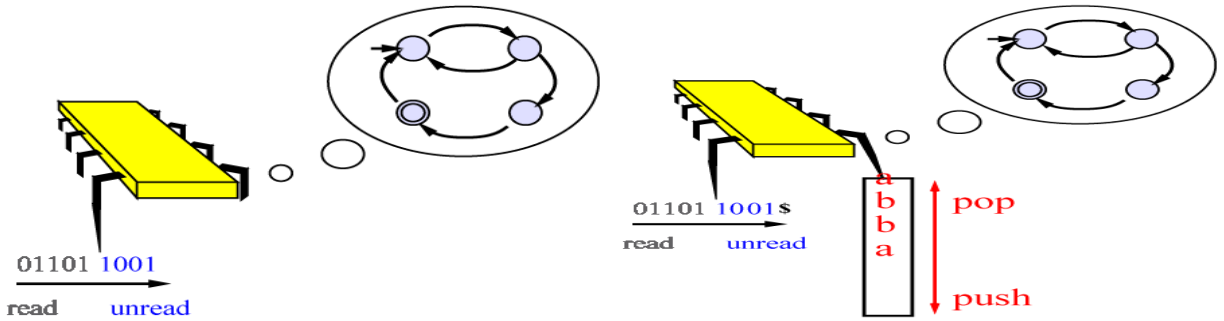
Computational Models - Lecture 5

- Push Down Automata (PDA)
- Equivalence of CFGs and PDAs
- Closure properties for CFL



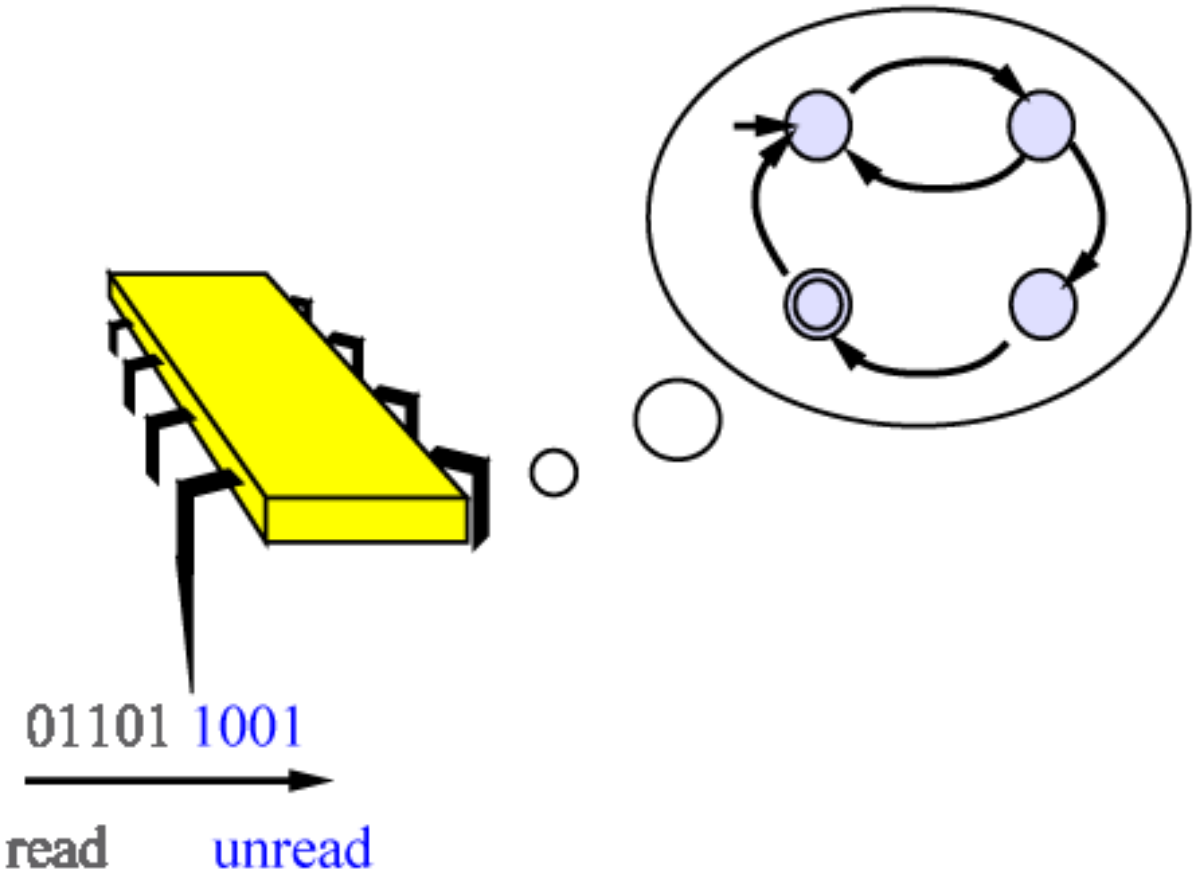
- Sipser's book, 2.1, 2.2 & 2.3
- Next week, exam practicing

