

Computational (Cognitive) modeling of language

- **Grammars and machines as computational models**
- **Finite state automata (FSA), finite state grammars
regular languages (regular expressions, RE)**
- **Context-free grammars (CFG). context-free
languages (CFL), pushdown automata (PDA)**
- **Beyond CFG (CFL), relationship to Turing Machines**
- **Some processing issues-- sources of complexity
sources of ambiguity**

Neural nets, abstract neural nets (ANN)

- single layer, multi-layer feed forward ANN**
- recurrent ANN**

Alternate characterization of neural nets

- in terms of patterns of input signals (symbols)
as 'events' described by machines and grammars**
- in particular, regular 'events' as characterized by
finite state automata and regular expressions**

Kleene (late 50's)

Computational (Cognitive) Modeling of Language

- sounds**
 - words**
 - sentences (utterances)**
 - discourse (text)**
 - dialog**
 - combined with other modalities**
- etc.**

We will focus on the sentence level

- specifies a level for modeling grammars**
- grammaticality judgments**
- success in computational modeling**

Given a finite alphabet, Σ , the set of strings over Σ will be denoted by Σ^* , including the null string ε

Let $\Sigma = \{a, b\}$. Then Σ^* denotes all strings of a's and b's, including the empty (null) string ε

- A language over Σ is a subset of Σ^*

$$\mathbf{L = \{ a^m b^n \mid m, n \geq 1 \}}$$

= set of all strings of a's and b's such that all a's precede all b's and there is at least one a and one b

- L is a language over $\{a, b\}^*$

- Let $\Sigma = \{ \text{all words of English} \}$
 - Then Σ^* denotes all strings of words of English
 - Only some of these strings are grammatical sentences of English

- Let L_E be the set of all grammatical sentences of English
 - $L_E \leq \Sigma^*$ is a language over $\Sigma = \{ \text{all words of English} \}$

Sentences in L_E :

John likes apples

Apples like John

Two is greater than four

The black cat is on the mat

etc.

Sentences not in L_E :

The the the

John like peanuts

Every student hates any course

The rat the cat the dog chased bit ate the cheese (?)

etc.

A language over Σ can be finite or infinite

L_E : the set of all grammatical sentences of English

L_E is potentially infinite

Finite characterization of a potentially infinite set

- grammar characterization**
- machine characterization**
- behavioral characterization**

We will return to the general characterization of grammars (Sections 16.4 and 16.5) later -- after discussing Chapters 17 and 18.

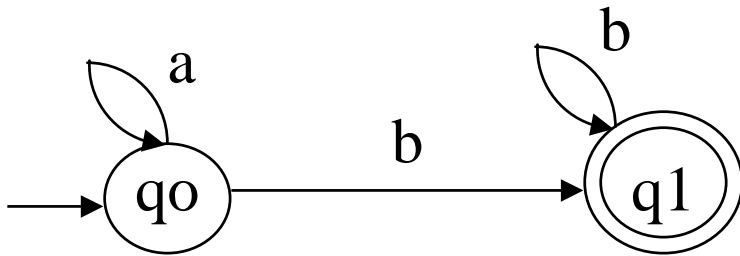
Skip Section 16.3 (formal definitions of trees)

Chapter 17:

- machine characterization of languages
with finite state machines**
- equivalent grammar characterization**
- equivalent 'behavioral' characterization**
 - in terms of terminal symbols only**
 - regular expressions**

Finite State Automata (FSA or FA), Regular Languages and Finite State Grammars (or Type 3 Grammars)

- Characterize languages, i.e. $L \subseteq \Sigma^*$ by FSA**
- FSA (FA) $M = (K, \Sigma, \delta, q_0, F)$ where**
 - K is the finite set of states**
 - Σ is the input alphabet (finite)**
 - $q_0 \in K$, the initial state**
 - $F \subseteq K$, the set of final states**
 - δ is the transition function--next state function**
 - for each state and each input symbol δ specifies the next state of the machine**



$$\Sigma = \{a,b\}$$

L = any number of a's (including none) followed by any number of b's (but at least one b)

$$= \{ a^n b^m \mid n \geq 0, m \geq 1 \}$$

= $a^* b b^*$ (* means any number of repetitions including none)

