context-free grammars

cmsc 141

context-free grammar

- “Ang kuneho ay nasa loob ng sombrero”
  - At least syntactically correct
- “Ang sombrero ay nasa loob ng kuneho”
- You can readily tell whether sentences are formed according to generally accepted rules for sentence structure
  - Language Recognizer – a device that accepts valid strings
    - Finite automata
- You are also capable of producing legal Filipino sentences
  - Language generator

context-free grammar

- Recall regular expressions
  - Can be viewed as language generators
- Consider a(a* U b*)b
  - Output an a
  - Then do one of the following two things
    - Output a number of a’s
    - Output a number of b’s
  - Finally, output a b

context-free grammar

- Notice that any string in the language generated by a(a* U b*)b consists of a leading a, followed by a middle part, then by a trailing b
- If we let S be a new symbol interpreted as “a string in the language” and M be a symbol standing for “middle part” then we can express this observation by
  - $S \rightarrow aMb$ – referred to as a rule
  - $M \rightarrow A$ and $M \rightarrow B$ – additional rules where A and B are new symbols that stand for strings of a’s and b’s
    - $A \rightarrow e$, $A \rightarrow aA$
    - $B \rightarrow e$, $B \rightarrow bB$

context-free grammar

- The language denoted by the r.e. in the previous slide can then be defined alternatively
  - Start with the string consisting of the single symbol S
  - Find a symbol in the current string that appears to the of a → in one of the rules
  - Replace every occurrence of this symbol with the string that appears to the right of → in the same rule
  - Repeat this process until so such symbol can be found
- To generate aaab:
  - S => aMb using S → aMb
  - => aAb using M → A
  - => aaAb using A → aA
  - => aaaAb using A → aA
  - => aaab using A → e
- We say S =>* aaab

context-free grammar

- A context-free grammar is a language generator that operates like the one in the previous slides with some such set of rules
- Why context-free?
  - Consider the string aaAb
  - The strings aa and b that surround the symbol A the context of A in the string above
  - The rule A $\rightarrow$ aA says that we can replace A by the string aA no matter what the surrounding strings are (*independently of the context* of A)

context-free grammar

- A context-free grammar is denoted by G = (V, T, P, S) where V is the finite set of symbols called non-terminals, T is a finite set of symbols called terminals, S an element of V called the start symbol and P is the finite set of productions
- Each production is of the form $A \rightarrow \alpha$, where A is a variable and α is a string of symbols from the set of strings formed from the elements of the non-terminals and terminals, i.e. $(V \cup T)^*$
- The language generated by the CFG G is denoted by L(G) and is called a context-free language (CFL)

context-free grammar

- Conventions on CFGs
- Capital letters denote variable (non-terminals)
- S being the start symbol unless otherwise stated
- Small letters and digits are used to represent terminals
- Lowercase Greek letters are used to denote strings of variables and terminals
- Use | (or) to represent alternatives in the productions

context-free grammar

- The grammar for the language composed of strings starting with a and followed by any number of b's and any number of a's ended by a b is given by G = ({S,M,A,B}, {a, b}, P, S) where P = {$S \rightarrow aMb$, $M \rightarrow A|B$, $A \rightarrow aA|e$, $B \rightarrow bB|e$}
- Derivation
  - If $A \rightarrow \beta$ is a production of P in grammar G and α and γ are any strings in $(V \cup T)^*$, then $\alpha A\gamma \Rightarrow \alpha\beta\gamma$ (read as derives)
  - The operator => may be applied one or more steps, i.e.
    - $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n$ where $\alpha_1, \alpha_2, \dots, \alpha_n$ are strings in $(V \cup T)^*$
    - $\alpha_1 \Rightarrow^* \alpha_n$
      - α_1 derives α_n in one or more steps in grammar G

context-free grammar

- Consider the CFG G = ({S}, {a, b}, P, S) where
  - $P = \{S \rightarrow aSb, S \rightarrow e\}$
- A possible derivation is
  - S => aSb => aaSbb => aabb
- What is L(G)? What does this suggest?

