# Context-Free Languages

PDAs and CFGs

# Equivalence of CFG and PDA

1. Given a CFG, G, construct a PDA, M, such that L(M) = L(G)
2. Given a PDA, M, define a CFG, G, such that L(G) = L(M)

We will show #1 by construction … #2 is a little messy but in the book.

# Step 1: CFG → PDA

- Given:  A context-free grammar G
- Construct: A pushdown automata M
- Such that:
  - The language recognized by M is the same as the language generated by G

# Step 1: CFG → PDA

- Basic idea
  - Use the stack of the PDA to simulate the derivation of a string in the grammar.
    - Push S (start variable of G) on the stack
    - From this point on, there are two moves the PDA can make:

1. If a variable A is on the top of the stack, pop it and push the right-hand side of a production $A \rightarrow \beta$ from G.
2. If a terminal a is on the top of the stack, pop it and match it with whatever symbol is being read from the tape.

# Step 1: CFG → PDA

- Observations:
  - There can be many productions that have a given variable on the left hand side:
    - $S \rightarrow \epsilon \mid 0S1 \mid 1S0$
  - In these cases, the PDA can "choose" which string to push onto the stack after popping a variable.
    - i.e. the PDA being constructed is non-deterministic.
    - Every path through the PDA will correspond to a valid application of the grammar's rules

# Step 1: CFG → PDA

- More observations:
  - A string will only be accepted if:
    - After the string is completely read, the stack is empty

# Step 1: CFG → PDA

- One small problem
  - The PDA definition in the book only allows 1 character to be pushed on the stack during a move of the machine
  - The right hand side of productions of a CFG can be of any finite length.
  - However, we can fix that…

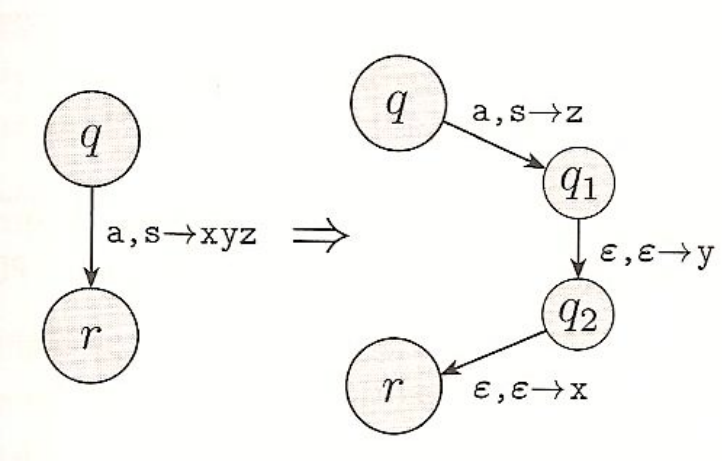
# Step 1: CFG → PDA

- transition function δ – SHORTHAND:
  - During a move of a PDA:
    - At most **one** character is read from the input tape
      - ϵ transitions are okay
    - At most **one** character is popped from the top of the stack
      - ϵ transitions are okay
    - The machine will move to a new state based on:
      - The character read from the tape
      - The character popped off the stack
      - The current state of the machine
    - ***Zero or more characters*** from the stack alphabet are pushed onto the stack.

# Step 1: CFG → PDA

- New shorthand
  - In long form, we can achieve this by introducing new states, one new state for each character in the string to be pushed onto the stack in a single move.

Note that in the shorthand notation, symbols will be pushed onto the stack from right to left. The leftmost symbol will be at the top of the stack when finished.



# Step 1: CFG $\rightarrow$ PDA

- Let's formalize this:
  - Let $G = (V, \Sigma, R, S)$ be a context-free grammar.
  - We define a pushdown automata
    - $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$
  - Such that
    - $L(M) = L(G)$

# Step 1: CFG $\rightarrow$ PDA

- Define M as follows:
  - $Q = \{ q_{start}, q_{loop}, q_{accept} \} \cup E$
    - $q_{start}$ will be the start state
    - $q_{loop}$ will be where all the work gets done
    - $q_{accept}$ will be the accepting state
    - $E$ = set of states required to support shorthand for allowing $\delta$ to place strings on the stack.
  - $\Gamma = V \cup \Sigma \cup \{ \$ \}$ $\quad \$ \notin (V \cup \Sigma)$
  - $F = \{ q_{accept} \}$

# Step 1: CFG $\rightarrow$ PDA

- Transition function $\delta$ (using the string based shorthand) is defined as follows:
  - $\delta\ (q_{start}, \epsilon, \epsilon\ ) = \{\ (q_{loop}, S\$)\ \}$
    - To start things off, push the empty stack marker and then S onto the stack and immediately go into the work state
  - $\delta\ (q_{loop}, \epsilon, A) = \{\ (q_{loop}, \alpha\ )\ |\ A \rightarrow \alpha$ is a production of G$\}$ for all variables A
    - Pop and replace variable.

