CS 3313
Foundations of Computing:

Pushdown Automata

http://gw-cs3313-2021.github.io

Automaton Models

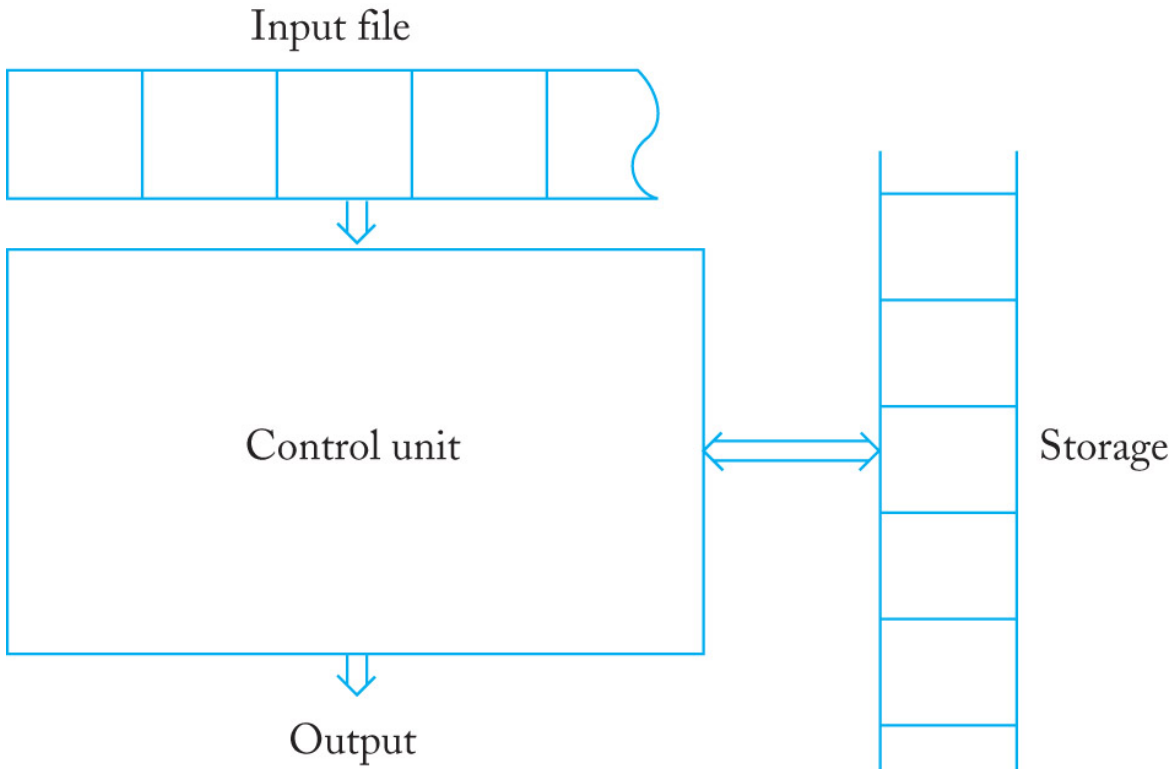
- Deterministic Finite Automata/ Finite State Machines
  - Finite number of states
  - Each state “summarizes” history of events occurred until current time
  - Reads one input at each step
  - Goes to a next state depending on value of input and current state

  - Simplification rules: transform a grammar such that:
  - Resulting grammar generates the same language
  - and has “more efficient” production rules in a specific format

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

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.

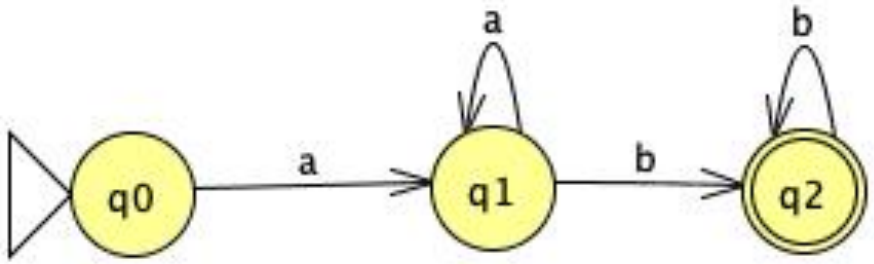


Adding a simple memory model to DFAs

- 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 "DFA+S" (DFA + Stack)
- Behavior of machine M:
  1. Reads input, Reads (or not?) from top of the box, and checks current states
  2. Goes to next state, and store (or not?) something into the box
     - And then reads next input

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 stat
  - 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



Exercises:

- For each of the languages, is it possible to design a DFA+S machine (i.e., a DFA with a box/stack as storage) that accepts the language ?

