Theoretical Computer Science (Bridging Course)

Context Free Languages

Gian Diego Tipaldi

Topics Covered

- Context free grammars
- Pushdown automata
- Equivalence of PDAs and CFGs
- Non-context free grammars
- The pumping lemma

Context Free Grammars

- Extend regular expressions
- First studied for natural languages
- Often used in computer languages
  - Compilers
  - Parsers
- Pushdown automata

Context Free Grammars

- Collection of substitution rules
- Rules: Symbol -> string

- Variable symbols (Uppercase)
- Terminal symbols (lowercase)
- Start variable

Context Free Grammars

- Example grammar G1:

$$
\begin{aligned}
A &\to 0A1 \\
A &\to B \\
B &\to \#
\end{aligned}
$$

- A, B are variables
- 0,1,# are terminals
- A is the start variable

Context Free Grammars

Example string: 000#111

Does it belong to the grammar?

Context Free Grammars

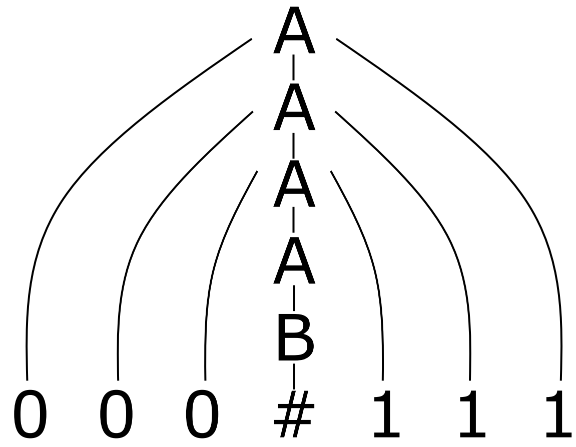
Example string: 000#111

- A -> 0A1
- 0A1 ->00A11
- 00A11 -> 000A111
- 000A111 -> 000B111
- 000B111 -> 000#111

Context Free Grammars

Example string: 000#111

- A -> 0A1
- 0A1 ->00A11
- 00A11 -> 000A111
- 000A111 -> 000B111
- 000B111 -> 000#111

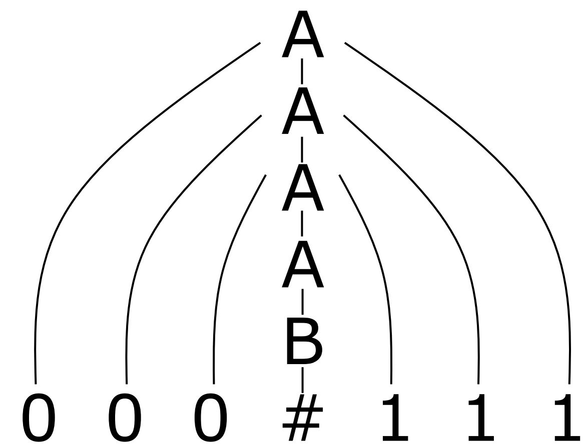


Parse tree for 000#111 in G_1

Context Free Grammars

Example string: 000#111

- A -> 0A1
- 0A1 ->00A11
- 00A11 -> 000A111
- 000A111 -> 000B111
- 000B111 -> 000#111



Parse tree for 000#111 in G_1

$$L(G_1) = \{0^n \# 1^n \mid n \geq 0\}$$

Natural Language Example

$$
\begin{array}{rcl}
\langle\textbf{SENTENCE}\rangle & \rightarrow & \langle\textbf{NOUN-PHRASE}\rangle\langle\textbf{VERB-PHRASE}\rangle \\
\langle\textbf{NOUN-PHRASE}\rangle & \rightarrow & \langle\textbf{CMPLX-NOUN}\rangle \mid \langle\textbf{CMPLX-NOUN}\rangle\langle\textbf{PREP-PHRASE}\rangle \\
\langle\textbf{VERB-PHRASE}\rangle & \rightarrow & \langle\textbf{CMPLX-VERB}\rangle \mid \langle\textbf{CMPLX-VERB}\rangle\langle\textbf{PREP-PHRASE}\rangle \\
\langle\textbf{PREP-PHRASE}\rangle & \rightarrow & \langle\textbf{PREP}\rangle\langle\textbf{CMPLX-NOUN}\rangle \\
\langle\textbf{CMPLX-NOUN}\rangle & \rightarrow & \langle\textbf{ARTICLE}\rangle\langle\textbf{NOUN}\rangle \\
\langle\textbf{CMPLX-VERB}\rangle & \rightarrow & \langle\textbf{VERB}\rangle \mid \langle\textbf{VERB}\rangle\langle\textbf{NOUN-PHRASE}\rangle \\
\langle\textbf{ARTICLE}\rangle & \rightarrow & \textbf{a} \mid \textbf{the} \\
\langle\textbf{NOUN}\rangle & \rightarrow & \textbf{boy} \mid \textbf{girl} \mid \textbf{flower} \\
\langle\textbf{VERB}\rangle & \rightarrow & \textbf{touches} \mid \textbf{likes} \mid \textbf{sees} \\
\langle\textbf{PREP}\rangle & \rightarrow & \textbf{with}
\end{array}
$$

- A boy sees
- The boy sees the flower
- A girl with the flower likes the boy

Context Free Grammar

Definition 2.2:

A context-free grammar is a 4-tuple

$$(V, \Sigma, R, S)$$

where:

- V is the set of variables
- Σ is the set of terminals, $\Sigma \cap V = \emptyset$
- R is the set of rules
- $S \in V$ is the start symbol

Language of a grammar

- u,v,w are strings, A->w a rule
- uAv yields uwv: uAv ⇒ uwv
- u derives v: u $\overset{*}{\Rightarrow}$ v if

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v$$

- Language of a grammar

$$\{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$$

Parsing a string

- Consider the following grammar

$$G_3 = (V, \Sigma, R, <Expr>)$$

$$V = \{<Expr>, <Term>, <Factor>\}$$

$$\Sigma = \{a, +, \times, (,)\}$$

$$R \text{ is}$$

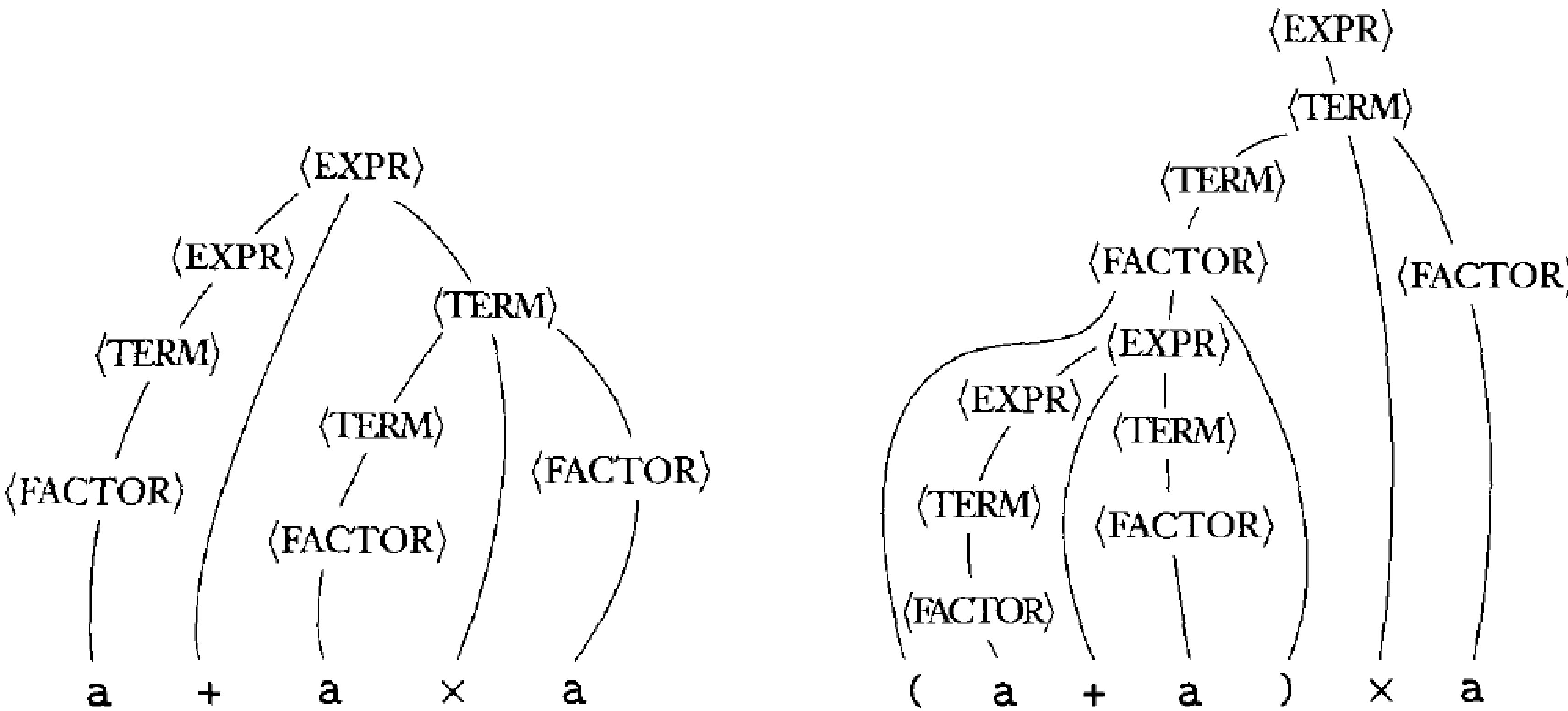
$$<Expr> \to <Expr> + <Term> | <Term>$$

$$<Term> \to <Term> \times <Factor> | <Factor>$$

$$<Factor> \to (<Expr>) \mid a$$

- What are the parse trees of
  - a + a x a
  - (a + a) x a

Parsing a string



Designing Grammars

Harder than designing automata

Few techniques can be used

- Union of context free languages
- Conversion from DFA (regular)
- Exploit linked variables ($0^n 1^n$)
- Exploit recursive structure (trickier)

Union of Different CFGs

$$S_1 \rightarrow 0S_11 \mid \varepsilon \qquad L(G_1) = \{0^n1^n \mid n \geq 0\}$$

$$S_2 \rightarrow 1S_20 \mid \varepsilon \qquad L(G_2) = \{1^n0^n \mid n \geq 0\}$$

$$S \rightarrow S_1 \mid S_2 \qquad L(G) = L(G_1) \cup L(G_2)$$

Conversion from DFAs

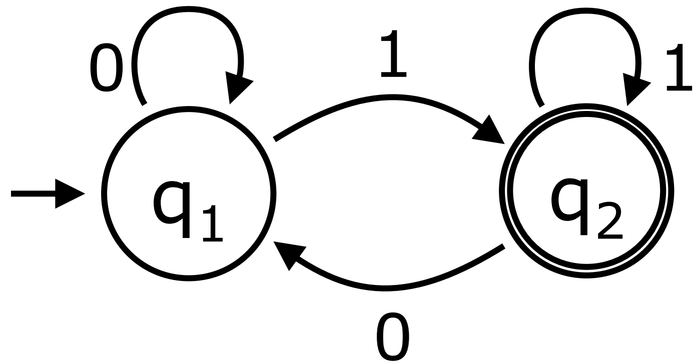
- Take the same vocabulary: $\Sigma_g = \Sigma_a$
- For each state q_i insert a variable R_i
- For each transition $\delta(q_i, a) = q_j$ insert

$$R_i \rightarrow aR_j$$

- For each accept state q_k insert

$$R_k \rightarrow \epsilon$$

Conversion from DFAs



- Take the same vocabulary: $\Sigma = \{0,1\}$
- Insert all the variables: $V = \{R_1, R_2\}$
- Insert the rules:

$$R_1 \to 0R_1, \qquad R_1 \to 1R_2$$
$$R_2 \to 0R_1, \qquad R_2 \to 1R_2$$
$$R_2 \to \epsilon$$