# Step 1: CFG $\rightarrow$ PDA

- Transition function $\delta$ is defined as follows:
  - $\delta\ (q_{loop}, a, a) = \{\ (q_{loop}, \epsilon\ )\ \}$ for all terminals a
    - Pop and match terminal.
  - $\delta\ (q_{loop}, \epsilon, \$) = \{\ (q_{accept}, \epsilon)\ \}$
    - After all reading is done, accept only if stack is empty.
  - No other transitions exist for M

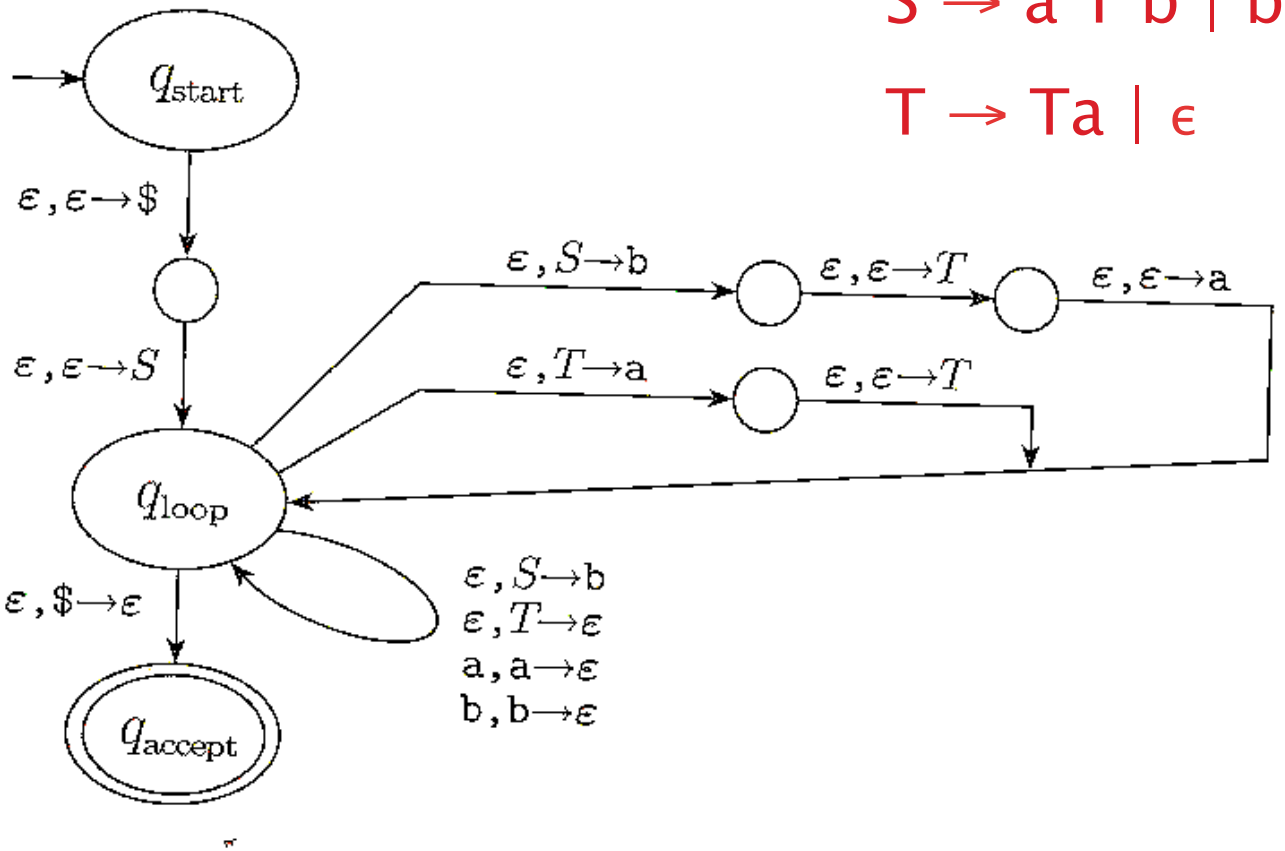
# Step 1: CFG → PDA

- Let's look at an example:
  - Example 2.25 ( in text )
    - $S \rightarrow a\ T\ b \mid b$
    - $T \rightarrow Ta \mid \epsilon$
  - Let's convert this to a PDA.

# Step 1: CFG->PDA

S → a T b | b

T → Ta | ϵ



# Summary

- What have we shown?
  - Given a CFG, we can build a PDA that accepts the same language generated by the CFG
    - This shows that the class of languages represented by PDAs is at least as large as the class of languages represented by CFGs.
  - We can show the other direction (in the book; we won't show in class):
    - Given a PDA, we can define a CFG that can generate the language accepted by the PDA.
- Put together, these show that CFGs and PDAs are equivalent