Push-Down Automata

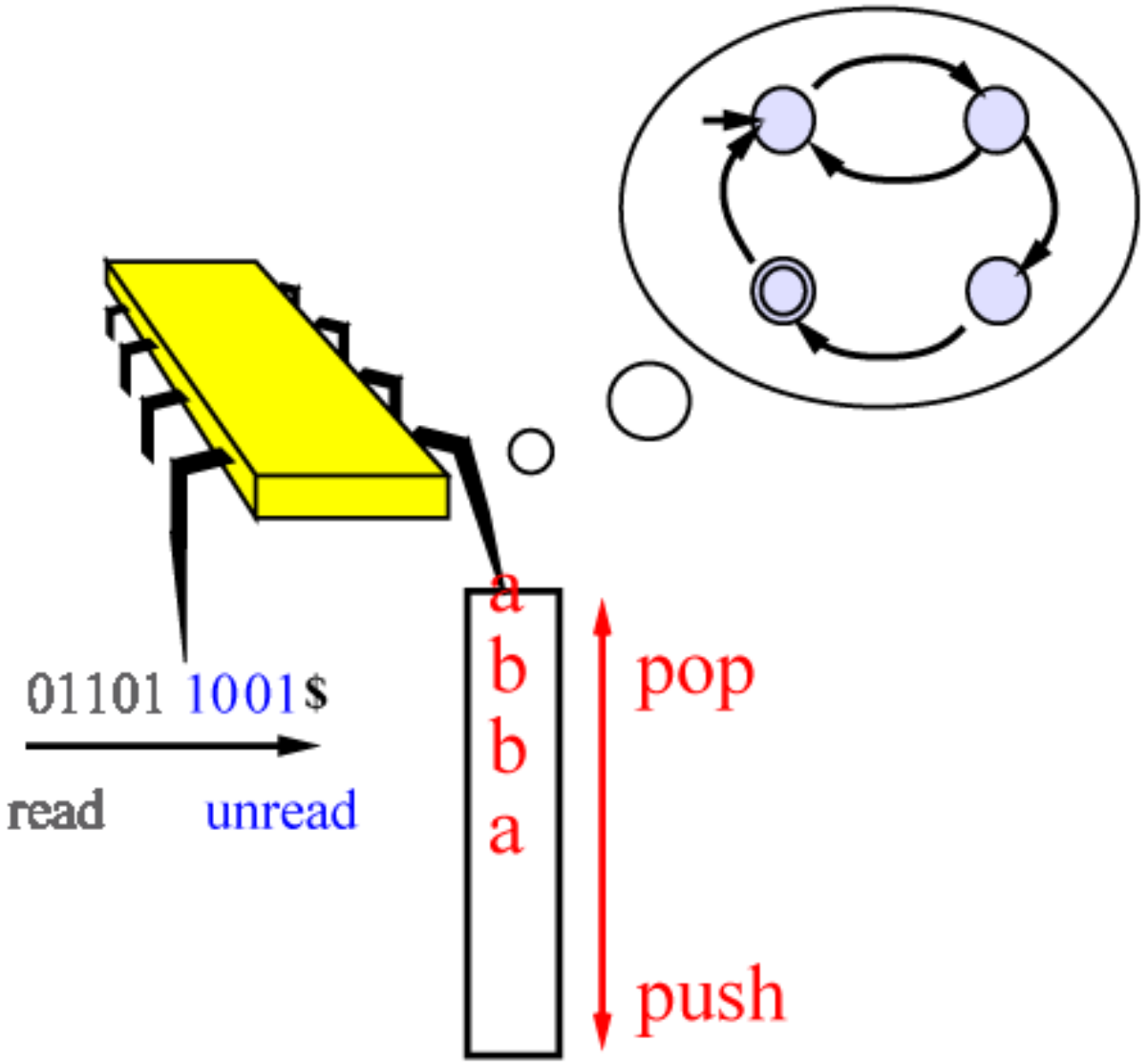
String Generators and String Acceptors

- Regular expressions are string generators – they tell us how to generate all strings in a language L
- Finite Automata (DFA, NFA) are string acceptors – they tell us if a specific string w is in L
- CFGs are string generators
- Are there string acceptors for Context-Free languages?
- YES! Push-down automata

A Finite Automaton



A PushDown Automaton



Example 1

Recall that the language $L_1 = \{0^n 1^n \mid n \geq 0\}$ is not regular. Consider the following PDA:

- Read input symbols
  - Push each read 0 on the stack
  - Pop a 0 for each read 1
  - Accept if stack is empty after last symbol read, and no 0 appears after 1

Example 2

A PDA that accepts

$$L_2 = \left\{ a^i b^j c^k | i = j \vee i = k \right\}$$

Informally:

- read and push a's
- either pop and match with b's
- or else pop and match with c's
- a non-deterministic choice!

Formal Definitions

A pushdown automaton (PDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

- Q is a finite set called the *states*,
- Σ is a finite set called the *input alphabet*,
- Γ is a finite set called the *stack alphabet*,
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the *transition function*,
- $q_0 \in Q$ is the *start state*, and
- $F \subseteq Q$ is the set of *accept states*.

PDA Transition Function

Denote input alphabet by Σ and stack alphabet by Γ.