Designing Linked Strings

- Languages of the type

$$L(G_1) = \{0^n 1^n \mid n \geq 0\}$$

- Create rules of the form

$$R \to uRv$$

- For the language above

$$S \to 0S1 \mid \epsilon$$

Designing Recursive Strings

- Example are arithmetic expressions

$$< Expr > \rightarrow < Expr > + < Term > | < Term >$$

$$< Term > \rightarrow < Term > \times < Factor > | < Factor >$$

$$< Factor > \rightarrow (< Expr >) \mid a$$

- Create the recursive structure <Expr>
- Place it where it appear <Factor>

Ambiguity

- Generate a string in several ways
- E.g., grammar G5:

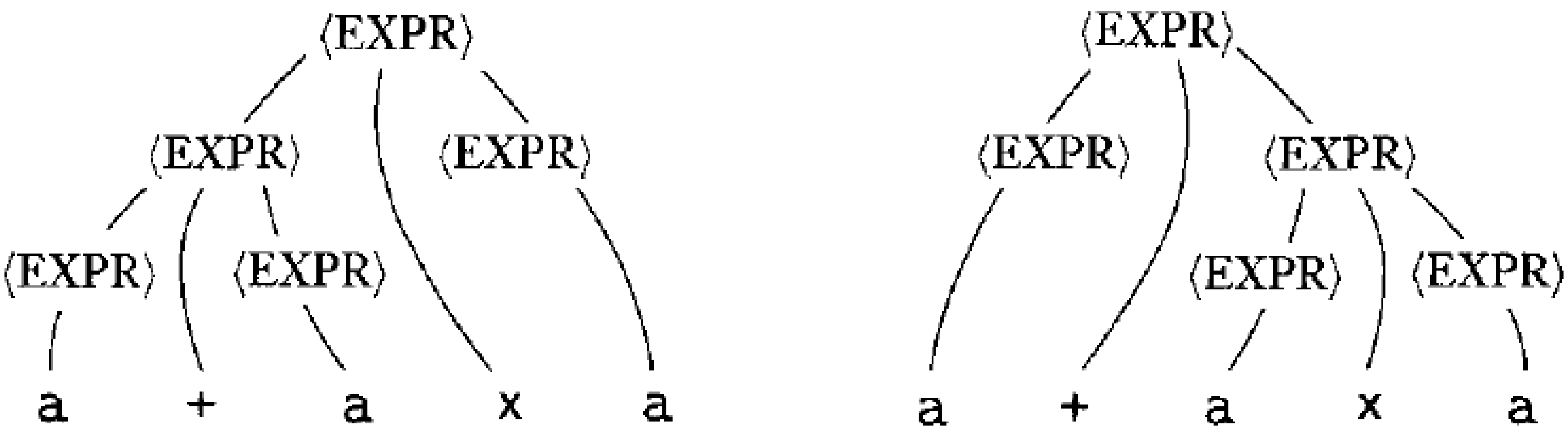
$$< Expr > \rightarrow < Expr > + < Expr > | < Expr > \times < Expr > | (< Expr >) | a$$

- No usual notion of precedence

- Natural language processing
- “a boy touches a girl with the flower”

Ambiguity

- Consider the string: a + a x a



Ambiguity – Definition

- Leftmost derivation: At every step, replace the leftmost variable
- A string is generated ambiguously if it has multiple leftmost derivations
- A CFG is ambiguous if generates some string ambiguously
- Some context free languages are inherently ambiguous

$$\{0^i 1^j 2^k \mid i = j \text{ or } j = k\}$$

Chomsky Normal Form (CNF)

<u>Definition 2.8:</u>

A context-free grammar is in Chomsky normal form if every rule is of the form

$$A \rightarrow BC$$
$$A \rightarrow a$$

where a is any terminal and A,B, and C are any variables—except that B and C may not be the start variable.

In addition we permit the rule $S \rightarrow \varepsilon$, where S is the start variable.

Chomsky Normal Form (CNF)

Theorem 2.9:

Any context-free language is generated by a context-free grammar in Chomsky normal form.

Proof Idea

- Rewrite the rules not in CNF
- Introduce new variables

- Four cases:
  - Start variable on the right side
  - Epsilon rules: $A \to \varepsilon$
  - Unit rules: $A \to B$
  - Long and/or mixed rules: $A \to aAbbBaB$

Proof Idea

- Start variable on the right side
  - Introduce a new start and $S_1 \rightarrow S_0$
- Epsilon rules: $A \rightarrow \varepsilon$
  - Introduce new rules without A
- Unit rules: $A \rightarrow B$
  - Replace B with its production
- Long and/or mixed rules: $A \rightarrow aAbbBaB$
  - New variables and new rules

Formal Proof: by Construction

1. Add a new start symbol S_0 and the rule $S_0 \rightarrow S$, where S is the old start
2. Remove all rules $A \rightarrow \epsilon$:
   - For each $R \rightarrow uAv$ add $R \rightarrow uv$
   - For each $R \rightarrow A$ add $R \rightarrow \epsilon$
   - Repeat until all gone (keep $S_0 \rightarrow \epsilon$)
3. Remove all rules $A \rightarrow B$:
   - For each $B \rightarrow u$ add $A \rightarrow u$
   - Repeat until all gone

Formal Proof: by Construction

4. Convert all rules $A \rightarrow u_1 \dots u_k, k \geq 3$ in:
   - $A \rightarrow u_1 A_1$
   - $A_1 \rightarrow u_2 A_2, \dots$
   - $A_{k-2} \rightarrow u_{k-1} u_k$
5. Convert all rules $A \rightarrow u_1 u_2$:
   - Replace any terminal u_i with U_i
   - Add the rules $U_i \rightarrow u_i$

- Be careful of cycles!

CNF: Example 2.10 from Book

- Convert the CFG in CNF

$$S \to ASA \mid aB$$
$$A \to B \mid S$$
$$B \to b \mid \varepsilon$$

- Added rules in **bold**
- Removed rules in ~~stroke~~

CNF: Example 2.10 from Book

- Add the new start symbol

$$\boldsymbol{S_0 \rightarrow S}$$
$$S \rightarrow ASA \mid aB$$
$$A \rightarrow B \mid S$$
$$B \rightarrow b \mid \varepsilon$$

CNF: Example 2.10 from Book

- Remove the empty rule $B \rightarrow \varepsilon$

$$
\begin{aligned}
S_0 &\rightarrow S \\
S &\rightarrow ASA \mid aB \mid \boldsymbol{a} \\
A &\rightarrow B \mid S \mid \boldsymbol{\varepsilon} \\
B &\rightarrow b \mid \cancel{\varepsilon}
\end{aligned}
$$

CNF: Example 2.10 from Book

- Remove the empty rule $A \rightarrow \varepsilon$

$$S_0 \rightarrow S$$

$$S \rightarrow ASA \mid aB \mid a \mid \boldsymbol{SA} \mid \boldsymbol{AS} \mid \boldsymbol{S}$$

$$A \rightarrow B \mid S \mid \cancel{\varepsilon}$$

$$B \rightarrow b$$

CNF: Example 2.10 from Book

- Remove unit rule: $S \to S$

$$S_0 \to S$$
$$S \to ASA \mid aB \mid a \mid SA \mid AS \mid \cancel{S}$$
$$A \to B \mid S$$
$$B \to b$$

CNF: Example 2.10 from Book

- Remove unit rule: $S_0 \rightarrow S$

$$S_0 \rightarrow \cancel{S} \mid \boldsymbol{ASA} \mid \boldsymbol{aB} \mid \boldsymbol{a} \mid \boldsymbol{SA} \mid \boldsymbol{AS}$$

$$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$$

$$A \rightarrow B \mid S$$

$$B \rightarrow b$$

CNF: Example 2.10 from Book

- Remove unit rule: $A \rightarrow B$

$$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$$
$$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$$
$$A \rightarrow \cancel{B} \mid S \mid \boldsymbol{b}$$
$$B \rightarrow b$$

CNF: Example 2.10 from Book

- Remove unit rule: $A \rightarrow S$

$$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$$
$$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$$
$$A \rightarrow \cancel{S} \mid b \mid \boldsymbol{ASA} \mid \boldsymbol{aB} \mid \boldsymbol{a} \mid \boldsymbol{SA} \mid \boldsymbol{AS}$$
$$B \rightarrow b$$

CNF: Example 2.10 from Book

- Convert the remaining rules

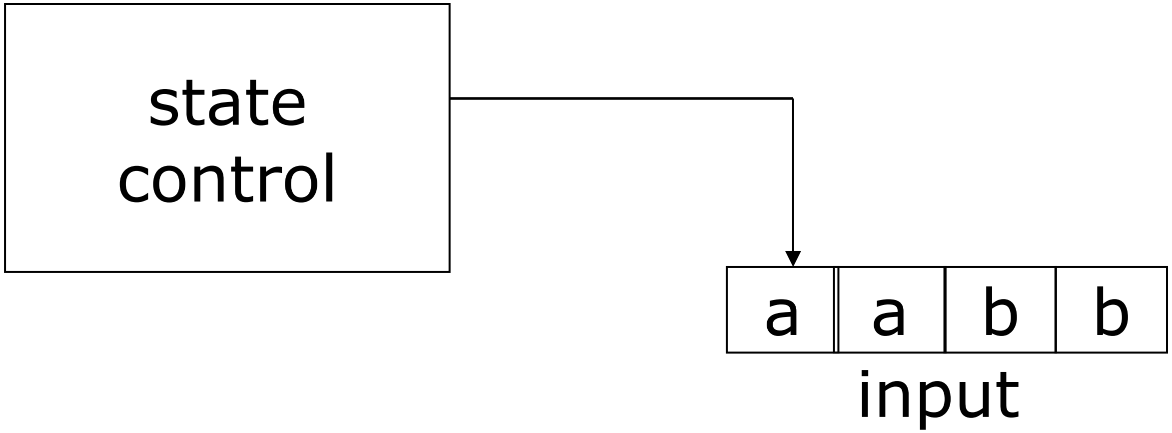
$$S_0 \rightarrow A\boldsymbol{A_1} \mid \boldsymbol{U}B \mid a \mid SA \mid AS$$
$$S \rightarrow A\boldsymbol{A_1} \mid \boldsymbol{U}B \mid a \mid SA \mid AS$$
$$A \rightarrow b \mid A\boldsymbol{A_1} \mid \boldsymbol{U}B \mid a \mid SA \mid AS$$
$$\boldsymbol{A_1} \rightarrow \boldsymbol{SA}$$
$$\boldsymbol{U} \rightarrow \boldsymbol{a}$$
$$B \rightarrow b$$

