(Nondeterministic) Pushdown Automata

Before

10/10

finite automata $\longleftrightarrow$ regular languages (lang. gen by regular grammars)

pushdown automata $\longleftrightarrow$ context-free languages (lang. gen by context-free grammars)

Pushdown automaton -

"memory" consists of

1) what state you are in (like in finite automaton)

2) ~~the top~~ a stack (you only have access to the top element at a given time)

At each step we read something from string, look at and pop the top ~~elem~~ letter from stack,

move to a ~~(diffe~~ state (nondet)

push some things on the stack.

$L = \{ a^n b^n \mid n \in \mathbb{Z}_{>0} \}$

How do we build a machine for this? Every time we read an a, we push a symbol on the stack - when we read a b, we pop. If at the end, we have an empty stack, good.

S start state

~~(S, a, ?)~~ At S, read a, (doesn't matter what's on stack)

→ go to state S, push an a on the stack

At S, read b, with a on stack

→ go to state B, pop the a off the stack

At B, read b, with a on stack

→ go to state B, pop a off stack

At B, read λ, with an empty stack

→ go to state F

F is final state.

(for PDA in general)

Convention: we accept at the end only when the whole string is read, but we allow stuff left on the stack.

Formal def'n:

A PDA is a tuple $M = (Q, \Sigma, \Gamma, q_0, F, z, \delta)$

Q - finite set of states

Σ - alphabet (finite)

Γ - stack alphabet (finite) - symbols we can put on stack

$q_0 \in Q$ - initial state

$F \subseteq Q$ - (set of) final states

$z \in \Gamma$ - bottom of stack symbol

$\delta \subseteq [Q \times (\Sigma \cup \{\lambda\}) \times \Gamma] \times [Q \times \Gamma^*]$

is the transition relation.

Convention: we'll always pop the top symbol off the stack (we can push it back on)

The transitions for the previous machine:

$(\Gamma = \{a, z\}$

$(S, a, z) \longrightarrow (S, az)$ — right goes on bottom of stack by convention

$(S, a, a) \longrightarrow (S, aa)$

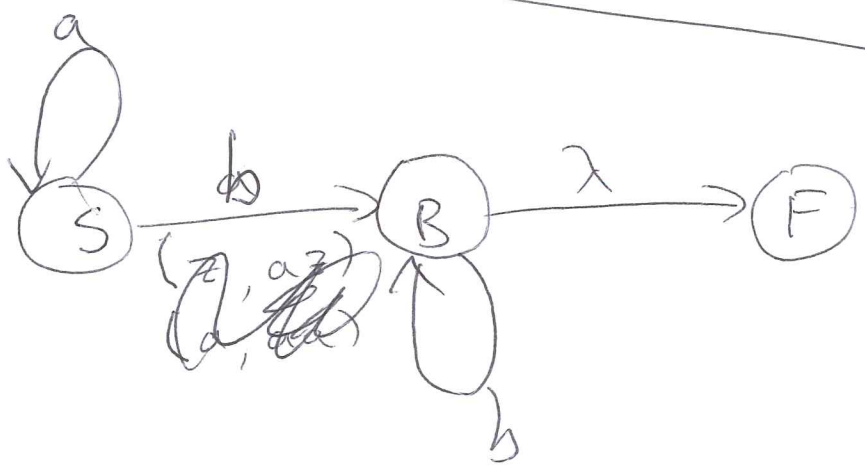
$(S, b, a) \longrightarrow (B, \lambda)$

$(B, b, a) \longrightarrow (B, \lambda)$

$(B, \lambda, z) \longrightarrow (F, z)$

convention: we always remove the top object from stack, so we need to put it back on (along w/ extra a)

F is the final state.



We could draw a picture like this, but we couldn't use it w/o separately tracking the stack, so the picture is pretty useless.

Parentheses matching: $\Gamma = \{$ 🙂, z $\}$

(~~equal~~ ~~more~~) can be more) than (at the end)

$(S, (, z) \rightarrow (S,$ 🙂 $z)$

~~$(S,),)$~~

$(S, (,$ 🙂 $) \rightarrow (S,$ 🙂 🙂 $)$

$(S,),$ 🙂 $) \rightarrow (S, \lambda)$

make S our final state.

---

variant - require equal # of ('s and)'s

S should no longer be final

new state F that we can only get to if stack is "empty"

add transition:

$(S, \lambda, z) \rightarrow (F, z)$

make F the only final state.

variant - require strictly more ('s than)'s

then we should only allow going to F on a  at top of stack; i.e.

$$(S, \lambda, \text{☺}) \longrightarrow (F, \lambda)$$

is the only transition to F.