Given a FSA, M , the language (i.e., the set of strings accepted by M) is defined as follows

$$\begin{aligned} L(M) &= \{ w \mid w \in \Sigma^* \text{ and starting with } q_0 \text{ and following} \\ &\quad \text{the transitions as specified by } \delta, \text{ FSA, } M \\ &\quad \text{reaches one of the final states} \} \\ &= \{ w \mid w \in \Sigma^* \text{ and } \delta'(q_0, w) \in F, \text{ where } \delta' \text{ is the} \\ &\quad \text{same as } \delta \text{ except that it is extended to} \\ &\quad \text{strings in } \Sigma^* \} \end{aligned}$$

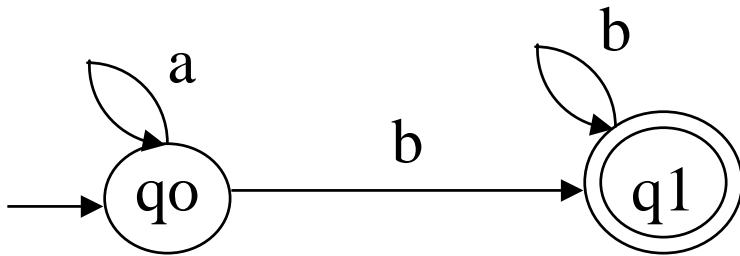
A language L is a finite state language (fsl) or a regular language if there is a FSA, M such that the language of M is L .

Let $\Sigma = \{0, 1\}$

Let $L =$ the set of all strings of 0's and 1's that contain exactly two 1's

Some other examples:

See Exercise 3, Ch. 17



$\Sigma = \{a,b\}$

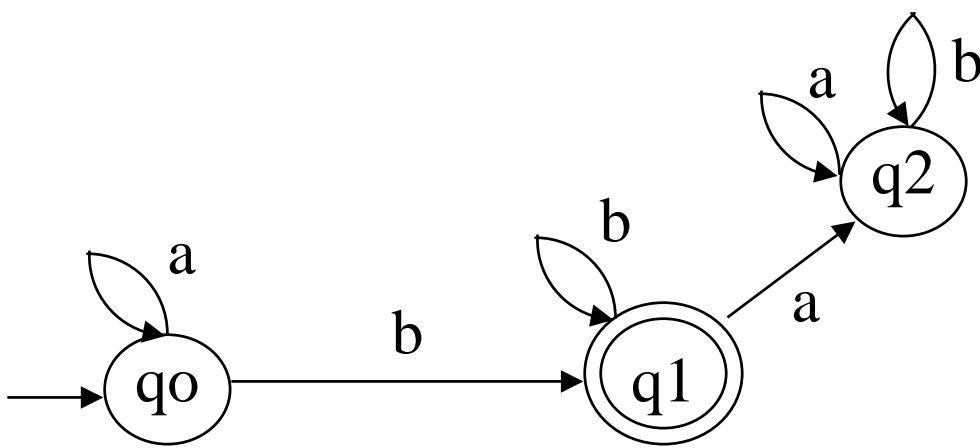
δ transition function

$q_0, a \rightarrow q_0$

$q_0, b \rightarrow q_1$

$q_1, b \rightarrow q_1$

$q_1, a \rightarrow ?$ (dead state)