- the domain of the transition function δ is
  - current state: Q
  - next input, if any: Σ_ε (=$\Sigma \cup \{\varepsilon\}$)
  - stack symbol popped, if any: Γ_ε(=$\Gamma \cup \{\varepsilon\}$)
- and its range is
  - new state: Q
  - stack symbol pushed, if any: Γ_ε
  - non-determinism: $\mathcal{P}$(two components above)
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$

Model of Computation – Informal

- A pushdown automaton (PDA) M accepts a string w if there is a computation of M on w (a sequence of state and stack transitions according to M's transition function and corresponding to w) that leads to an accepting state.
- The language accepted by M, denoted $L(M)$, is the set of all strings $w \in \Sigma^*$ accepted by M.

- A (non-deterministic) PDA may have many computations on a single string

Model of Computation – Formal

M accepts w, if w can be written as $w_1 \dots w_m$ where each $w_i \in \Sigma_\varepsilon$, and exist states $r_0, r_1, \dots, r_m \in Q$ and strings $s_0, s_1, \dots s_m \in \Gamma^*$ such that the following hold:

- $r_0 = q_0$ and $s_0 = \varepsilon$
- For every $i \in \{0, \dots, m-1\}$ it holds that: $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where $s_i = at$ and $s_{i+1} = bt$, for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$
- $r_m \in F$

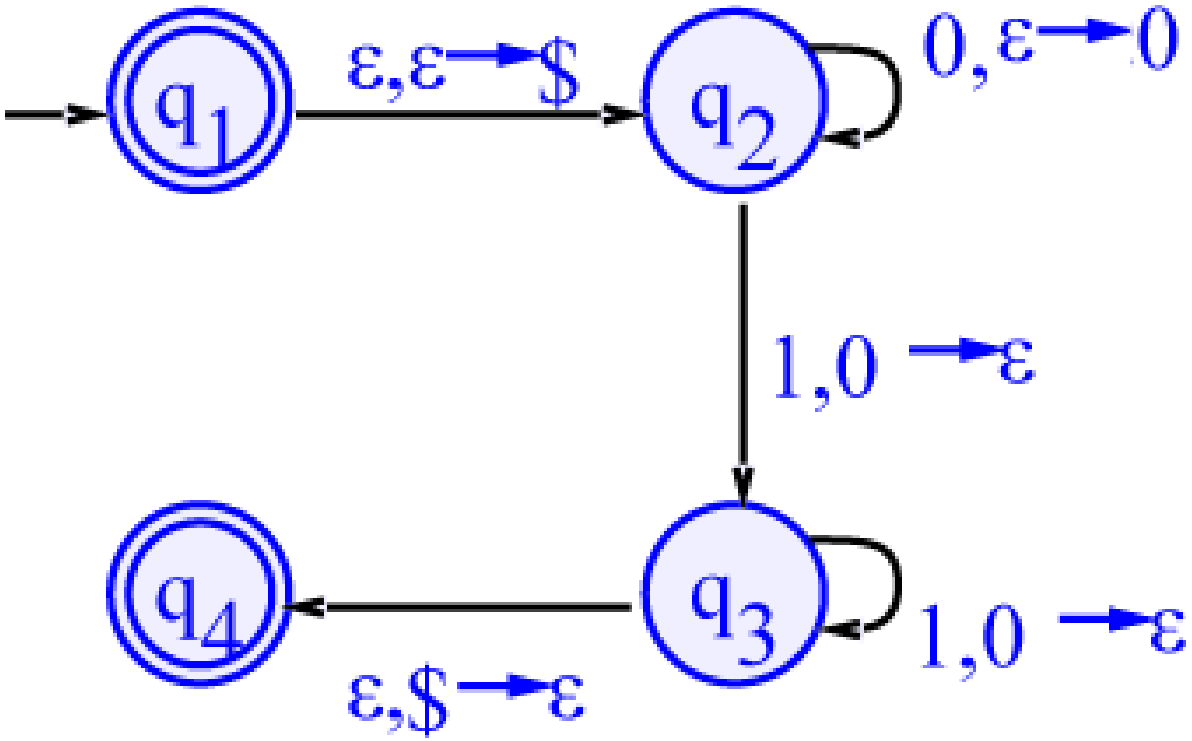
Conventions

- It will be convenient to be able to know when the stack is empty, but there is no built-in mechanism to do that.
  - Solution:
  - Start by pushing $ onto stack.
  - When you see it again, stack is empty.