context-free grammar

- Let G = (W, T, R, S)
  - W = {S, A, N, V, P}
  - T = {Jim, big, green, cheese, ate}
  - R = {S→PVP, P→N, P→AP, A→big, A→green, N→cheese, N→Jim, V→ate}
- Some strings in L(G)
  - Jim ate cheese
  - big Jim ate green cheese
  - big cheese ate Jim
- Too
  - big cheese ate green green big green big cheese
  - green Jim ate green big Jim

context-free grammar

- Challenge
  - Consider the CFG G = ({S, A}, {a, b}, P, S} and P = { $S \to AA$, $A \to AAA$, $A \to a$, $A \to bA$, $A \to Ab$}
    - Give at least four distinct derivations for the string babbab

context-free grammar

- G = ({S},{(,)}, P, S)
  - P = {S→(S), S→∈}
  - Generate a derivation for the following
    - ((()))
    - ()(())()

context-free grammar

- $G = (\{S\},\{(,)\}, P, S)$
  - $P = \{S \rightarrow SS,\ S \rightarrow (S),\ S \rightarrow e\}$
  - Generate a derivation for the following
    - ((()))
    - ()(())()

context-free grammar

- Construct CFGs generating the following languages
  - accepts any string
  - accepts all strings containing any number of a’s only, including 0 a’s
  - accepts all strings containing the substring aba
  - accepts all strings w such that $w = w^R$

context-free grammar

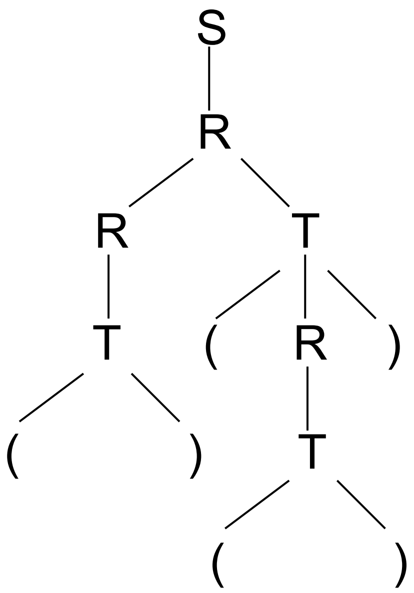
- Challenge
  - Design a CFG that generates all strings or properly balanced left and right parentheses: every left parenthesis can be paired with a unique subsequent right parenthesis, and every right parenthesis can be paired with a unique preceding left parenthesis
  - Design the rules/productions of a grammar that generates the language over the alphabet {x,1,2,+,*,(,)} that represent syntactically correct arithmetic expressions involving + and * over the variables x1 and x2

context-free grammar

- Derivation (Parse) Tree
  - an alternative to showing derivations is the use of derivation trees or parse trees
  - let G = (V, T, P, S) be a CFG. A tree is a derivation or parse tree in G if
    - every vertex has a label which is a symbol of V U T U {∈}
    - the label of the root is S
    - if a vertex is an interior vertex and has label A, then A must be in V
    - if vertex v has label A and vertices $v_1, v_2, \ldots, v_k$ are the children of v, in order from left to right, with labels $x_1, x_2, \ldots, x_k$, respectively, then $A \rightarrow x_1x_2\ldots x_k$ must be a production in P
    - if vertex v has label ∈, then v is a leaf and is the child of its parent

context-free grammar

- Derivation (Parse) Tree
  - G = ({S, R, T},{(,)}, P, S) where P = {S→R, R→RT|T, T→(R)|()}
  - the derivation tree for string ()(())



- ((()))()()
- ()()()(())
- (())(())

context-free grammar

- Derivation (Parse) Tree
  - a **leftmost derivation** is a derivation in which at each step, the leftmost non-terminal is replaced
  - $G = (\{S, A\}, \{a, b\}, P, S)$ where $P = \{ S \to aAS \mid a, A \to SbA \mid SS \mid ba\}$
  - $$\begin{aligned} S &\Rightarrow aAS \\ &\Rightarrow aSbAS \\ &\Rightarrow aabAS \\ &\Rightarrow aabbaS \\ &\Rightarrow aabbaa \end{aligned}$$
  - Draw the parse tree