Pushdown Automata (PDA)

- Extend NFAs with a stack
- The stack provides additional memory
- Equivalent to context free grammars
- They recognize context free languages

Finite State Automata

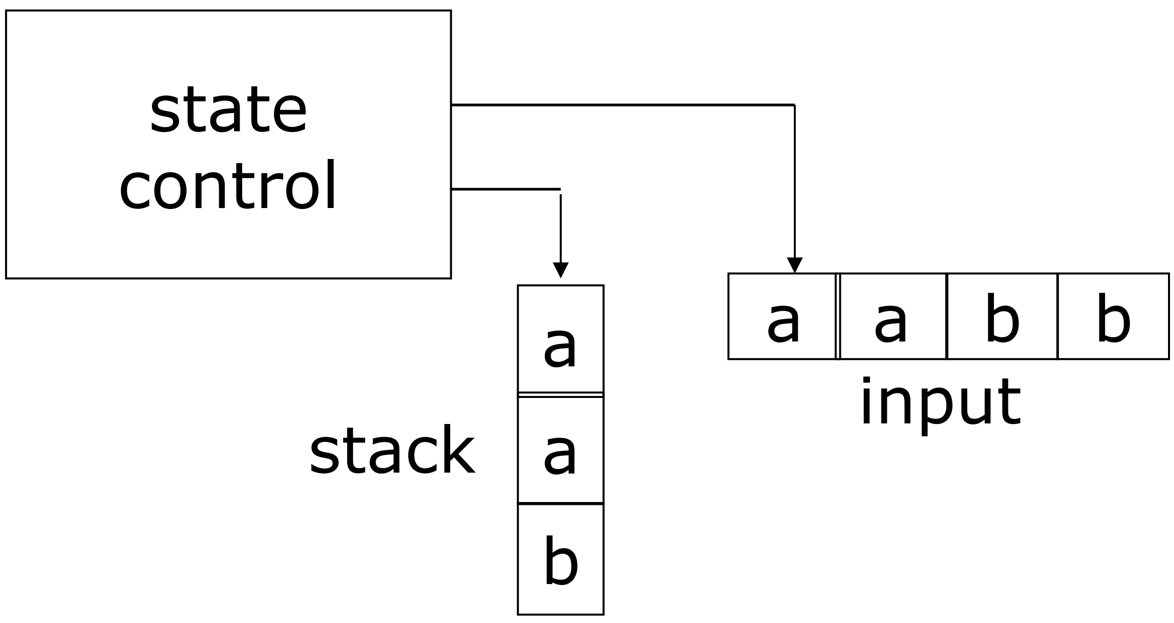
- Can be simplified as follow



- State control for states and transitions
- Tape to store the input string

Pushdown Automata

- Introduce a stack component



- Symbols can be read and written there

What is a Stack?

- Stacks are special containers
- Symbols are “pushed” on top
- Symbols can be “popped” from top
- Last in first out principle
- Similar to plates in cafeteria

Formal Definition of PDA

A pushdown automata is a 6-tuple

$$(Q, \Sigma, \Gamma, \delta, q_o, F)$$

- Q is a finite set of states
- Σ is a finite set, the input alphabet
- Γ is a finite set, the stack alphabet
- $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow P(Q \times \Gamma_\epsilon)$ is the transition function
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is the set of accept states

Transition Function

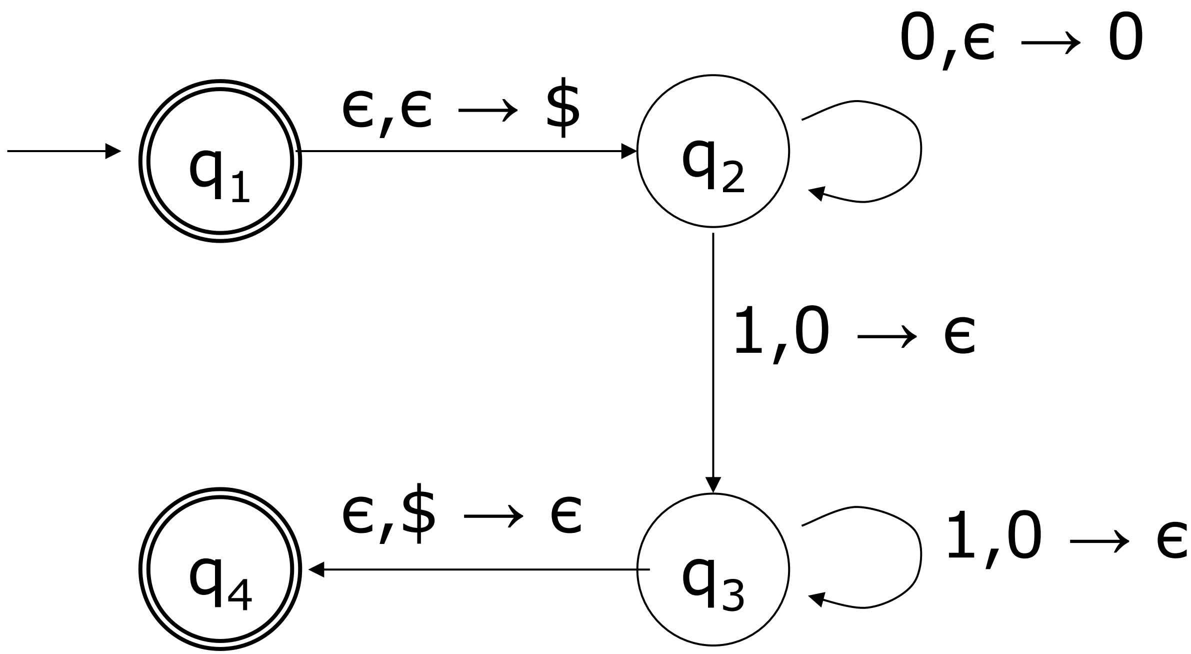
- Maps (state, in, stk) in (state, stk)
- Can include empty symbols
- $ is used to indicate the stack end

Input	**0**			**1**			**ϵ**		
Stack	**0**	**$**	**ϵ**	**0**	**$**	**ϵ**	**0**	**$**	**ϵ**
q_1									$\{(q_2,\$)\}$
q_2			$\{(q_2,0)\}$	$\{(q_3,\epsilon)\}$					
q_3				$\{(q_3,\epsilon)\}$				$\{(q_4,\epsilon)\}$	
q_4									

Example PDA

- PDA for the language

$$L(G_1) = \{0^n 1^n \mid n \geq 0\}$$



Computation of the PDA

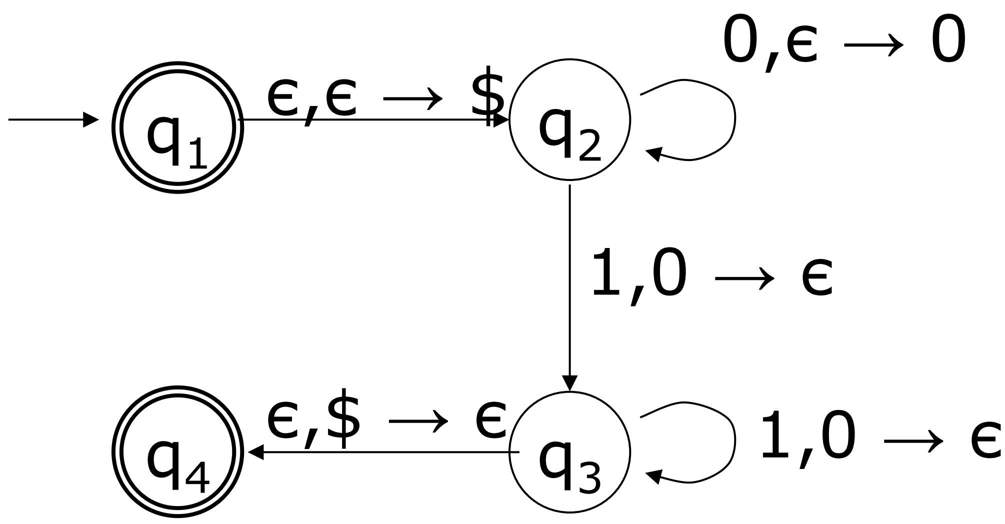
Compute keeping track of

- String
- State
- Stack

Computation of the PDA

Compute keeping track of

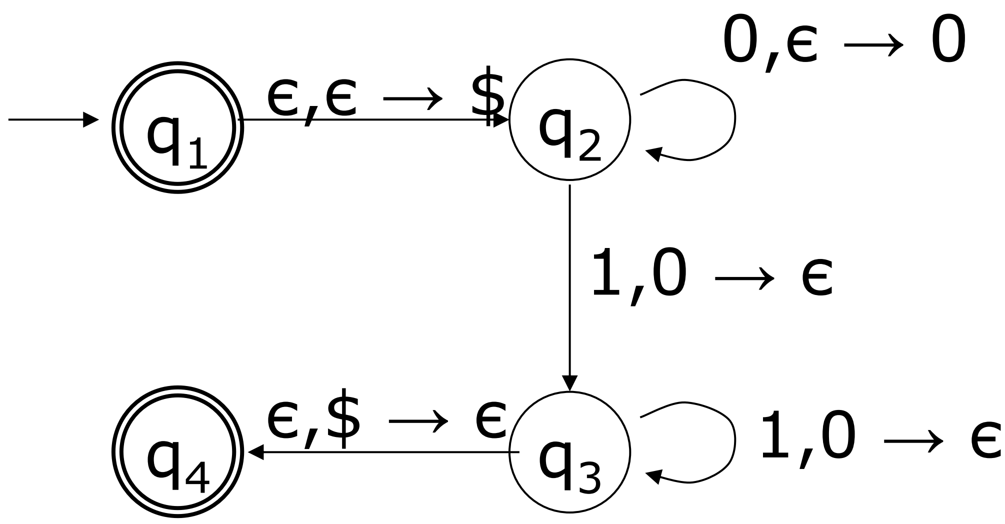
- String
- State
- Stack



Computation of the PDA

Compute keeping track of

- String
- State
- Stack



$(0011, q_1, \varepsilon)$

$\downarrow$

$(0011, q_2, \$)$

$\downarrow$

$(011, q_2, 0\$)$

$\downarrow$

$(11, q_2, 00\$)$

$\downarrow$

$(1, q_3, 0\$)$

$\downarrow$

$(\varepsilon, q_3, \$)$

$\downarrow$

(q_4, ε) accept

Definition of Computation

Let M be a pushdown automaton $(Q, \Sigma, \Gamma, \delta, q_0, F)$

Let $w = w_1 \ldots . w_n$ be a string over Σ

M accepts w if $w \in \Sigma^*$ and $w = w_1 \ldots . w_n$ where $w_i \in \Sigma_\varepsilon$ and a sequence of states $r_0, \ldots, r_n$ exists in Q and strings $s_0, \ldots, s_n$ exists in Γ^* such that