Notation

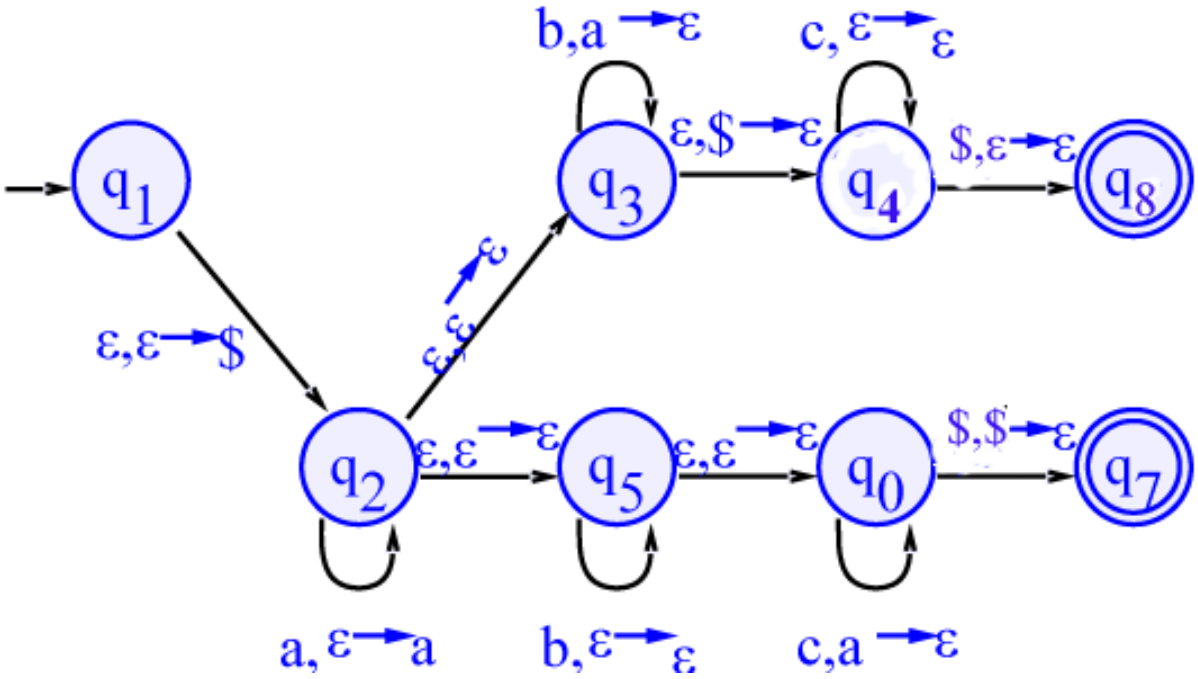
- Transition $a, b \rightarrow c$ means
  - if read a from input
  - and pop b from stack
  - then push c onto stack
- Meaning of ε transitions:
  - if $a = \varepsilon$, don't read inputs
  - if $b = \varepsilon$, don't pop any symbols
  - if $c = \varepsilon$, don't push any symbols

A PDA for Example 1



Accepts $L_1 = \{0^n 1^n | n \geq 0\}$.

A PDA for Example 2



Accepts $L_2 = \{a^i b^j c^k | i = j \vee i = k\}$

Typos:

- On arrow to q_7 should be $\varepsilon, \$ \rightarrow \varepsilon$
- On arrow to q_8 should be $\varepsilon, \varepsilon \rightarrow \varepsilon$

A PDA for Example 2 (cont.)

- Note: non-determinism is essential here!
- Unlike finite automata, non-determinism does add power.
- Let us try to think how a proof could go,
- ⋮
- and realize it does not seem trivial or immediate.
- One can show that $L = \{x^n y^n | n \geq 0\} \cup \{x^n y^{2n} \mid n \geq 0\}$ is accepted by a non-deterministic PDA, but not by a deterministic one.

Example 3 – Palindrome

A palindrome is a string w satisfying $w = w^{\mathcal{R}}$.

- "Madam I'm Adam"
- "Dennis and Edna sinned"
- "Red rum, sir, is murder"
- "Able was I ere I saw Elba"
- "In girum imus nocte et consumimur igni" (Latin: "we go into the circle by night, we are consumed by fire".)
- "$\nu\iota\psi o\nu\ \alpha\nu o\mu\eta\mu\alpha\tau\alpha\ \mu\eta\ \mu o\nu\alpha\nu\ o\psi\iota\nu$"
- Palindromes also appear in nature. For example as DNA restriction sites – short genomic strings over $\{A, C, T, G\}$, being cut by (naturally occurring) restriction enzymes.

A PDA for Example 3

- On input x, the PDA starts pushing x into stack.
- At some point, PDA guesses that the mid point of x was reached.
- Pops and compares to input, letter by letter.
- If end of input occurs together with emptying of stack, accept.
- This PDA accepts palindromes of *even length* over the alphabet (all lengths is easy modification).
- Again, non-determinism (at which point to make the switch) seems necessary.

PDA Languages

The Push-Down Automata Languages, L_{PDA}, is the set of all languages that can be described by some PDA:

- $L_{PDA} = \{L(M) : M \text{ is a PDA}\}$

It is immediate that $L_{PDA} \supsetneq L_{DFA}$ (every DFA is just a PDA that ignores the stack).

- $L_{CFG} \subseteq L_{PDA}$?
- $L_{PDA} \subseteq L_{CFG}$?
- $L_{PDA} = L_{CFG}$!!!

Equivalence Theorem

The CFG–PDA Equivalence Theorem

Theorem: A language is context free if and only if some pushdown automata accepts it.

This time (unlike the regular expression vs. regular languages theorem), the proofs of both the "if" part and the "only if" part are non trivial.

Proof sketch follows.

CFL $\longrightarrow$ PDA

Lemma: If a language is context free, then some pushdown automaton accepts it.

- Let A be a context-free language.
- By definition, A has a context-free grammar G generating it.
- On input w, the PDA P should figure out if there is a derivation of w using G.

Question: How does P figure out which substitution to make?

Answer: It guesses.