context-free grammar

- Derivation (Parse) Tree
  - a **rightmost derivation** is a derivation in which at each step, the rightmost non-terminal is replaced
  - $G = (\{S, A\}, \{a, b\}, P, S)$ where $P = \{ S \rightarrow aAS \mid a, A \rightarrow SbA \mid SS \mid ba\}$
  - $S \Rightarrow aAS$
    $\Rightarrow aAa$
    $\Rightarrow aSbAa$
    $\Rightarrow aSbbaa$
    $\Rightarrow aabbaa$
  - Draw the parse tree and compare this with the leftmost derivation

context-free grammar - ambiguity

- A CFG is ambiguous if and only if it generates some sentence by two or more distinct leftmost (rightmost) derivations
- $G = (\{S, T\}, \{a, b\}, P, S)$ where $P = \{S \rightarrow T, T \rightarrow TT \mid ab\}$
    - produce a leftmost derivation for the string ababab
    - draw the parse tree as well

context-free grammar - simplification

- Given a CFL $L \neq \varnothing$, it can be generated by a grammar CFG G with the following properties
  - each variable and each terminal of G appears in the derivation of some string in L, i.e. each symbol is useful
  - there are no productions of the form $A \rightarrow B$ (unit productions), where A and B are variables
  - if the empty string is not in L, then there is no need for the production $A \rightarrow \in$

context-free grammar - simplification

- G = (V, T, P, S) where the productions in P is given by
  - {S→aAS | C, S→a, A→SbA, A→SS, A→ba, B→abc, C→c}
  - Notice that
    - variable B and production B→abc are useless
    - S→C is useless as well, use S→c instead

context-free grammar - chomsky normal form

- Any CFL without ∈ can be generated by a grammar in which all productions are of the form $A \rightarrow BC$ or $A \rightarrow a$, where A, B and C are non-terminals and a a terminal
  - $G = (\{S, A, B\}, \{a, b\}, P, S)$ where $P = \{A \rightarrow bA \mid aB, A \rightarrow bAA \mid aS \mid a, B \rightarrow aBB \mid bS \mid b\}$
    - note that $A \rightarrow a$ and $B \rightarrow b$ are already in CNF
    - what need to be transformed are the following
      - $S \rightarrow bA$ by $S \rightarrow C_bA$ and $C_b \rightarrow b$
      - $S \rightarrow aB$ by $S \rightarrow C_aB$ and $C_a \rightarrow a$
      - $A \rightarrow aS$ by $A \rightarrow C_aS$
      - $B \rightarrow bS$ by $B \rightarrow C_bS$
      - $A \rightarrow bAA$ by $A \rightarrow C_bAA$
      - $B \rightarrow aBB$ by $B \rightarrow C_aBB$
      - Then we transform the non-CNF productions produced in the previous step into CNF
        - $A \rightarrow C_bAA$ by $A \rightarrow C_bD_1$ and $D_1 \rightarrow AA$
        - $B \rightarrow C_aBB$ by $B \rightarrow C_aD_2$ and $D_2 \rightarrow BB$