$\Sigma = \{a,b\}$

δ transition
function

$q_0, a \rightarrow q_0$

$q_0, b \rightarrow q_1$

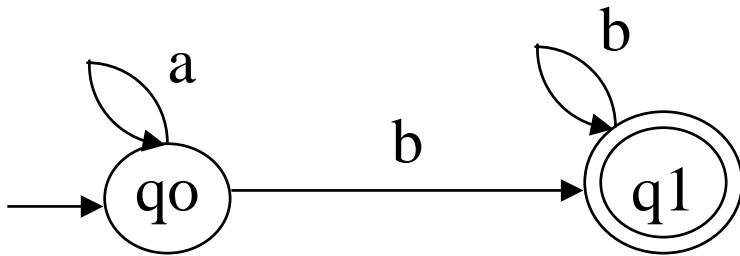
$q_1, b \rightarrow q_1$

$q_1, a \rightarrow q_2$

$q_2, a \rightarrow q_2$

$q_2, b \rightarrow q_2$

**Complete specification
of FSA, i.e., including the
dead states**



$$\Sigma = \{a, b\}$$

δ transition function

$$q_0, a \rightarrow q_0$$

$$q_0, b \rightarrow q_1$$

$$q_1, b \rightarrow q_1$$

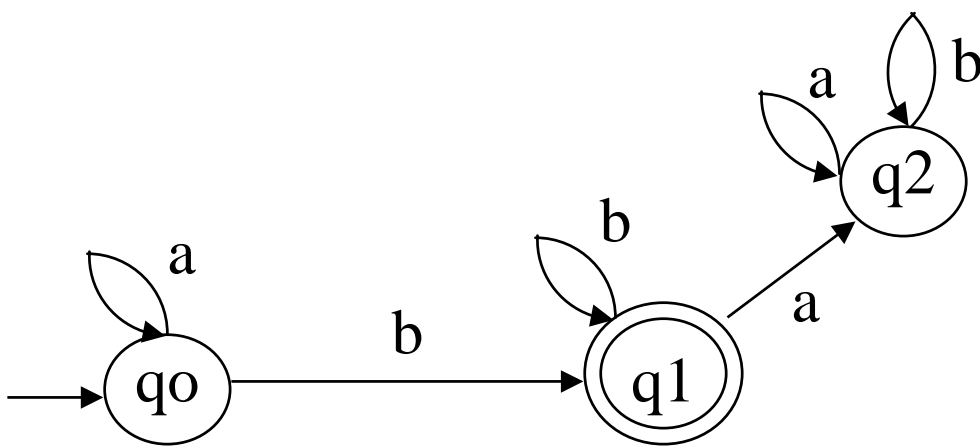
$$q_1, a \rightarrow ? \text{ (dead state) in } L$$

