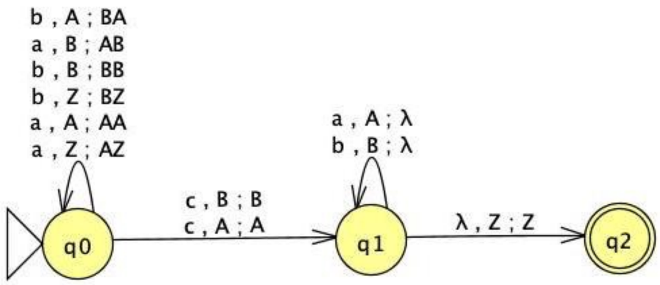
27

Example 2: DesignPDAs { wcw^R | w in {a,b}+ }

- $L = \{ wcw^R \mid w \text{ in } \{a,b\}+ \}$
- Before reading c, we are in the first half (w) and we store it on the stack; after reading c we should check if w^R is being read
  - Start Stack symbol = Z; symbol for a = A; symbol for b = B
- Algorithm Outline:

28

Example 2: PDA for { wcw^R | w in {a,b}+ }



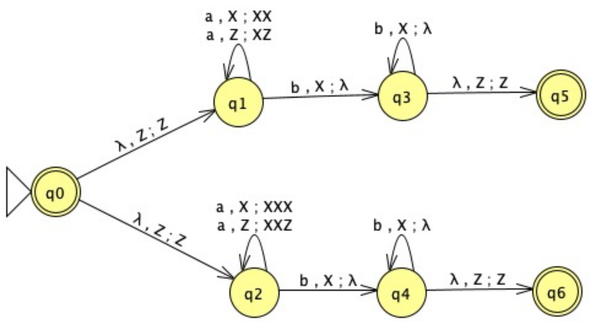
29

Example 3: Design PDA for { a^nb^n | n>0 } ∪ { a^nb^{2n} | n>0 }

- L ={ a^nb^n | n>0 } ∪ { a^nb^{2n} | n>0 }
- Requires non-determinism
- Can you design a PDA M_1 for { a^nb^n | n>0 }
- Can you design a PDA M_2 for { a^nb^{2n} | n>0 }
- Recall "technique" for constructing NFA for union of two machines…..
  - Start machine and then without reading any input we non-deterministically go to M_1 or M_2

30

PDA for $\{ a^n b^n \mid n>0 \} \cup \{ a^n b^{2n} \mid n>0 \}$



31

Exercises: Design/Describe PDAs for languages

- For each of the languages, design/describe PDAs (algorithm) that accept the language

1. $L_2 = \{a^i b^j c^k \mid i=j, \text{ and } i,j,k>0\} \cup \{a^i b^j c^k \mid j=k, \text{ and } i,j,k>0 \}$
2. $L_3 = \{ a^n b^m \mid n <= m <= 2n \text{ and } n > 0 \}$

32