1. $r_0 = q_0$ and $s_0 = \varepsilon$
2. for all $i = 0, \ldots, n-1$

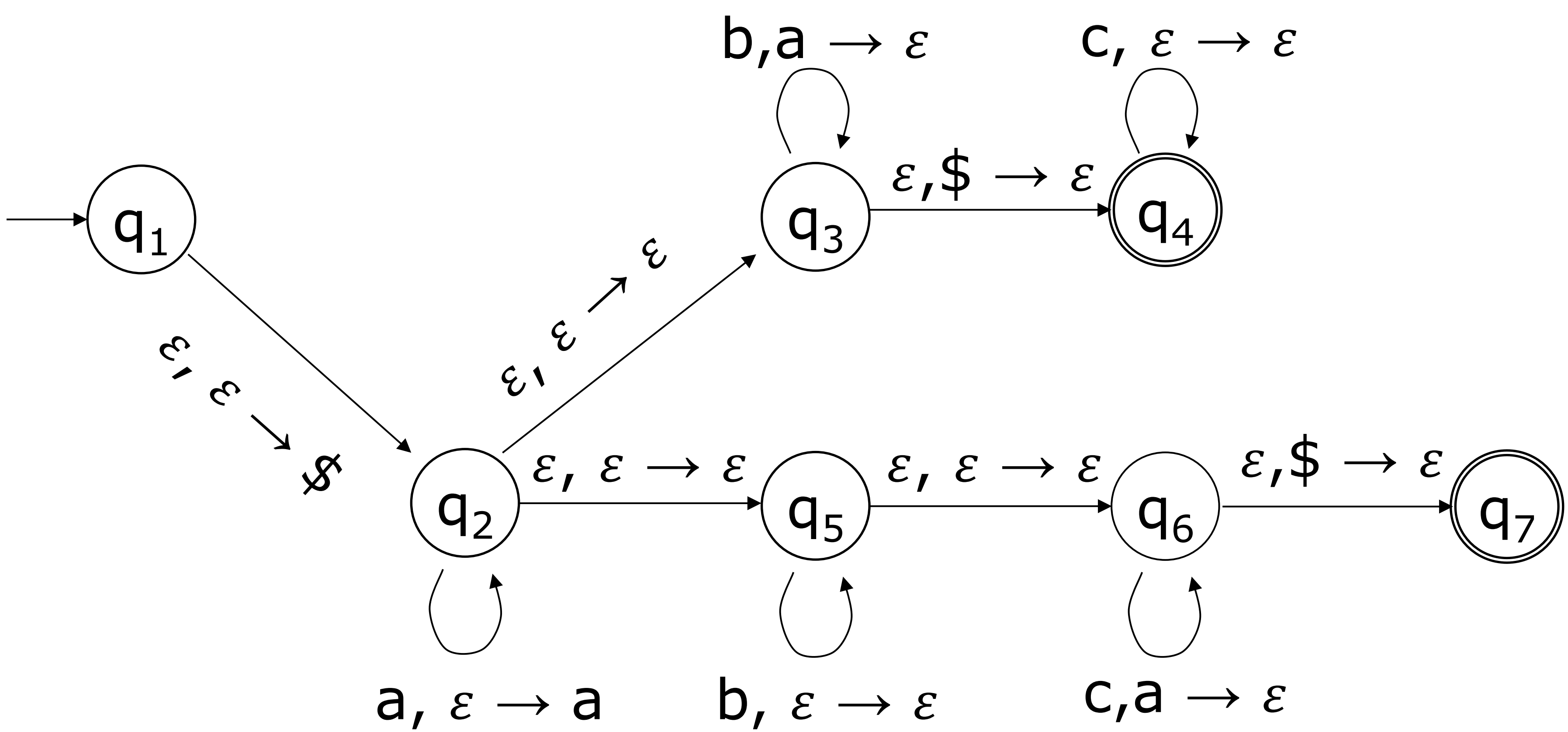
   $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ where $s_i = at$ and $s_{i+1} = bt$

   for some $a, b \in \Gamma_\varepsilon$ and some $t \in \Gamma^*$
3. $r_n \in F$

No explicit test for empty stack and end of input

Another Example of PDA

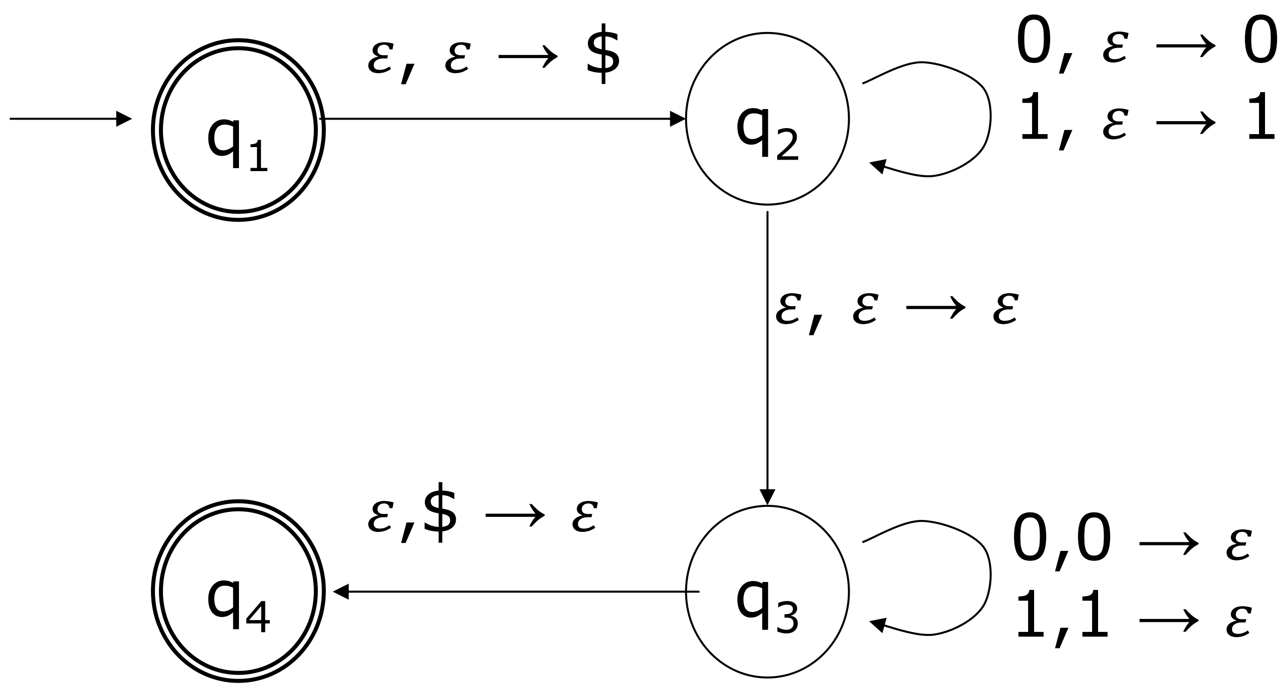
$$L = \{a^i\, b^j\, c^k \mid i, j, k \geq 0 \; and \; i \; = \; j \; or \; i \; = \; k\}$$



Another Example of PDA

$$L = \{ww^R \mid w \in \{0,1\}^*\}$$

w^R is w written "backwards"



Equivalence of PDAs and CFLs

Theorem 2.20:

A language is context free if and only if some pushdown automaton recognizes it.

Lemma 2.21:

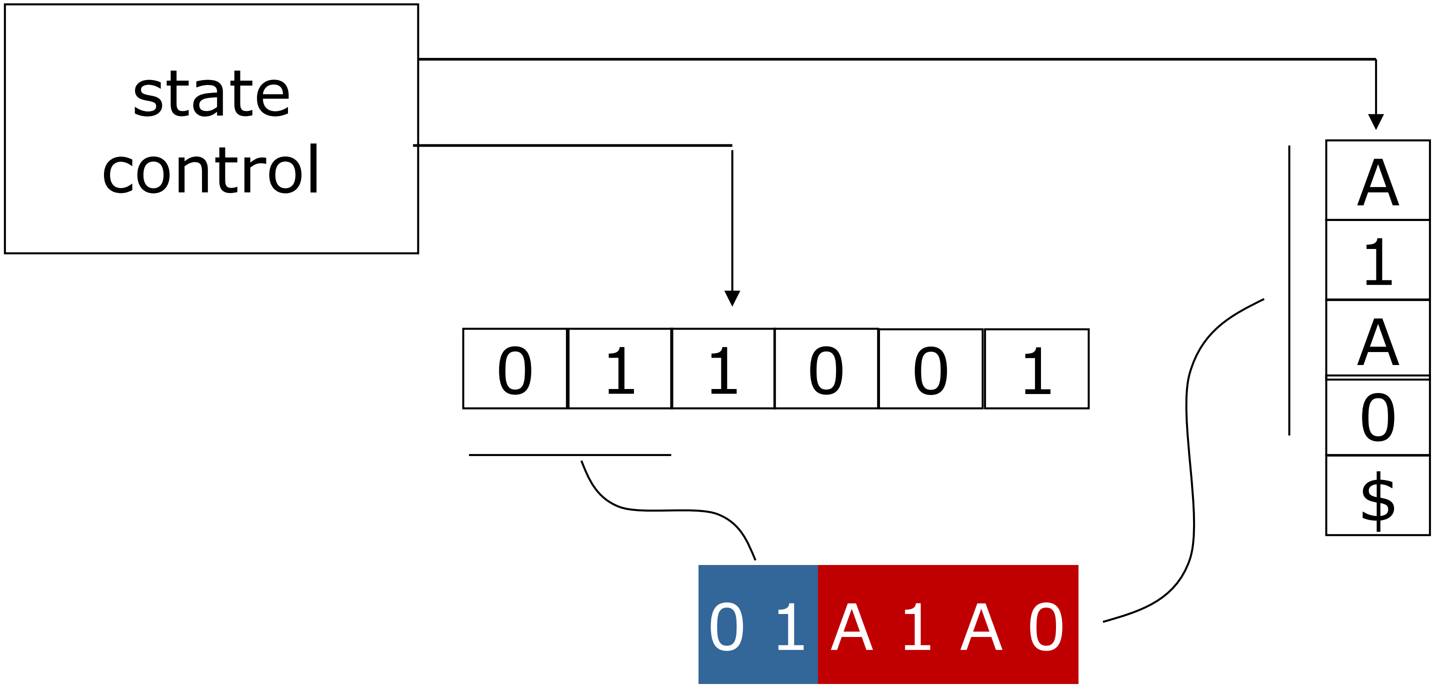
If a language is context free, then some pushdown automaton recognizes it. (Forward direction of proof)

Lemma 2.21: Proof Idea

- Construct a PDA P for the grammar
- P accepts w if there is a derivation
- Non determinism for multiple rules
- Represent intermediate strings on PDA
- Store the variables on the stack

Lemma 2.21: Proof Idea

- Representing 01A1A0



Proof by Construction

1. Place the marker symbol $ and the start variable on the stack.
2. Repeat the following steps forever. There are three possible cases:
   a. The top of stack is a variable symbol A;
   b. The top of stack is a terminal symbol a;
   c. The top of stack is the symbol $

Proof by Construction

The top of stack is a variable symbol A

Non-deterministically select one of the rules for A and substitute A on the stack.