$$L = \{ a^n b^m \mid n \geq 0, m \geq 1 \}$$

Complement of $L =$

$\Sigma^* - L =$ set of strings of
a's and b's not contained

in L



$\Sigma = \{a,b\}$

δ transition
function

$q_0, a \rightarrow q_0$

$q_0, b \rightarrow q_1$

$q_1, b \rightarrow q_1$

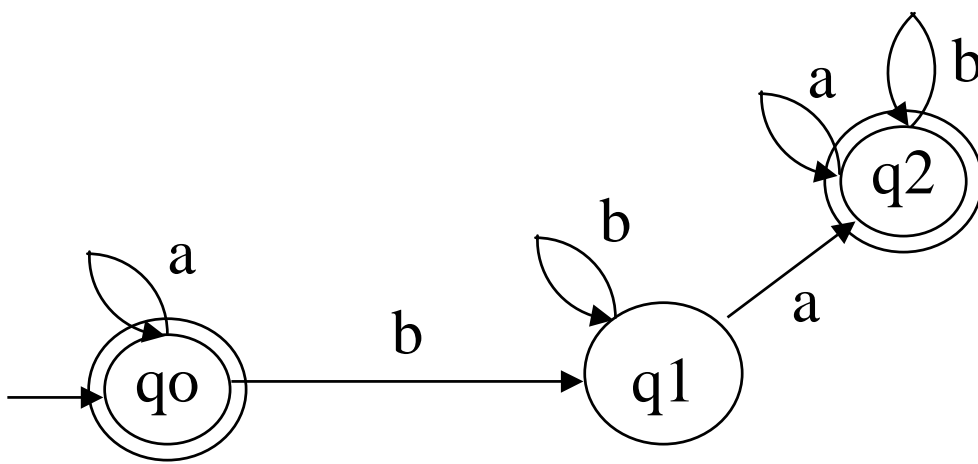
$q_1, a \rightarrow q_2$

$q_2, a \rightarrow q_2$

$q_2, b \rightarrow q_2$

**FSA for the complement
of L**

**Switch the final and non-
final states**



$\Sigma = \{a,b\}$

δ transition
function

$q_0, a \rightarrow q_0$

$q_0, b \rightarrow q_1$

$q_1, b \rightarrow q_1$

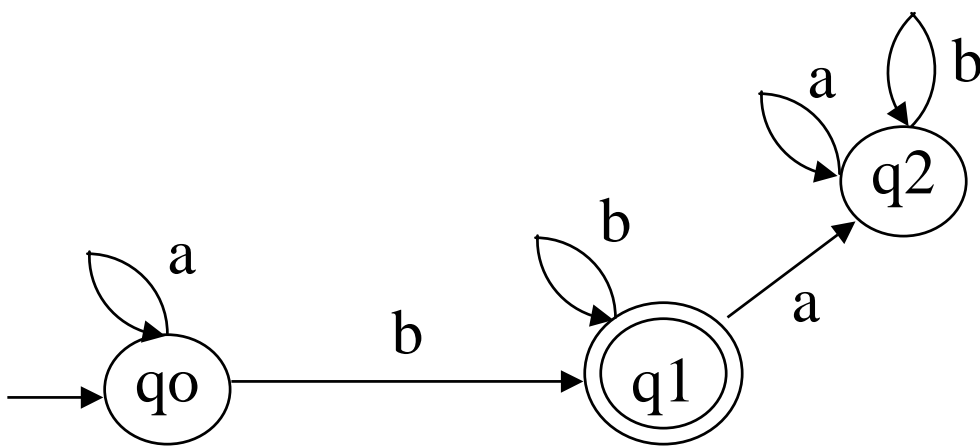
$q_1, a \rightarrow q_2$

$q_2, a \rightarrow q_2$

$q_2, b \rightarrow q_2$

**FSA for the complement
of L**

**Switch the final and non-
final states**



$\Sigma = \{a,b\}$

δ transition
function

$q_0, a \rightarrow q_0$

$q_0, b \rightarrow q_1$

$q_1, b \rightarrow q_1$

$q_1, a \rightarrow q_2$

$q_2, a \rightarrow q_2$

$q_2, b \rightarrow q_2$

For each state and each
input there is exactly one
new state-- δ is a
transition function

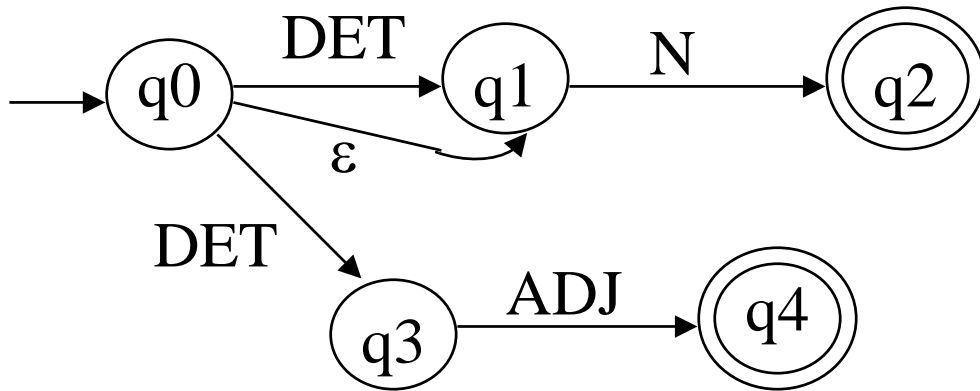
FSA is *deterministic*

Non-deterministic finite automaton

For each state and each input there can be more than one new states

We will also allow transitions on no input (i.e., on the null string)

**How do we define acceptance in a non-deterministic FSA?
-- a string w is accepted if there is at least one state sequence (starting with the initial state) that will reach one of the final states**