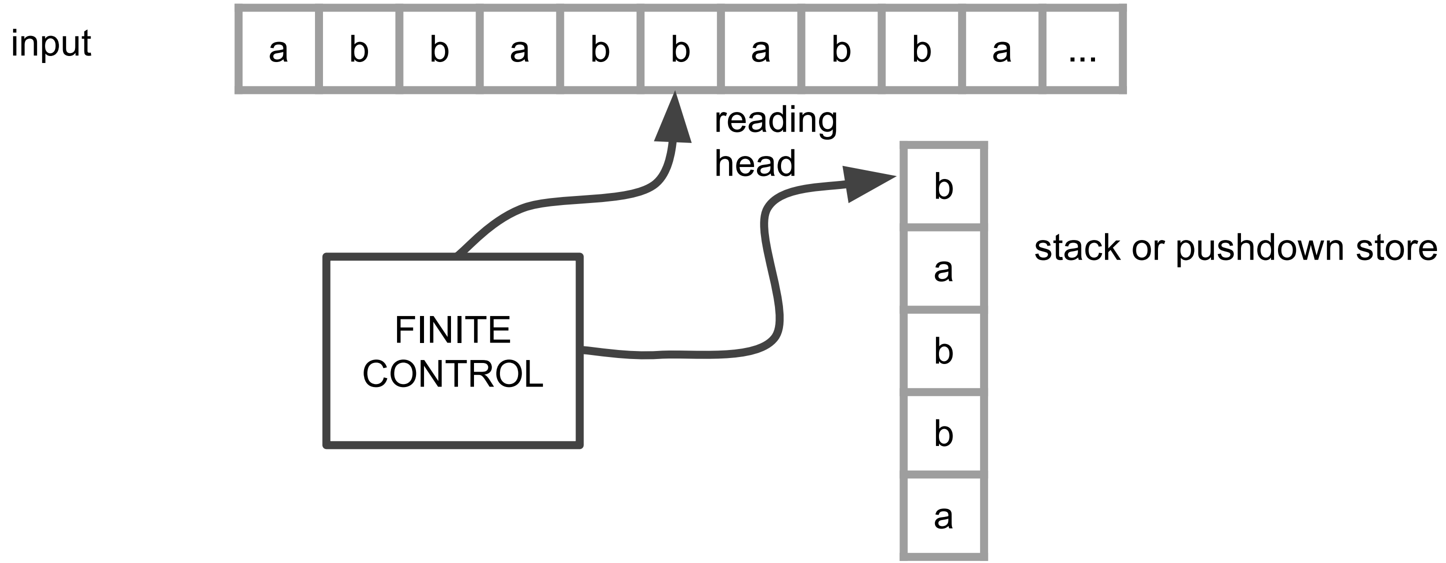
context-free grammar - backus-naur form

- BNF is a grammar developed for the syntactic definition of Algol-60
- BNF grammar is a set of rules or productions of the form
  - leftSide ::= rightSide
    - leftSide is a non-terminal symbol
    - rightSide is a string of non-terminals and terminals
    - a terminal represents the atomic symbols in the language and a non-terminal represents other symbols as defined to the right of the symbol “::=“ (read as “produces” or “is defined as”)
  - | - alternative and {} – possible repetition
    - A::=B | {C}

pushdown automaton (PDA)

- PDA is an automaton equivalent to the CFG in language-defining power.
- models parsers
  - most programming languages have deterministic PDA's
- think of an NFA with the additional power that it can manipulate a stack
  - moves are determined by:
    - current state (of its "NFA")
    - current input symbol (or ε)
    - current symbol on top of its stack.

pushdown automaton (PDA)



pushdown automaton (PDA)

- being non-deterministic, the PDA can have a choice of next moves
- in each choice, the PDA can
  - change state, and also
  - replace the top symbol on the stack by a sequence of zero or more symbols
    - zero symbols
      - pop
    - many symbols
      - sequence of pushes

pushdown automaton (PDA)

- A PDA is a sextuple $M = (K, \Sigma, \Gamma, \Delta, s, F)$ where
  - K is a finite set of states
  - Σ is the alphabet
  - Γ is a stack alphabet
  - s is the start state
  - $F \subseteq K$ is the set of final states
  - Δ is the transition relation, is a finite subset of $(K \times (\Sigma \cup \{e\}) \times \Gamma^*) \times (K \times \Gamma^*)$
    - $((p, a, \beta), (q, \gamma)) \in \Delta$
      - a transition of M
      - since several transitions of M may be simultaneously applicable at any point, the machines we are describing are nondeterministic in operation.

pushdown automaton (PDA)

- to push a symbol is to add it to the top of the stack
- to pop a symbol is to remove it from the top of the stack
  - $((p, u, e), (q, a))$
    - pushes a
  - $((p, u, a), (q, e))$
    - pops a
- as is the case with finite automata, during a computation the portion of the input already read does not affect the subsequent operation of the machine
  - a configuration of a pushdown automaton is defined to be a member of $K \times \Sigma^* \times \Gamma^*$
    - (q, w, abc)

pushdown automaton (PDA)