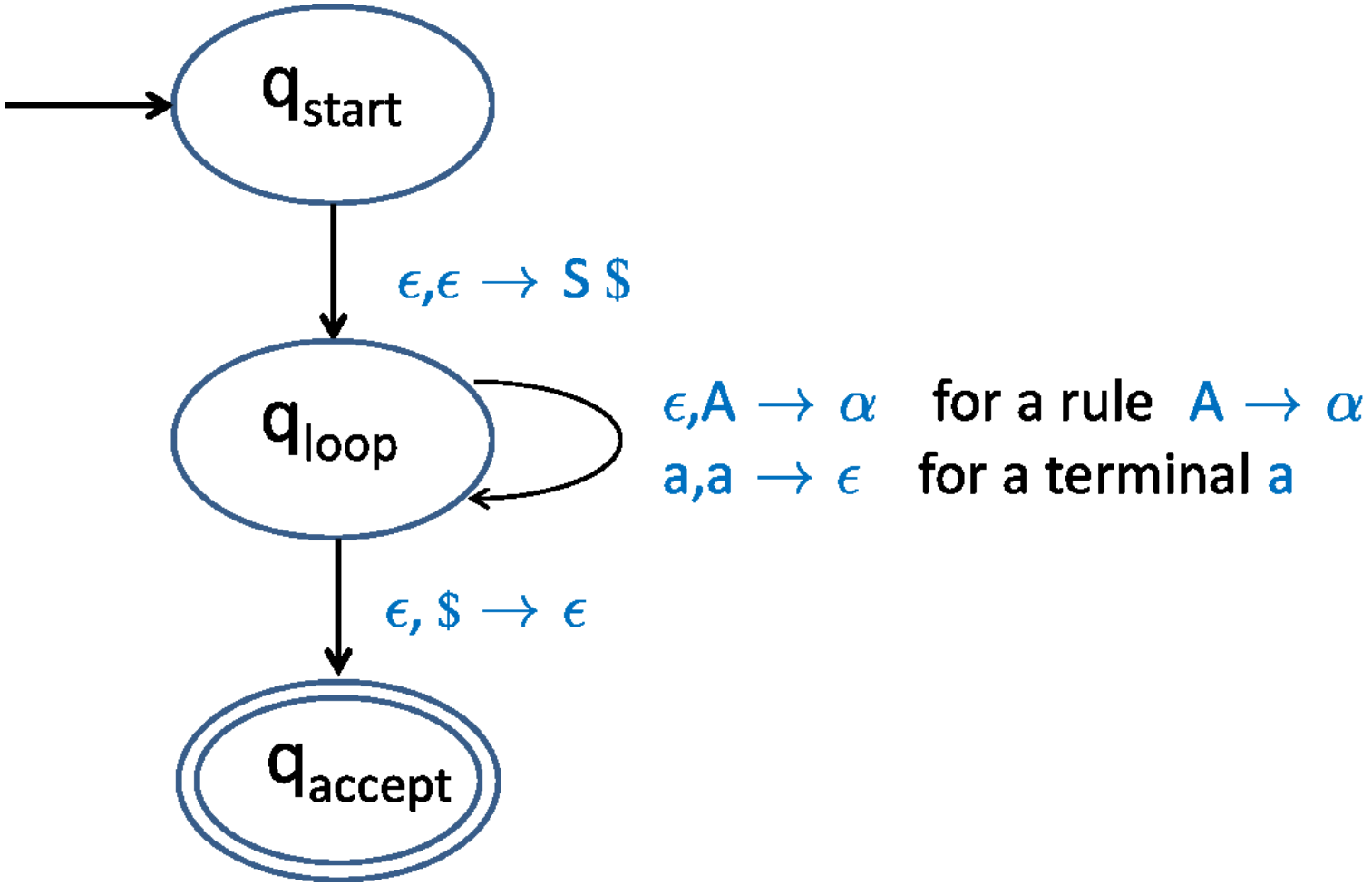
Simplifying Assumptions

- In a single move, the PDA can push a whole word into the stack (first letter at the top)
- Can we justify it?
- When deriving a word form a CFL, we always substitute the left most variable
- Does it change the derived language?

Informal Description of P

- Push $S\$$ on stack
- While top of the stack t is not $\$$:
- If t is variable A:
  non-deterministically select rule $A \to \alpha$ and substitute.
- If t is a terminal a:
  read next input and compare. If they differ, reject.
- Accept if end of input

State Diagram for P



Claim: $L(P) = L(G)$

- We prove that $S \stackrel{*}{\Rightarrow} \alpha$ iff $\alpha = \alpha_1\alpha_2$, and after reading the string α_1, it is possible for P to be at state q_{loop} with stack $\alpha_2\$$
- Does it prove the lemma?
- $\rightarrow$ is proved by induction on the number of derivations steps (of G)
- $\leftarrow$ is proved by induction on the number of transitions (of P)

Proof of board...