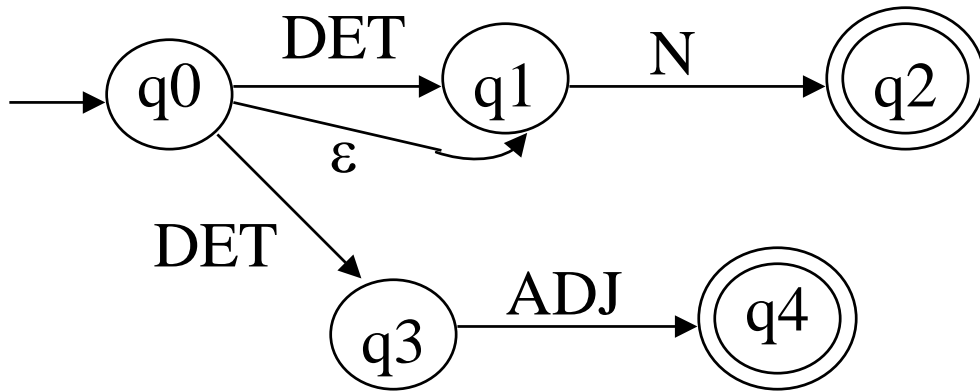


Simple noun phrases of English containing

-- a determiner (DET) followed by a noun (N)-- *the cat*

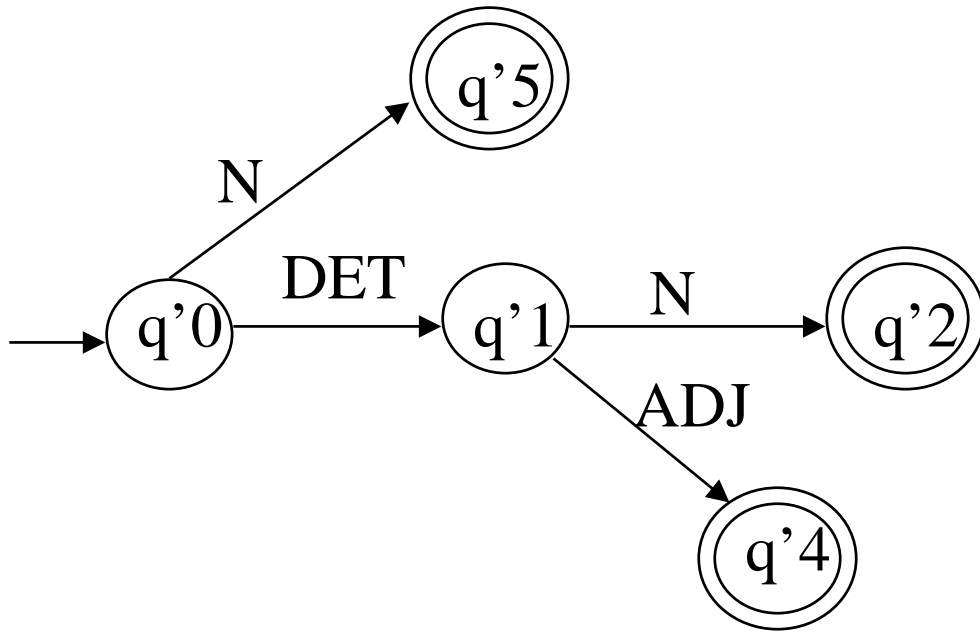
-- DET followed by an adjective (ADJ) -- *the poor*

-- N only -- *peanuts*



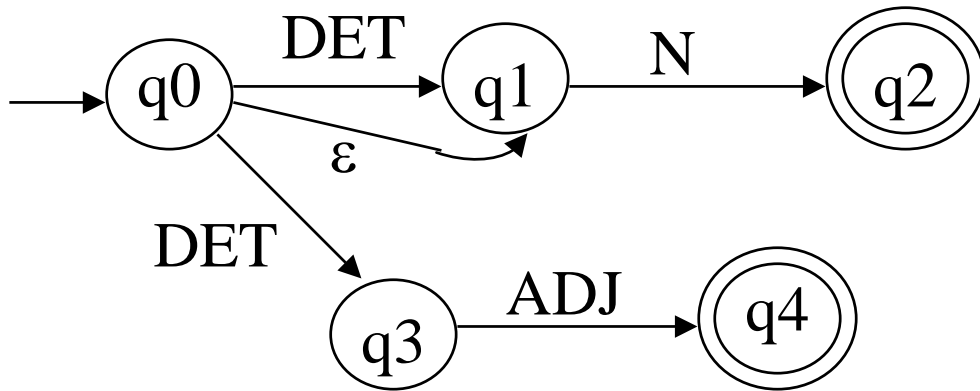
This FSA, M is non-deterministic.