1. $L1 = \{a^n b^n \mid n > 0\}$
2. $L2 = \{ wcw^R \mid w \text{ in } \{a,b\}+ \}$
3. $L3 = \{ a^n b^n c^n \mid n > 0 \}$

Some Observations from Examples

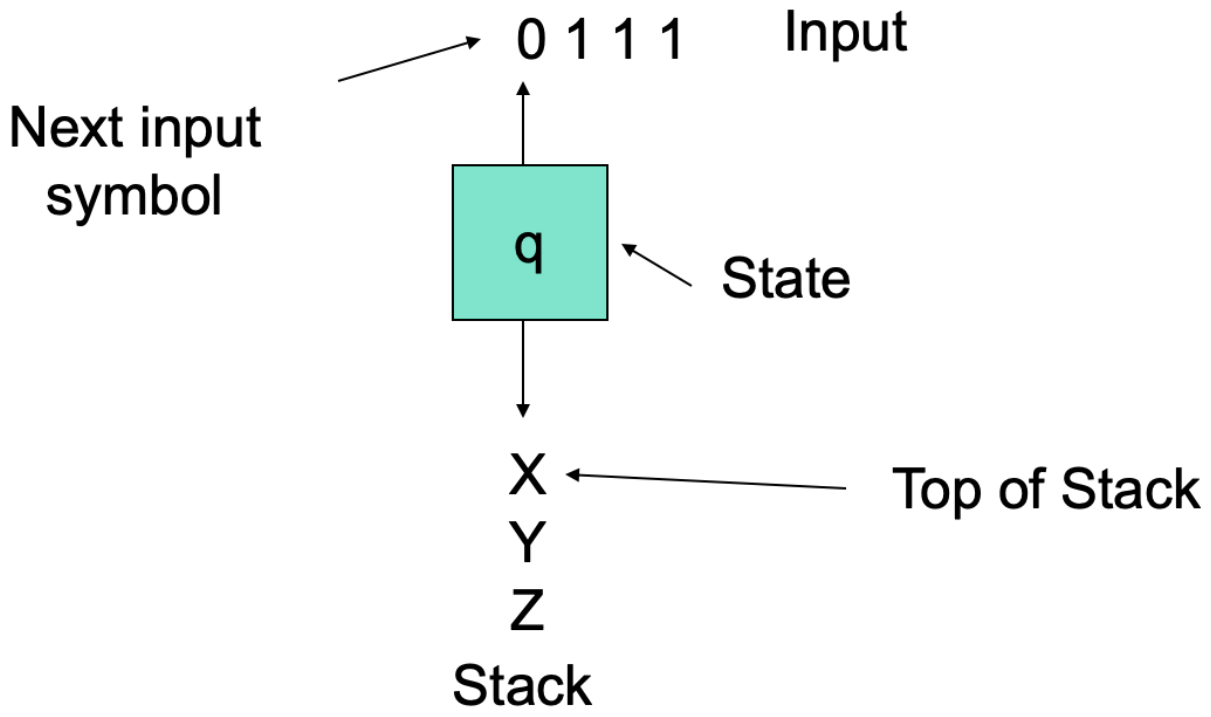
- $L1 = \{a^n b^n \mid n > 0\}$: We can keep track of value of one variable in the stack and compare with another.
- $L2 = \{ wcw^R \mid w \text{ in } \{a,b\}+ \}$: If we push a "string" into the stack, when we "pop" the string the reverse of the string is what can be read by machine M
- $L3 = \{ a^n b^n c^n \mid n > 0 \}$: ??? Looks like having a stack storage is not enough to recognize/accept this language
  - From observation in L1, we can have "one counter" but not two.

Formal Definition: Pushdown Automaton (PDA)

- This machine model (DFA with stack storage) is formally known as a Pushdown Automaton (PDA).
- The PDA is an automaton equivalent to the CFG in language-defining power.
- Only the nondeterministic PDA defines all the CFL's.
- But the deterministic version models parsers.
  - Most programming languages have deterministic PDA's.

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 ϵ), and
  3. The current symbol on top of its stack.



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."