The top of stack is a terminal symbol a

Read the next symbol from the input and compare it to a. If they match, repeat. If they do not match, reject the branch.

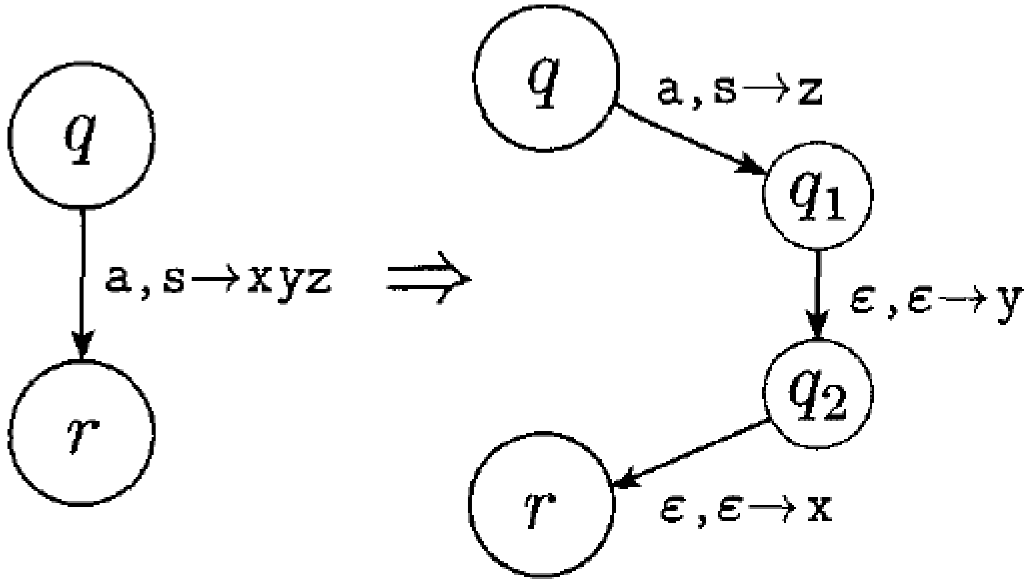
Proof by Construction

The top of stack is the symbol $

Enter the accept state. Doing so accepts the input if it has all been read.

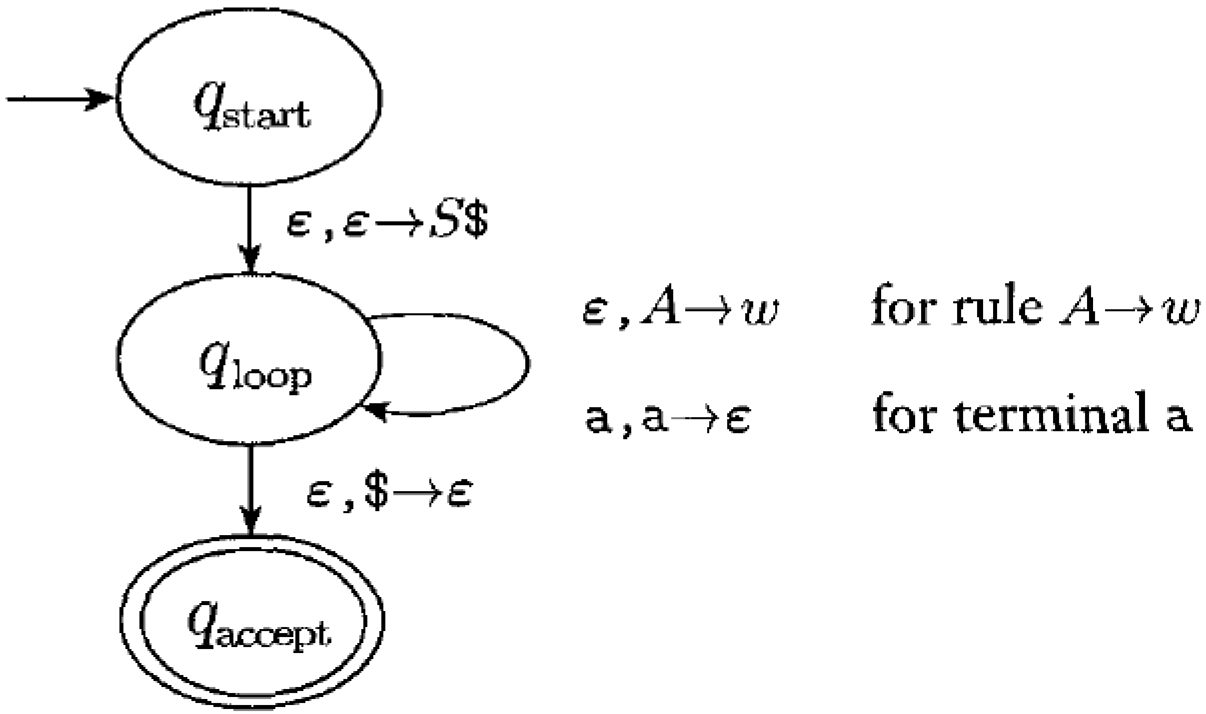
Proof by Construction

- PDA to substitute a whole string



Proof by Construction

- Final PDA to accept the string



Example 2.25 From the Book

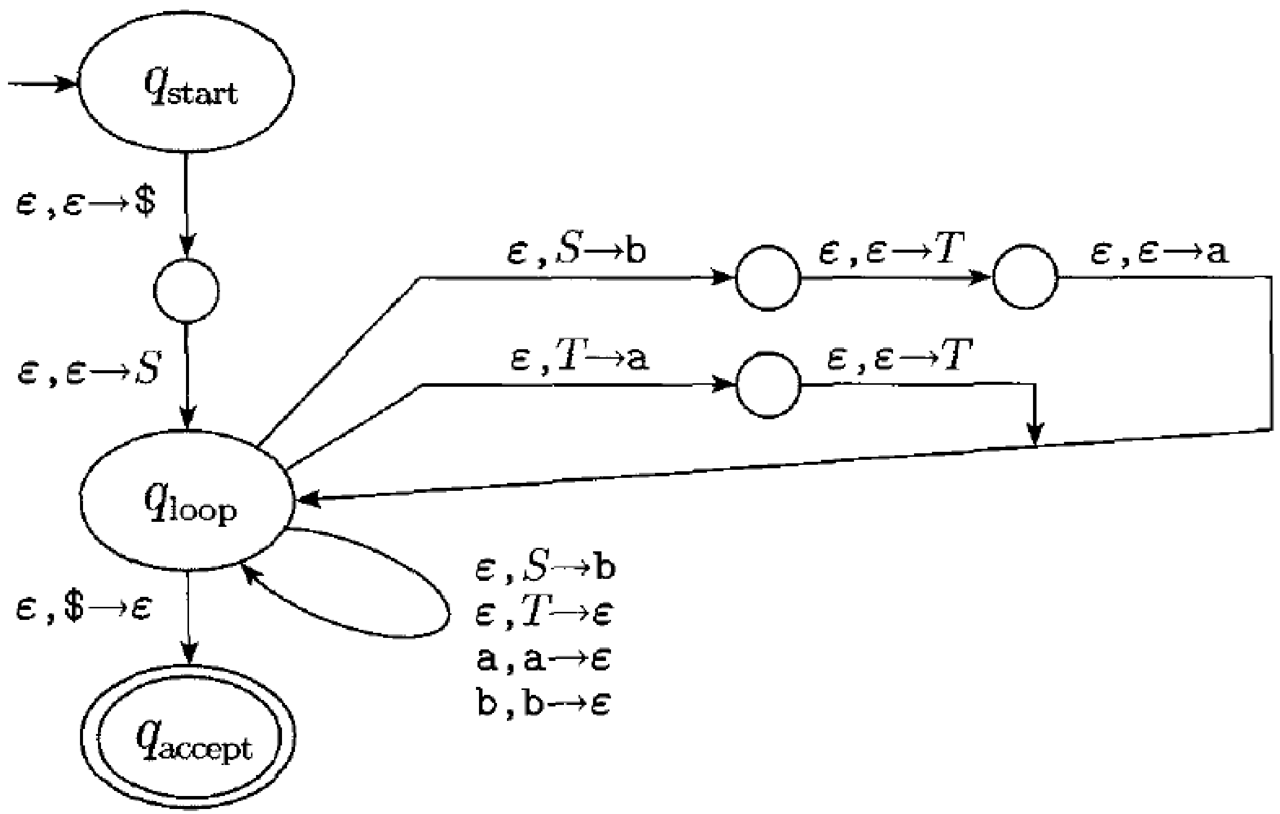
- Construct a PDA to accept the CFG

$$S \to aTb \mid b$$
$$T \to Ta \mid \varepsilon$$

Example 2.25 From the Book

- Construct a PDA to accept the CFG

$$S \to aTb \mid b$$
$$T \to Ta \mid \varepsilon$$



Equivalence of PDAs and CFLs

<u>Lemma 2.27:</u>

If a pushdown automaton recognizes some languages, then it is context free. (Backward direction of proof)

Assumptions:

1. The PDA has a single accept state
2. The PDA empties the stack before accepting
3. Transitions either push or remove symbols

Lemma 2.27: Assumptions

- Assumption 1
  - Create a new accept state with empty transitions from the previous ones
- Assumption 2
  - Creates dummy transitions to empty the stack before accepting

Lemma 2.27: Assumptions

- Assumption 3
  - Replace each transitions that pushes and pops with two transitions and a new state
  - Replace each transitions without push and pop with two transitions that push and pop a dummy symbol and a new state