PDA $\longrightarrow$ CFL

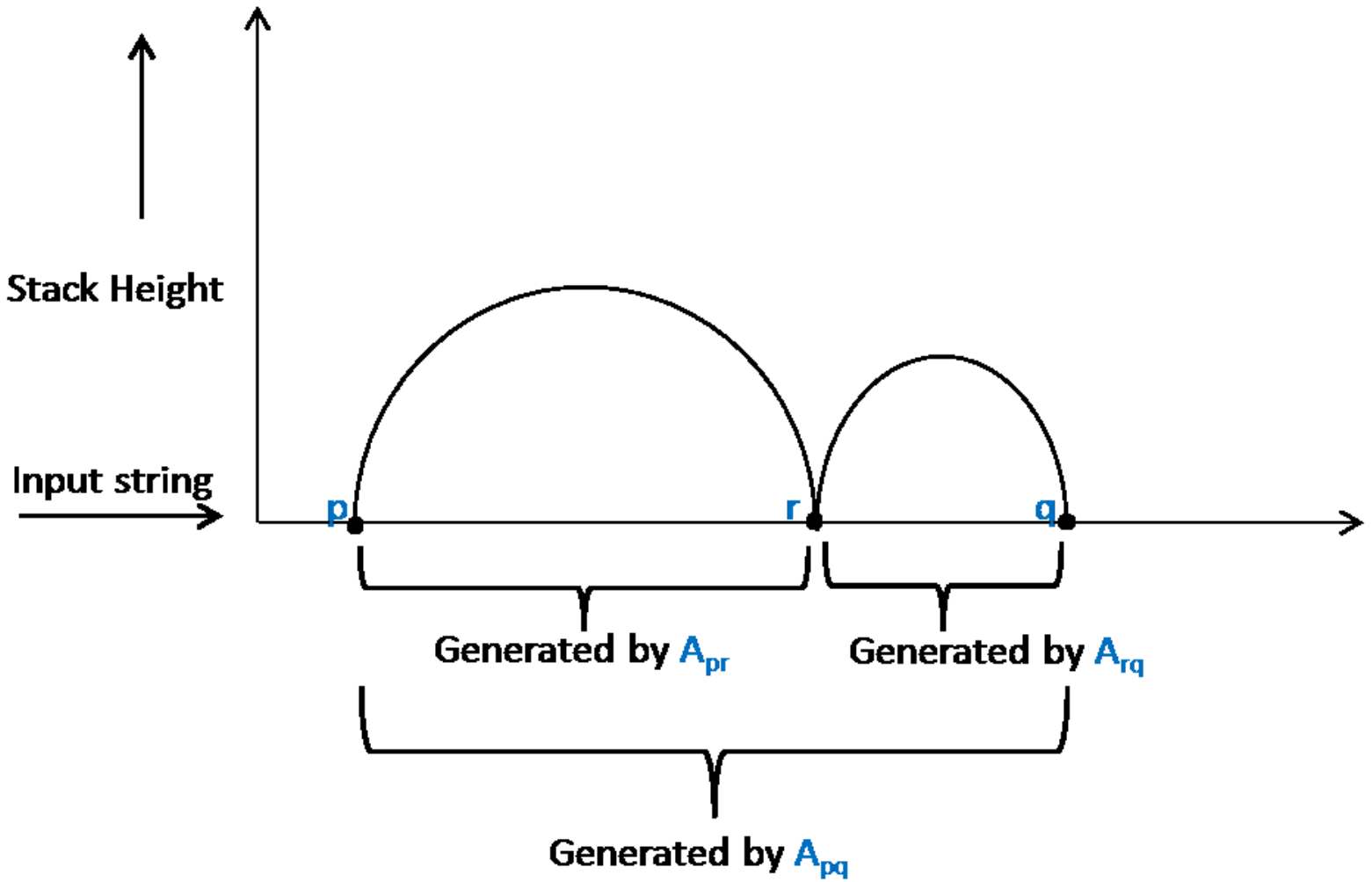
Lemma: If a PDA P accepts a language L, then L is CFL.

- Idea: for each pair of states p and q in P, we define a variable A_{pq} in the grammar G.
- A_{pq}, generates all strings that take P from p with an empty stack to q with an empty stack
- Start variable is A_{q_0,q_a} (assuming single accept state q_a)