PDA Formalism

- 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 $Q \times \Gamma^*$)
  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* $F \subseteq Q$

Pushdown Automata

- A (*nondeterministic) pushdown automata (*pda) is defined by M= (Q, Σ, Γ,δ, q_0, z, F):
  - A finite set of states Q (same as before)
  - An input alphabet Σ (same as before)
  - A stack alphabet Γ
  - A transition function δ: $(Q \times (\Sigma \cup \lambda) \times (\Gamma \cup \lambda)) \rightarrow 2^{(Q \times \Gamma^*)}$ (subset of $Q \times \Gamma^*$)
  - An initial state q_0 (same as before)
  - A stack start symbol z – *similar to an empty stack marker*
  - A set of final states F (same as before)
- Input to the transition function δ consists of a triple consisting of a state, input symbol (or λ), and the symbol at the top of stack
- Output of δ consists of a new state and new top of stack
- Transitions can be used to model common stack operations

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: Σ ={a,b} Γ= {X,Y} where X corresponds to a and Y to b.
- PDA by default is non-deterministic
  - δ(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

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.

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.
- δ(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.

Actions of the PDA

- If δ(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

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
  - X = marker used to count the number of a's seen in the input

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

- The transitions:
  - $\delta(q_0, a, Z_0) = \{(q_0, XZ_0)\}$.
  - $\delta(q_0, a, X) = \{(q_0, XX)\}$. These two rules cause one X to be pushed onto the stack for each a read from the input.
    - Observe how we push XX after reading/popping X
  - $\delta(q_0, b, X) = \{(q_1, \lambda)\}$. When we see a b, go to state q_1 and pop one X.
  - $\delta(q_1, b, X) = \{(p, \lambda)\}$. Pop one X per b.
  - $\delta(q_1, \lambda, Z_0) = \{(q_2, Z_0)\}$. Accept at bottom.

Deterministic PDA's

- 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, δ(q, a, X) and δ(q, λ, X) cannot both be nonempty.

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.

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,

  if $\delta(q, a, X)$ 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 $I \vdash^* K$.

Example: Moves

- Using the previous example PDA for $\{a^n b^n\}$, we can describe the sequence of moves by:

  1. $(q_0, aabb, Z_0) \vdash (q_0, abb, XZ_0)$
  2. $(q_0, abb, XZ_0) \vdash (q_0, bb, XXZ_0)$
  3. $(q_0, bb, XXZ_0) \vdash (q_1, b, XZ_0)$
  4. $(q_1, b, XZ_0) \vdash (q_1, \lambda, Z_0)$
  5. $(q_1, \lambda, Z_0) \vdash (q_2, \lambda , Z_0)$

- Thus, $(q_0, aabb, Z_0) \vdash^* (q_2, \lambda , Z_0)$.

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

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

Language of a PDA – Alternate Definition

- 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.

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!

Proof: L(P) -> N(P') Intuition

- P' will simulate P.
- If P accepts, P' will empty its stack.
- P' has to avoid accidentally emptying its stack, so it uses a special bottom-marker to catch the case where P empties its stack without accepting.

Proof: L(P) -> N(P')

- P' has all the states, symbols, and moves of P, plus:
  1. Stack symbol X_0 (the start symbol of *P'*), used to guard the stack bottom.
  2. New start state *s* and "erase" state *e*.
  3. $\delta(s, \lambda, X_0) = \{(q_0, Z_0X_0)\}$. Get *P* started.
  4. Add $\{(e, \lambda)\}$ to $\delta(f, \lambda, X)$ for any final state f of P and any stack symbol X, including X_0.
  5. $\delta(e, \lambda, X) = \{(e, \lambda)\}$ for any X.

Proof: N(P) -> L(P'') Intuition

- P'' simulates P.
- P'' has a special bottom-marker to catch the situation where P empties its stack.
- If so, P'' accepts.

Proof: N(P) -> L(P'')

- P'' has all the states, symbols, and moves of P, plus:
  1. Stack symbol X_0 (the start symbol), used to guard the stack bottom.
  2. New start state s and final state f.
  3. $\delta(s, \lambda, X_0) = \{(q_0, Z_0X_0)\}$. Get P started.
  4. $\delta(q, \lambda, X_0) = \{(f, \lambda)\}$ for any state q of P.

Next: Equivalence of PDAs and CFLs

- Do PDAs accept Context free languages ?

- Prove: Given any CFG, there is a PDA that accepts the language generated by the grammar.
- Prove: Given any PDA, there is a CFG that generates the language accepted by the PDA.