Lemma 2.27: Proof

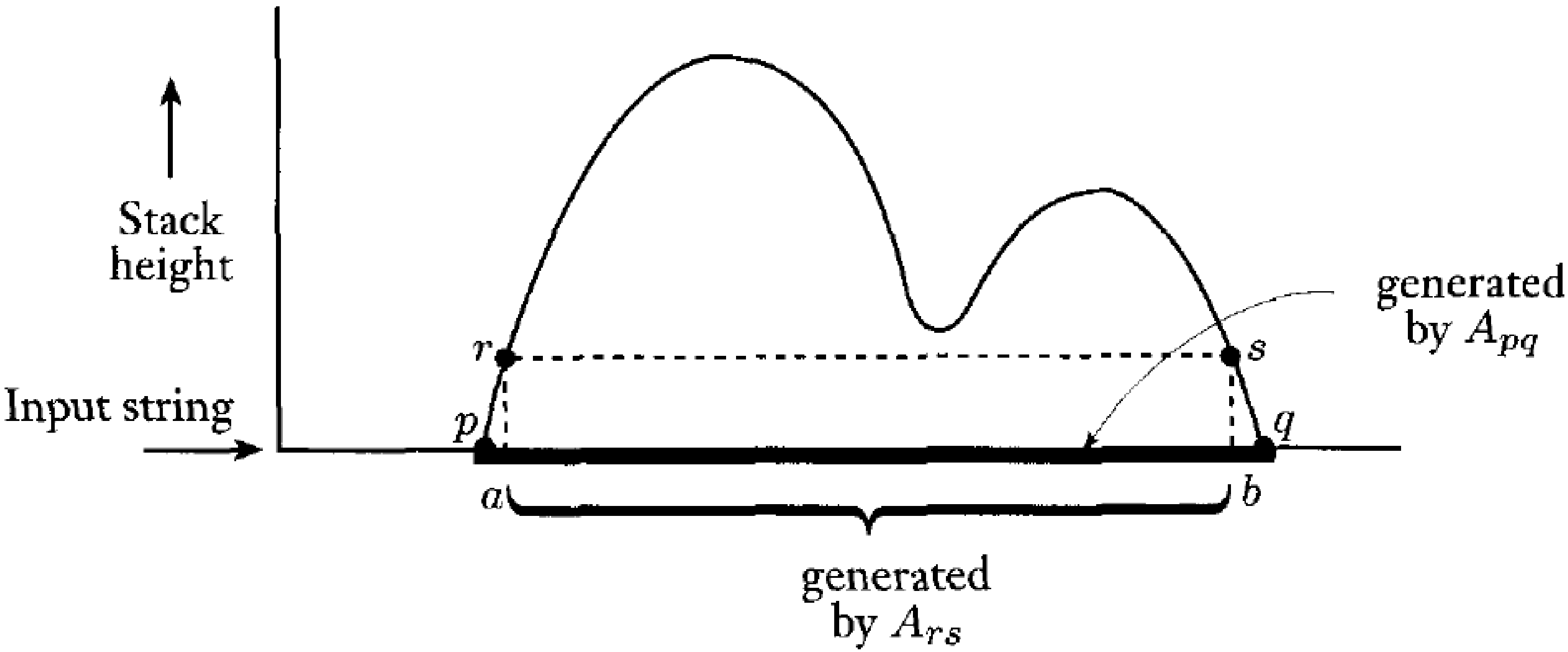
Say that $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$ and construct G. The variables of G are $\{A_{pq} \mid p,q \in Q\}$. The start variable is $A_{q_0, q_{accept}}$.

Now we describe G´s rules.

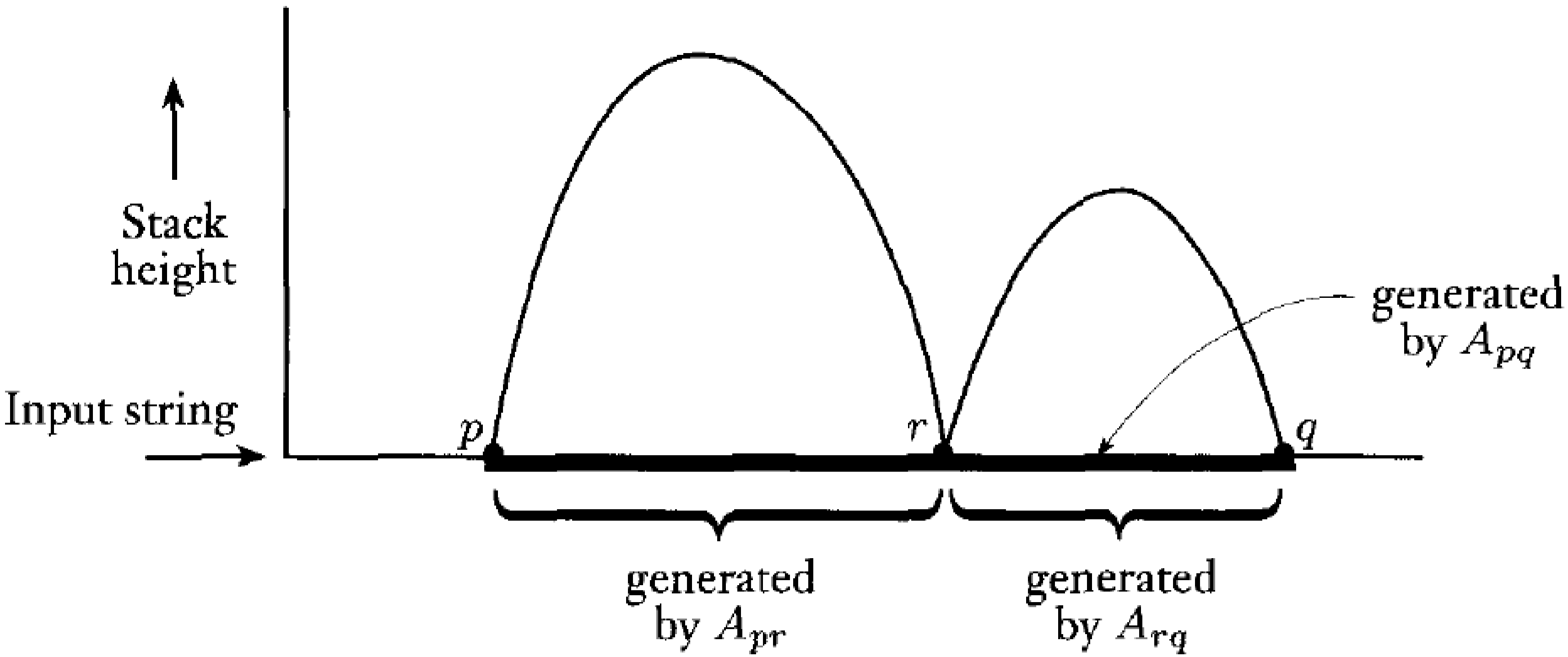
- For each $p,q,r,s \in Q; t \in \Gamma$, and $a,b \in \Sigma_\varepsilon$, if $\delta(p,a,\varepsilon)$ contains (r,t) and $\delta(s,b,t)$ contains (q,ε) put the rule $A_{pq} \rightarrow aA_{rs}b$ in G.
- For each $p,q,r \in Q$ put the rule $A_{pq} \rightarrow A_{pr}A_{rq}$ in G.
- Finally, for each $p \in Q$ put the rule $A_{pp} \rightarrow \varepsilon$ in G.

You may gain some intuition for this construction from the following figures.

Inserting $A_{pq} \rightarrow aA_{rs}b$



Inserting $A_{pq} \to A_{pr}A_{rq}$



Lemma 2.27: Proof

- We now need to prove that the construction works
- $\boldsymbol{A_{pq}}$ generates $\boldsymbol{x}$ iff $\boldsymbol{x}$ brings $\boldsymbol{P}$ from $\boldsymbol{p}$ with an empty stack to $\boldsymbol{q}$ with an empty stack
- Prove by induction

Lemma 2.27: Proof (Forward)

If $\boldsymbol{A_{pq}}$ generates $\boldsymbol{x}$, it brings $\boldsymbol{P}$ from $\boldsymbol{p}$ with empty stack to $\boldsymbol{q}$ with empty stack

<u>Basis:</u> The derivation has 1 step

There is only one rule possible $\boldsymbol{A_{pp}} \rightarrow \boldsymbol{\epsilon}$ which trivially brings P from p to p.

Lemma 2.27: Proof (Forward)

<u>Induction:</u>

Assume true for k steps, prove for k+1

Case a): $A_{pq} \Rightarrow aA_{rs}b$

$x = ayb$ and $A_{rs} \overset{*}{\Rightarrow} y$ in k steps with empty stack (induction assumption).

Now, because $A_{pq} \Rightarrow aA_{rs}b$ in G, we have

$$\delta(p, a, \varepsilon) \ni (r, t) \text{ and } \delta(s, b, t) \ni (q, \varepsilon)$$

Therefore, x can bring P from p to q with empty stack.

Lemma 2.27: Proof (Forward)

Induction:

Assume true for k steps, prove for k+1

Case b): $A_{pq} \Rightarrow A_{pr}A_{rq}$

$x = yz$ such that $A_{pr} \overset{*}{\Rightarrow} y$ and $A_{pr} \overset{*}{\Rightarrow} z$ in at most k steps with empty stack.

Therefore, x can bring P from p to q with empty stack.

Lemma 2.27: Proof (Backward)

If $\boldsymbol{x}$ brings $\boldsymbol{P}$ from $\boldsymbol{p}$ with empty stack to $\boldsymbol{q}$ with empty stack, then $\boldsymbol{A_{pq}}$ generates $\boldsymbol{x}$

<u>Basis:</u> The computation has 0 steps

If it has 0 steps, it starts and ends in the same state. P can only read the empty string. The rule $\boldsymbol{A_{pp}} \rightarrow \boldsymbol{\epsilon}$ generates it.

Lemma 2.27: Proof (Backward)

<u>Induction:</u>

Assume true for k steps, prove for k+1

Case a): Stack is not empty in between

The symbol pushed at the beginning is the same popped at the end, we have therefore $A_{pq} \to aA_{rs}b$ in the grammar.

We have $x = ayb$, from induction we have $A_{rs} \overset{*}{\Rightarrow} y$, therefore $A_{pq} \overset{*}{\Rightarrow} ayb$

Lemma 2.27: Proof (Backward)

Induction:

Assume true for k steps, prove for k+1

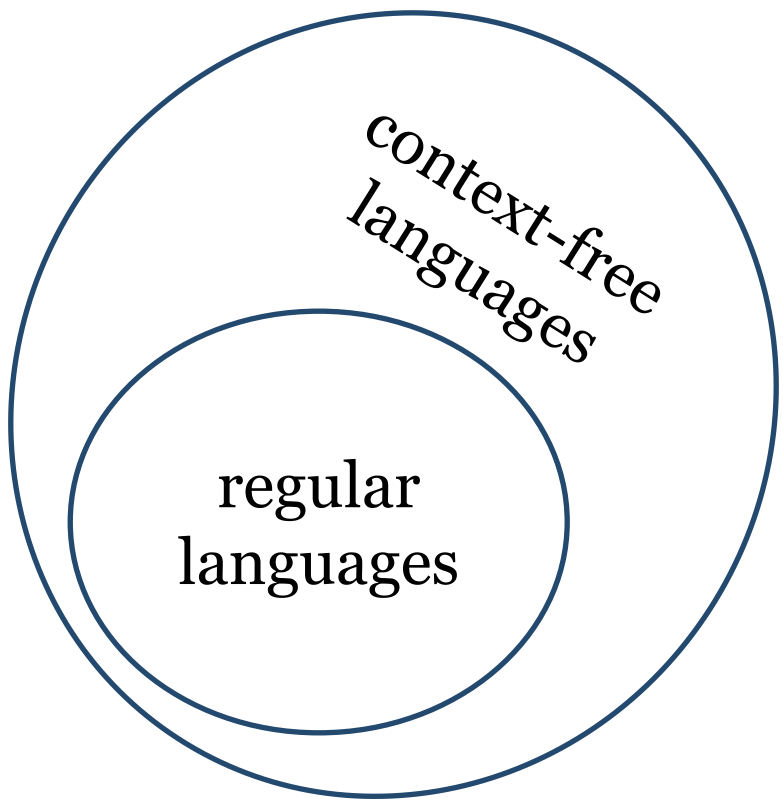
Case b): Stack is empty in between

There exists a state r in between and computations from p to r and r to q have at most k steps. We have $x = yz$, from induction $A_{pr} \overset{*}{\Rightarrow} y$ and $A_{rq} \overset{*}{\Rightarrow} z$.