- if (p, x, α) and (q, y, ζ) are configurations of M
  - (p, x, α) yields in one step (q, y, ζ) ($(p, x, \alpha) \vdash_M (q, y, \zeta)$) if there is a transition $((p, a, \beta), (q, \gamma)) \in \Delta$ such that $x = ay$, $\alpha = \beta\eta$, and $\zeta = \gamma\eta$ for some $\eta \in \Gamma^*$
- M accepts a string $w \in \Sigma^*$ if and only if $(s, w, e) \vdash^*_M (p, e, e)$ for some state $p \in F$
  - $C_0, C_1, \ldots, C_n$ $(n > 0)$ such that $C_0 \vdash_M C_1 \vdash_M \ldots \vdash_M C_n$, $C_0 = (s,w,e)$, and $C_n = (p, e, e)$ for some $p \in F$
  - Any sequence of configurations $C_0, C_1, \ldots, C_n$ such that $C_i \vdash_M C_{i+1}$ for $i = 0, \ldots, n - 1$ will be called a computation by M
    - length n or have n steps
- the language accepted by M, denoted L(M), is the set of all strings accepted by M

pushdown automaton (PDA)

- $L = \{wcw^R : w \in \{a, b\}^*\}$
  - ababcbaba $\in L$
  - abcab $\notin L$
  - cbc $\notin L$
- $M = (K, \Sigma, \Gamma, \Delta, s, F)$
  - $K = \{s, f\}$
  - $\Sigma = \{a, b, c\}$
  - $\Gamma = \{a,b\}$
  - $F = \{f\}$
- Δ
  - ((s, a, e), (s, a))
  - ((s, b, e), (s, b))
  - ((s, c, e), (f, e))
  - ((f, a, a), (f, e))
  - ((f, b, b), (f, e))

pushdown automaton (PDA)

- Δ
  - ((s, a, e), (s, a))
  - ((s, b, e), (s, b))
  - ((s, c, e), (f, e))
  - ((f, a, a), (f, e))
  - ((f, b, b), (f, e))
- abbcbba

state	unread input	stack	transition used
s	abbcbba	e	-
s	bbcbba	a	1
s	bcbba	ba	2
s	cbba	bba	2
f	bba	bba	3
f	ba	ba	5
f	a	a	5
f	e	e	-

pushdown automaton (PDA)

- $L = \{ww^R : w \in \{a, b\}^*\}$
  - ababbaba $\in L$
  - abab $\notin L$
  - cc $\in L$
- $M = (K, \Sigma, \Gamma, \Delta, s, F)$
  - $K = \{s, f\}$
  - $\Sigma = \{a, b, c\}$
  - $\Gamma = \{a,b\}$
  - $F = \{f\}$
- Δ
  - $((s, a, e), (s, a))$
  - $((s, b, e), (s, b))$
  - $((s, c, e), (f, e))$
  - $((f, a, a), (f, e))$
  - $((f, b, b), (f, e))$

pushdown automaton (PDA)

- $L = \{ww^R : w \in \{a, b\}^*\}$
  - $ababbaba \in L$
  - $abab \notin L$
  - $cc \in L$
- $M = (K, \Sigma, \Gamma, \Delta, s, F)$
  - $K = \{s, f\}$
  - $\Sigma = \{a, b, c\}$
  - $\Gamma = \{a,b\}$
  - $F = \{f\}$
- Δ
  - $((s, a, e), (s, a))$
  - $((s, b, e), (s, b))$
  - $((s, e, e), (f, e))$
  - $((f, a, a), (f, e))$
  - $((f, b, b), (f, e))$

pushdown automaton (PDA)

- $L = \{a^n b^n : n \geq 0\}$
  - aaaabbbb $\in$ L
  - abab $\notin$ L
  - aaaabbb $\notin$ L
  - aaabbbb $\notin$ L
- $M = (K, \Sigma, \Gamma, \Delta, s, F)$
  - $K = \{q_0, q_1, q_2, q_3\}$
  - $\Sigma = \{a, b\}$
  - $\Gamma = \{a, b, c\}$
  - $s = q_0$
  - $F = \{q_3\}$
- Δ
  - $((q_0, e, e), (q_1, c))$
  - $((q_1, a, e), (q_1, a))$
  - $((q_1, e, e), (q_2, e))$
  - $((q_2, b, a), (q_2, e))$
  - $((q_2, e, c), (q_3, e))$