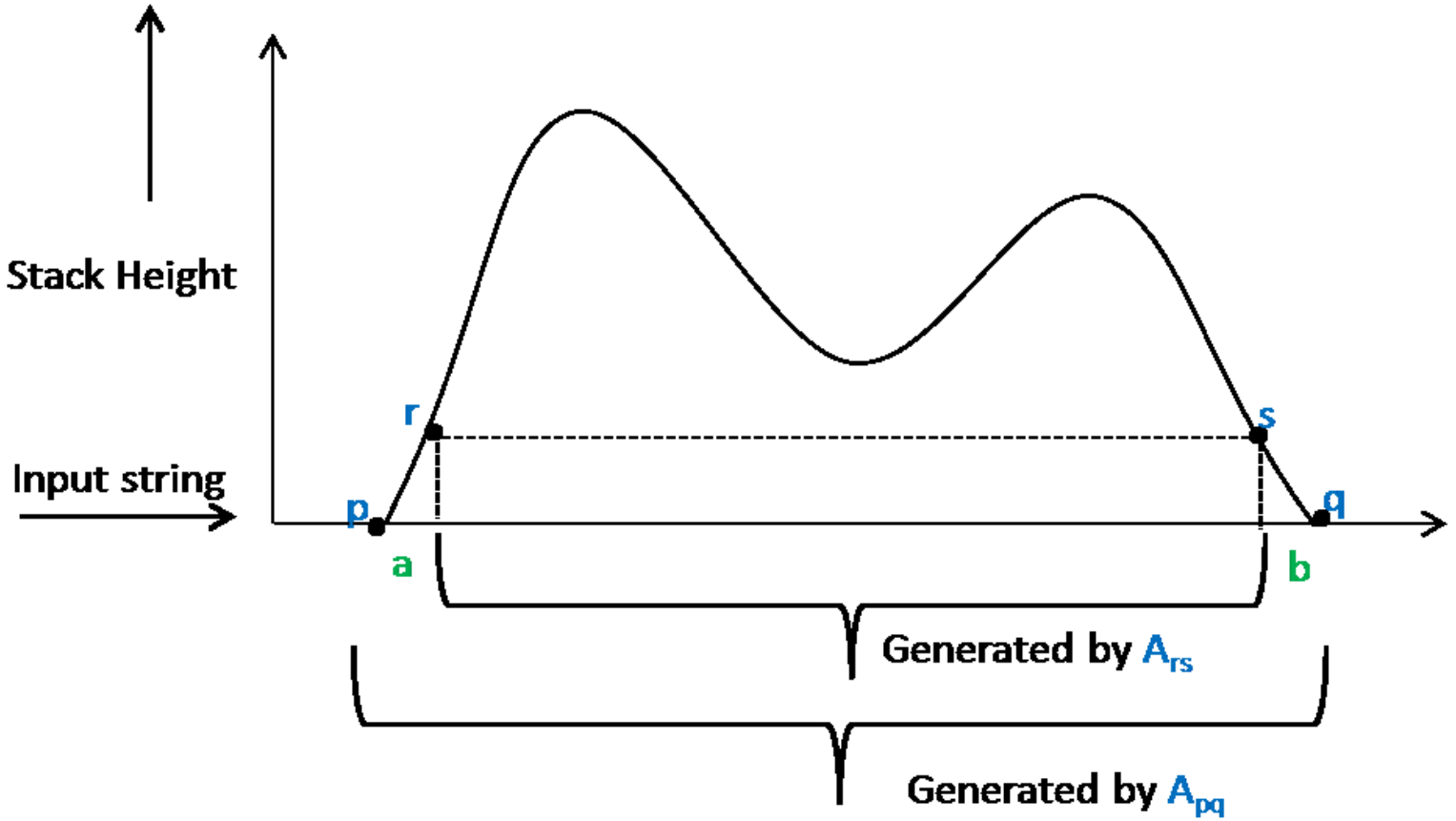
Defining G

- Simplifying assumptions
  - A single accepting states q_a.
  - P empties the stack before accepting
  - Each transition either pops or pushes
- Definition of G
  - For all $p, q, r \in Q$: add $A_{pq} \rightarrow A_{p,r} A_{r,q}$ to G
  - For all $p, q, r, s \in Q$, $a, b \in \Sigma_\varepsilon$ and $t \in \Gamma$: add $A_{pq} \rightarrow a A_{r,s} b$ to G, if $(r, t) \in \delta(p, a, \varepsilon)$ and $(q, \varepsilon) \in \delta(s, b, t)$
  - For all $p \in Q$, add $A_{pp} \rightarrow \varepsilon$ to G

PDA Computation corresponding to $A_{pq} \rightarrow A_{p,r}A_{r,q}$



PDA Computation corresponding to $A_{pq} \rightarrow aA_{r,s}b$



Claim: $L(G) = L(P)$

Proof by induction on the number of derivation rules/ transitions

The CFG–PDA Equivalence Theorem

Theorem: A language is context free if and only if some pushdown automata accepts it.

Closure Properties

Simple Closure Properties

- Context-Free Languages are closed under
  - Union: $S \rightarrow S_1 \mid S_2$
  - Concatenation: $S \rightarrow S_1 S_2$
  - Star: $S_{new} \rightarrow \varepsilon \mid S_{old} \mid S_{new} S_{new}$
- What about complement and intersection of context-free languages?

Intersection

$S_1 \rightarrow A_1 B_1$ $\quad$ $S_2 \rightarrow A_2 B_2$

$A_1 \rightarrow 0A_1 1|01$ $\quad$ $A_2 \rightarrow 0A_2|\varepsilon$

$B_1 \rightarrow 2B_1|\varepsilon$ $\quad$ $B_2 \rightarrow 1B_2 2|12$

$L_1 = 0^n 1^n 2^*$ $\quad$ $L_2 = 0^* 1^n 2^n$

- $L_1 \cap L_2 = 0^n 1^n 2^n$
- L_1 and L_2 are a context free languages (why?),
- But $L_1 \cap L_2$ is not a context free language.
- But can't we run two PDA's in parallel, and accept iff both accept??
- What about intersection of a CFL with a regular language?

When CFL Intersects Regular Language

- Are the context free languages context free languages closed under intersection with a regular language?
- That is, if L_1 is context free languages, and L_2 is regular, must $L_1 \cap L_2$ be context free languages?
- Run PDA L_1 and DFA L_2 "in parallel" (just like the intersection of two regular languages).
- Formal details omitted (but you should be able to figure them out).

Applications

- Is $L = \{(0 \cup 1 \cup 2)^* : \text{ \# of } 0\text{'s = \# of } 1\text{'s = \# of } 2\text{'s } \}$ context free?
  - $L \cap 0^*1^*2^* = \{0^n 1^n 2^n : n \geq 0\}$ which is not context free.
  - $0^*1^*2^*$ is regular.
  - Context free languages intersected with a regular languages are context free.
  - So L is not a context free language
- This could also be established using pumping lemma, but proof above is more elegant.