Since $A_{pq} \rightarrow A_{pr}A_{rq}$ is in the grammar, we have that $A_{pq} \overset{*}{\Rightarrow} yz$

Regular vs. Context Free

- Every regular language is context free
- NFAs are PDAs without a stack!



Pumping Lemma

Theorem Pumping Lemma

If A is a context free language, then there is a number p

such that if s is any string in A of length at least p

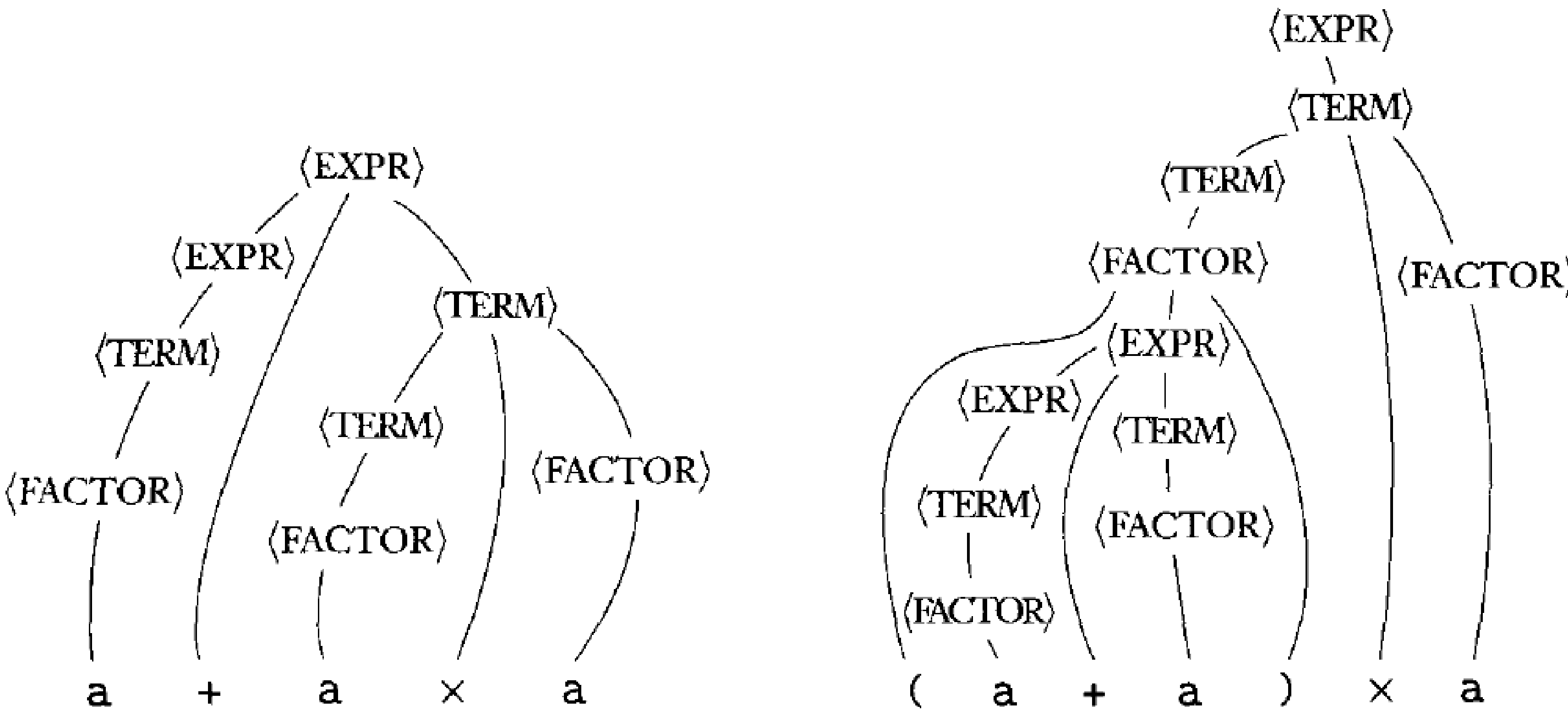
then s may be dived into $s = uvxyz$ such that

1. For each $i \geq 0$; $uv^i xy^i z \in A$

2. $|vy| > 0$

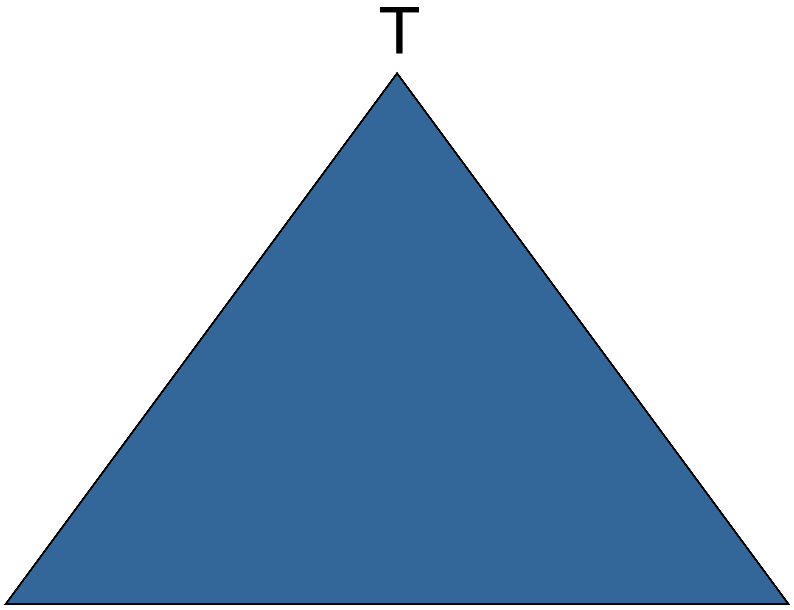
3. $|vxy| \leq p$

Remember the Parse Tree?



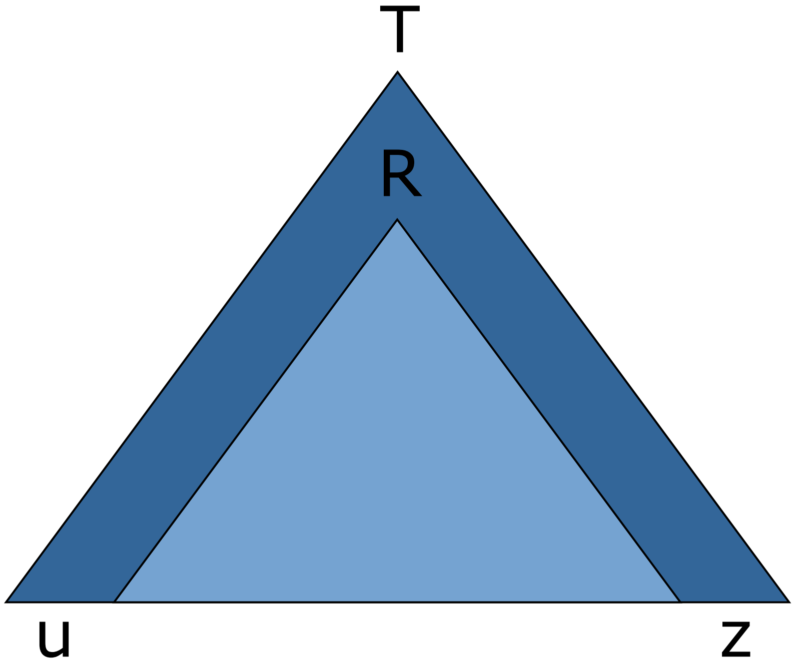
Pumping Lemma: Proof Idea

- Let T be the parse tree for A
- Show that s can be broken into uvxyz
- Prove the conditions holds



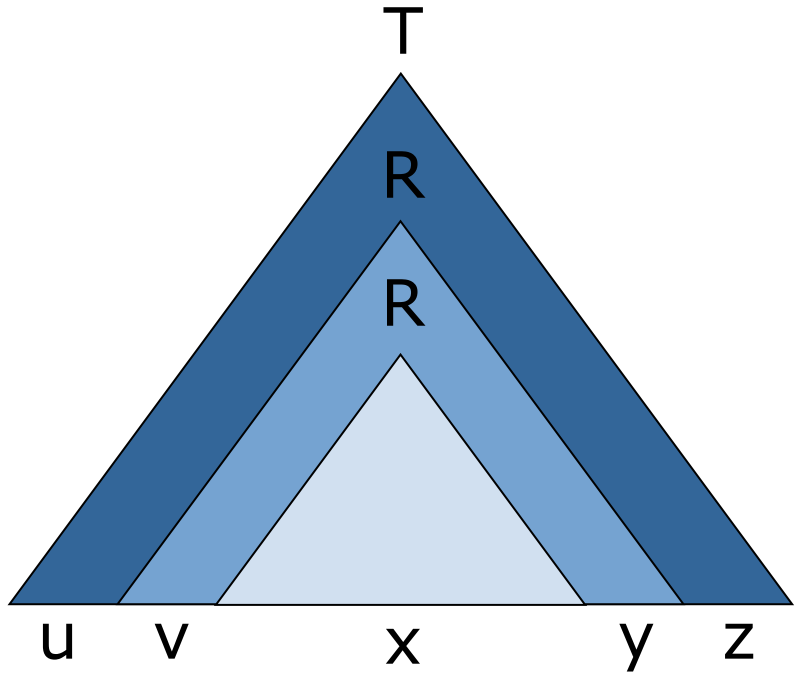
Pumping Lemma: Proof Idea

- Let T be the parse tree for A
- Show that s can be broken into uvxyz
- Prove the conditions holds



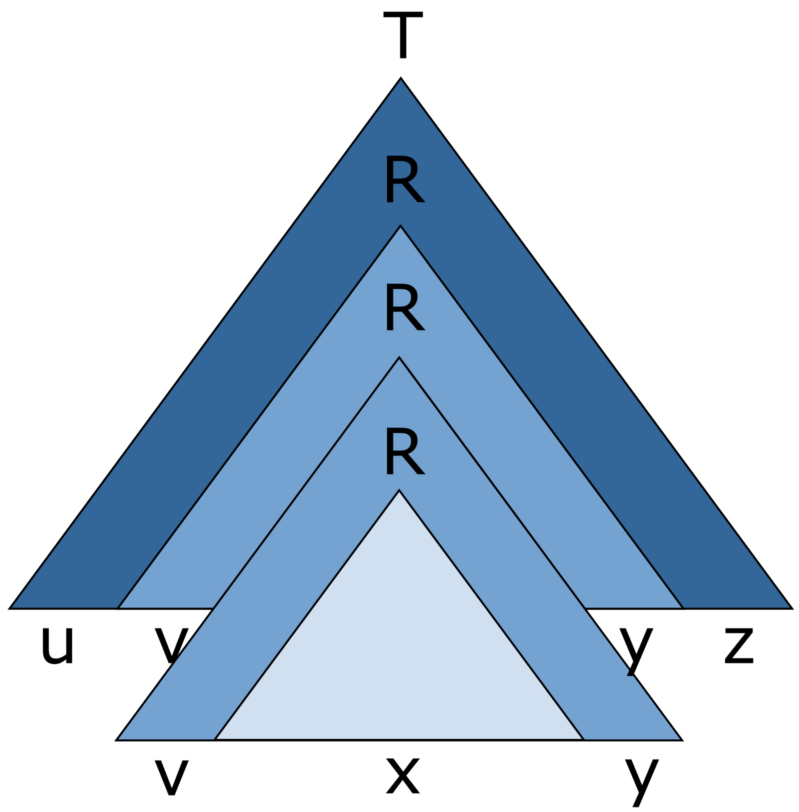
Pumping Lemma: Proof Idea

- Let T be the parse tree for A
- Show that s can be broken into uvxyz
- Prove the conditions holds