CFG and PDA equivalence

- CFG and PDA equivalence
  - The class of languages accepted by pushdown automata is exactly the class of context-free languages.
    - each context-free language is accepted by some pushdown automaton

CFG and PDA equivalence

- CFG and PDA equivalence
  - G = ({S,A,B}, {a,b}, P, S)
  - P = {S->AbB, A->aA|a, B->bbB|e}
  - S => AbB
    => aAbB
    => aaAbB
    => aaaAbB
    => aaaaAbB
    - leftmost derivation

CFG and PDA equivalence

- each context-free language is accepted by some pushdown automaton

CFG and PDA equivalence

- push the starting symbol and go to next state
- for every rule, push the right-hand side
- for every terminal symbol matched, pop the stack
  - for both cases, stay at the same state
  - there might be intermediate states
- check whether we are done

CFG and PDA equivalence

- each context-free language is accepted by some pushdown automaton
  - Let G = (V, T, R, S) be a context-free grammar
  - Construct a machine M = ({p, q}, T, VUT, Δ, p, {q})
  - Δ
    - ((p, e, e), (q, S))
    - ((q, e, A), (q, x)) for each rule A -> x in R
    - ((q,a,a),(q,e)) for each a ∈ T

CFG and PDA equivalence

- G = (V, T, R, S) with V = {S,a,b,c}, T = {a, b, e}, and R = {S->aSa, S->bSb, S->c)
  - M = ({p, q}, T, VUT, Δ, p, { q}) and Δ
    - ((p, e, e), (q, S))
    - ((q, e, S), (q, aSa))
    - ((q, e, S), (q, bSb))
    - ((q, e, S), (q, c))
    - ((q, a, a), (q, e))
    - ((q, b, b), (q, e))
    - ((q, c,c), (q,e))

CFG and PDA equivalence

- M = ({p, q}, T, VUT, Δ, p, { q})
  - ((p, e, e), (q, S))
  - ((q, e, S), (q, aSa))
  - ((q, e, S), (q, bSb))
  - ((q, e, S), (q, c))
  - ((q, a, a), (q, e))
  - ((q, b, b), (q, e))
  - ((q, c,c), (q,e))

state	unread input	stack	transition
p	abbcbba	e	
q	abbcbba	S	
q	abbcbba	aSa	
q	bbcbba	bSba	
q	bcbba	Sba	
q	bcbba	bSbba	
q	cbba	Sbba	
q	cbba	cbba	
q	bba	bba	
q	ba	ba	
q	a	a	
q	e	e	

CFG and PDA equivalence

- The class of languages accepted by pushdown automata is exactly the class of context-free languages.
  - if a language is accepted by a pushdown automaton, it is a context-free language

CFG and PDA equivalence

- PDA has some states
  - start state (q_0)
  - some intermediate states
  - final state/s
- for every pair of states, create a non-terminal
- starting symbol for the grammar will be Aq_0q_f

CFG and PDA equivalence

- simplify the PDA
  - one final state only
  - make the PDA empty the stack
    - new start state
    - new final state
  - the PDA either pushes or pops only but not both
  - the PDA does not do neither
    - use a non-alphabet, non-stack alphabet
    - introduce a new state

CFG and PDA equivalence

- the curious case of using a stack
  - the stack may have been empty or may have contents

CFG and PDA equivalence

- consider states p and q
  - create a nonterminal A_{pq}
    - first transition can be a push but not a pop
    - last transition can be a pop but not a push
    - $A_{pq} \rightarrow aA_{rs}b$
    - first transition pushes a symbol different from the symbol popped in the last transition but still ends up empty
      - what happened?
    - $A_{pq} \rightarrow aA_{pr}A_{rq}$