Complementation

The fact that CFLs are not closed under intersection, but are closed under union implies they are not closed under complementation, as $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$.

We give a simple example where L is not CFL but $\overline{L}$ is

- Take $L = \{ww \mid w \in \{0,1\}^*\}$.
- L is not a CFL (why?)
- We prove that $\overline{L}$ is CFL

Complementation cont.

- For any $y \in \overline{L}$, either
  - y's length is odd, or
  - y's length is even, 2ℓ, and $\exists i \geq 1$ such that $y_i \neq y_{\ell+i}$.
- We construct a PDA P for accepting $\overline{L}_{even}$ – the even length members of $\overline{L}$:
  Guess $k, j \geq 0$, and accept y if it is of the form:
  $\{0,1\}^k \, 0 \, \{0,1\}^k \, \{0,1\}^j \, 1 \, \{0,1\}^j$ or
  $\{0,1\}^k \, 1 \, \{0,1\}^k \, \{0,1\}^j \, 0 \, \{0,1\}^j$
- Claim: P is a PDA
- Claim: $L(P) = \overline{L}_{even}$ (we prove $L(P) \subseteq \overline{L}_{even}$)
- Assume $y = \{0,1\}^k \, 0 \, \{0,1\}^k \, \{0,1\}^j \, 1 \, \{0,1\}^j$, and let $\ell = k + j + 1$ and $i = k + 1$.
  Then $y_i = 0$ and $y_{\ell+i} = y_{2k+2+j} = 1$

Homomorphism and Inverse Homomorphism

- *Homomorphism:* replaces each letter with a word
- Example: $h(1) = aba$, $h(0) = aa$
  $h(010) = aa\,aba\,aa$
  $L_1 = (01)^*$, $h(L_1) = (aaaba)^*$.
- Claim: Assuming that L is a CFL, then so is $h(L)$
- Proof?
- *Inverse homomorphism:* $h^{-1}(w) = \{x | h(x) = w\}$,
  $h^{-1}(L) = \{x | h(x) \in L\}$
- Example: $L_2 = (ab + ba)^* a$, $h^{-1}(L_2) = \{1\}$.
- Claim: Assuming that L is a CFL, then so is $h^{-1}(L)$
- Proof?

CFL's are Closed Under Inverse Homomorphism

- Idea: let P be a PDA for L. On input word w, emulate $P(h(w))$
- But we cannot afford to store $h(w)$!
- Solution, compute $h(w)$ "on demand":
  - Initialized a "buffer" Buff to $h(a)$, where a is the first letter of the input string
  - Emulate a running of P with Buff as its input string. Each time Buff is fully read by P, set $\mathrm{Buff} = h(a)$, where a is the next input letter (if exists)
  - Accept, iff P does
- How do we implement Buff?

Emptiness of CFGs

Given a CFG, G, is $L(G) = \emptyset$?

In other words, is there any string w, such that G generate w?

Theorem: There is an algorithm that solves this problem (and always halts).

Possible approaches for a proof:

Bad Idea: We know how to test whether $w \in L(G)$ for any string w, so just try it for each w...

Better Idea: Can the start variable generate a string of terminals?

A more holistic approach: Can a particular variable generate a string of terminals?

Removing redundant variables and terminals of a CFG

1. Mark all terminal symbols in G.
2. Repeat until no new variable become marked:
   Mark any A where $A \rightarrow U_1 U_2 \ldots U_k$ and all U_i have already been marked.
4. Remove all unmarked variables, and any rule they appear in.
5. If S is removed, then $L(G) = \emptyset$.
6. Remove any variable A not reachable from S .
7. Remove any terminal which does not appear in some rule.

CFG Emptiness (2)

Algorithm: On input G (a CFG),

1. Remove redundant variables and terminals, get G'.
2. $L(G') = \emptyset$ iff S is removed

Finiteness of CFGs

Given a CFG, G, is $|L(G)|$ finite?

1. Remove redundant variables and terminals.
2. Create a graph where nodes are variables and directed edges are derivations.
3. $L(G)$ is infinite iff the graph has a cycle.

CFGs "Fullness"

Given a CFG, G, is $L(G) = \Sigma^*$?

We just saw an algorithm to determine, given a CFG, G, if $L(G) = \emptyset$

$L(G) = \Sigma^*$ iff $\overline{L(G)} = \emptyset$. Why not modify the algorithm so it determines emptiness of the complement?

Unfortunately, CFGs are not closed under complement.

Fact: There is no algorithm to solve this problem.

We are not prepared to prove this remarkable fact (yet).

CFGs Inherent Ambiguity

Given a CFG, G, is $L(G)$ inherently ambiguous?

This means that for any CFG that generates $L(G)$, there is a word in the language with two different parse trees.

Fact: There is no algorithm to solve this problem.

We will not prove this fact, yet you want to know it to put things in context.

When Are Two CFGs equivalent?

Is there an algorithm to solve this problem?

A Short Summary

- Regular Languages $\equiv$ Finite Automata.
- Context Free Languages $\equiv$ Push Down Automata.
- Closure properties of regular languages and of CFLs.
- Most algorithmic problems for finite automata are solvable.
- Some algorithmic problems for finite automata are not solvable.
- Pumping lemmata for both classes of languages.
- There are additional languages out there.

The View Over The Horizon