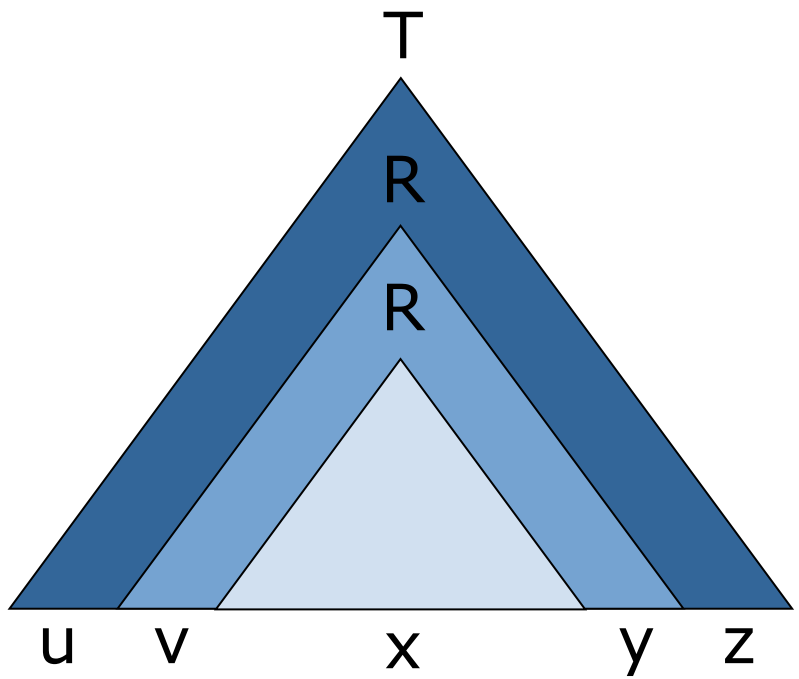
Pumping Lemma: Proof Idea

- Let T be the parse tree for A
- Show that s can be broken into uvxyz
- Prove the conditions holds



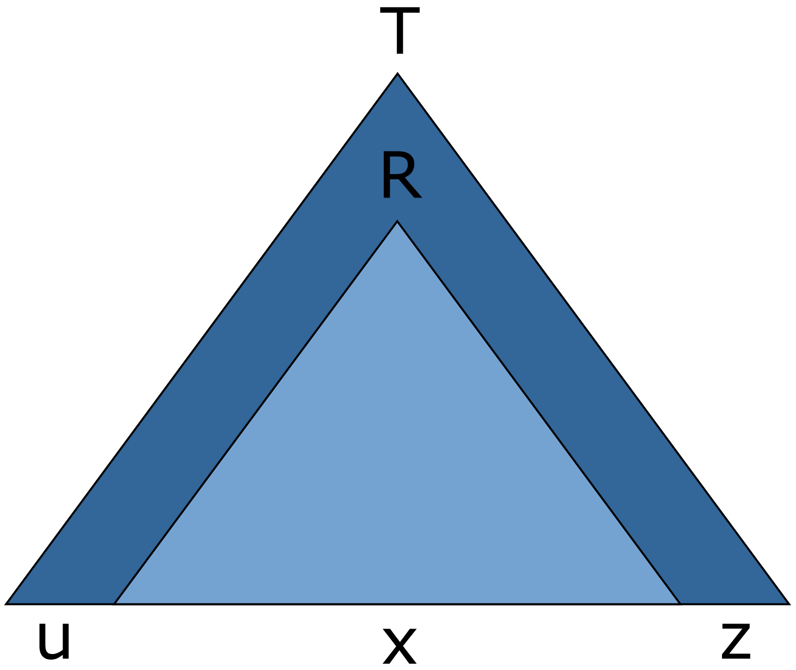
Pumping Lemma: Proof Idea

- Let T be the parse tree for A
- Show that s can be broken into uvxyz
- Prove the conditions holds



Pumping Lemma: Proof Idea

- Let T be the parse tree for A
- Show that s can be broken into uvxyz
- Prove the conditions holds



Pumping Lemma: Proof

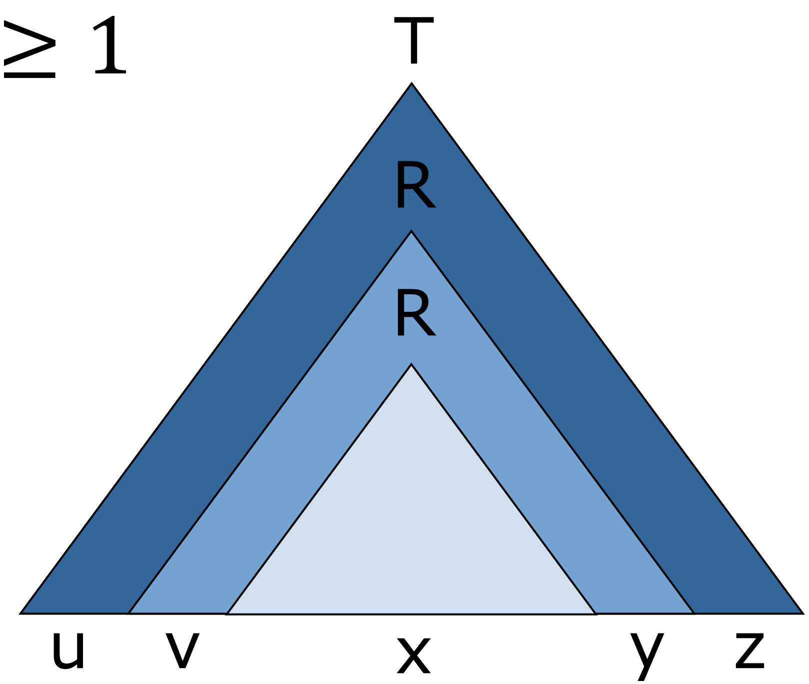
- Let b be the maximum number of symbols on right hand side of a rule
- The number of leaves in a parse tree of height h is at most b^h
- Hence, for any string s of such parse tree, its length $|s| \leq b^h$
- Let $|V|$ be the number of variables and choose the pumping length $p = b^{|V|+2}$

Pumping Lemma: Proof

- For any $|s| \geq p$: possible parse trees for s have height at least $|V| + 1$
- let τ be the minimum parse tree for s
  - It must contain a path P from root to a leaf of length at least $|V| + 1$
  - P has at least $|V| + 2$ nodes: one terminal and the rest variables
  - P has at least $|V| + 1$ variables → some variable must be doubled!

Pumping Lemma: Proof Cnd. 1

- Divide s into $uvxyz$ as in picture.
- R generates vxy, with a large subtree, or just x, with a smaller subtree.
- Pumping down gives uxz; pumping up gives uv^ixy^iz with $i \geq 1$



Pumping Lemma: Proof Cnd. 2

- Condition states $|vy| > 0$.
- We must be sure v and y are not ε.
- Assuming they were ε, substituting smaller for bigger subtree would lead to parse tree with <u>fewer</u> nodes.
- Contradiction: τ chosen to be parse tree with <u>fewest</u> number of nodes

Pumping Lemma: Proof Cnd. 3

- Condition states $|vxy| \leq p$
- Upper occurrence of R generates vxy
- R chosen such that both occurrences fall within the bottom $|V| + 1$ variables on the path and longest path
- Subtree where R generates vxy is at most $|V| + 2$ high.
- A tree of height $|V| + 2$ can generate strings of length at most $b^{|V|+2} = p$

Non Context Free Languages

$$B = \{a^n b^n c^n \mid n \geq 0\}$$

- Choose $a^p b^p c^p$
- Find $uvxyz$, either v or y not empty (2)
- Two cases:
  - Contain only one type of symbol: Impossible to respect the equal number
  - Contain mixed symbols: Impossible to keep the order of symbols

Non Context Free Languages

$$C = \left\{a^i b^j c^k \mid 0 \leq i \leq j \leq k\right\}$$

- Choose $a^p b^p c^p$
- Find $uvxyz$, either v or y not empty (2)
- Two cases as before:
  - Contain only one type of symbol
    More complex to prove (next slide)
  - Contain mixed symbols
    Impossible to keep the order of symbols

Non Context Free Languages

$$C = \left\{a^i b^j c^k \mid 0 \leq i \leq j \leq k\right\}$$

- Contain only one type of symbol
  - a does not appear:
    we have that $uv^0xy^0z \notin C$ (less b and c)
  - b does not appear:
    if a appears, $uv^2xy^2z \notin C$ (more a than b)
    if c appears, $uv^0xy^0z \notin C$ (more c than b)
  - c does not appear:
    we have that $uv^2xy^2z \notin C$ (more a and b)

Example Exam Question

Q: Let $G = \langle \{S\}, \{0,1\}, R, S \rangle$ be the CFG with rules:

$$S \to 0S0 \mid 1S1 \mid 0 \mid 1 \mid \epsilon$$

Specify a CFG G_0 in Chomsky Normal Form such that $L(G_0) = L(G)$.

A: Follow the algorithm:

(a) Introduce an additional start variable and the rule $S_1 \to S$

(b) Remove the ϵ rules:

$$S_1 \to S \mid \epsilon \qquad S \to 0S0 \mid 1S1 \mid 0 \mid 1 \mid 00 \mid 11$$

(c) Remove the unit rules:

$$S_1 \to 0S0 \mid 1S1 \mid 0 \mid 1 \mid 00 \mid 11 \mid \epsilon \quad S \to 0S0 \mid 1S1 \mid 0 \mid 1 \mid 00 \mid 11$$

(d) Remove the long rules:

$$\begin{aligned} S_1 &\to U_0U_{S0} \mid U_1U_{S1} \mid 0 \mid 1 \mid U_0U_0 \mid U_1U_1 \mid \epsilon & S &\to U_0U_{S0} \mid U_1U_{S1} \mid 0 \mid 1 \mid U_0U_0 \mid U_1U_1 \\ U_{S0} &\to SU_0 & U_{S1} &\to SU_1 \\ U_0 &\to 0 & U_1 &\to 1 \end{aligned}$$

Summary

- Context free grammars
- Pushdown Automata
- Equivalence of PDAs and CFGs
- Non-context free grammars
  - Pumping lemma