CFG and PDA equivalence

- for each p, q, r, s $\in$ K such that ((p, a, e), (r, t)) and ((s, b, t), (q, e)), then add the rule $A_{pq} \rightarrow aA_{rs}b$
  - for each p, q, r $\in$ K, such that we could go from p to q without "touching" the stack and from q to r, still without "touching" the stack, then add $A_{pq} \rightarrow A_{pr}A_{rq}$
  - add a rule A_{pp} for transitions that go from p to p (trivial)
    - $A_{pp} \rightarrow e$
  - A_{q0qf} is the start symbol

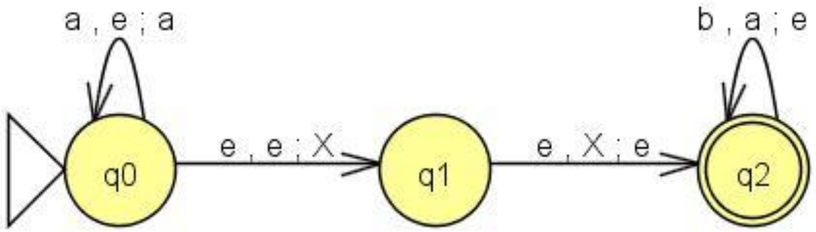
CFG and PDA equivalence (intermission)

- L = {w: w has the same number of a's and b's
  - ((s,e,e),(q,c))
  - ((q,a,c),(q,ac))
  - ((q, a, a), (q, aa))
  - ((q,a,b), (q,e))
  - ((q,b,e),(q,be))
  - ((q,b,b),(q,bb))
  - ((q,b,a),(q,e))
  - ((q,e,c), (f,e))

State	Input	Stack	Transition	Comments
s	abbbabaa	e	_	Initial configuration
q	abbbabaa	e	_	Bottom marker
q	bbbabaa	ac	_	Start a stack of a's
q	bbabaa	e	_	Remove one a
q	babaa	be	_	Start a stack of b's
q	abaa	bbc	_	
q	baa	be	_	
q	aa	bbc	_	
q	a	bc	_	
q	e	c	_	
f	e	e	_	

CFG and PDA equivalence (example)

$A_{00} \rightarrow A_{00}A_{00} \mid \varepsilon$
$A_{01} \rightarrow A_{00}A_{01} \mid A_{01}A_{11}$
$A_{02} \rightarrow A_{00}A_{02} \mid A_{01}A_{12} \mid A_{02}A_{22}$
$A_{11} \rightarrow A_{11}A_{11} \mid \varepsilon$
$A_{12} \rightarrow A_{11}A_{12} \mid A_{12}A_{22}$
$A_{22} \rightarrow A_{22}A_{22} \mid \varepsilon$
$A_{02} \rightarrow aA_{02}b$
$A_{02} \rightarrow A_{11}$



pumping lemma for CFL

- pumping lemma for context-free languages
  - $L = \{dada^n cb^n fafa \; n \geq 0\}$
  - $P = \{S \rightarrow dadRfafa, R \rightarrow aRb \mid c\}$
    - generate strings
      - for any string w, it can be written as uvxyz
        - $uv^ixy^iz \in L$

pumping lemma for CFL

- pumping lemma for context-free languages
  - Let L be a CFL. There is an integer $p \geq 1$ (pumping length) such that any string $s \in L$ with $|s| \geq p$ can be rewritten as $s = uvxyz$ such that $uv^ixy^iz \in L$ for each $i \geq 0$, $|vy| > 0$, and $|vxy| \leq p$

pumping lemma for CFL

- pumping lemma for context-free languages
  - $L = \{a^n b^n c^n, n \geq 0\}$ is not context-free
    - proof by contradiction
    - assume that L is context-free
    - let $s = a^p b^p c^p$, rewrite $s = uvxyz$
      - case 1: v contains same symbols, y contains same symbols
      - case 2: either v or y contains more than one symbol

PDA

- CFGs are extensively used in modeling the syntax of programming languages
  - compilers must embody parsers
    - determine whether a given string is in the language generated by a CFG
      - if so, construct the parse tree
    - takes too much time
      - use PDA
        - nondeterministic
  - can we always make pushdown automata operate deterministically?

PDA

- **deterministic** if for each configuration there is at most one configuration that can succeed it in a computation by M