Find FSA M' such that M' is deterministic and $L(M') = L(M)$, i.e., M' accepts exactly the strings accepted by M , M' 's is said to be equivalent to M .



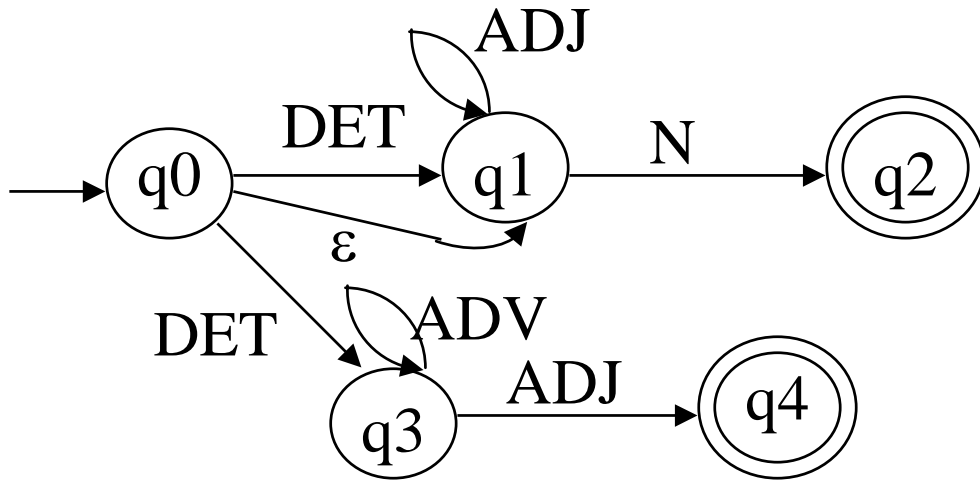
In general--subset construction

States of M' are subsets of states of M --Section 17.1.5



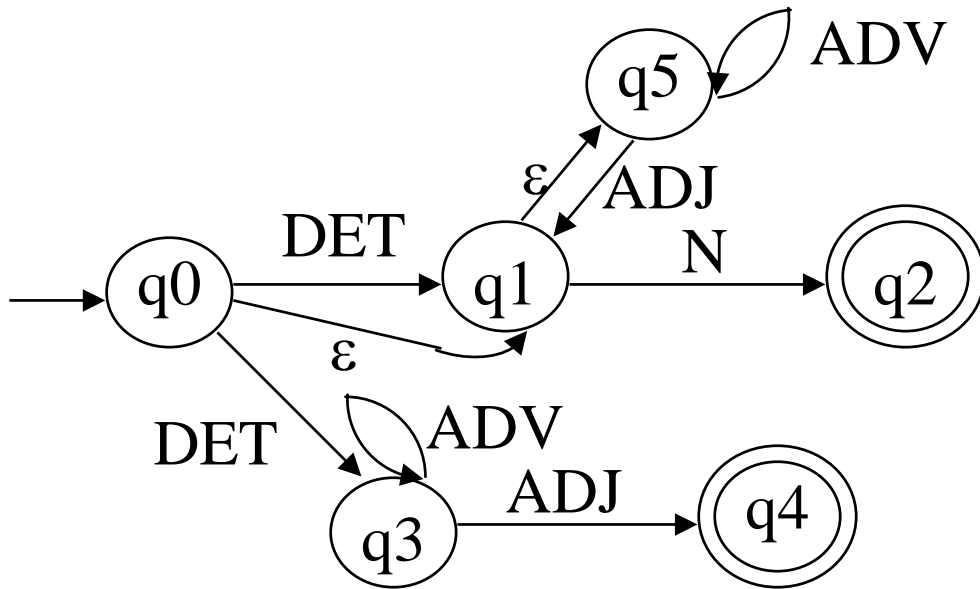
More about noun phrases

- optional adjectives on noun -- *the black cat, the beautiful black cat*
- optional adverbs (ADV) on adjectives-- *the very old cat*



More about noun phrases

-- optional adjectives on noun -- *the black cat, the beautiful black cat*

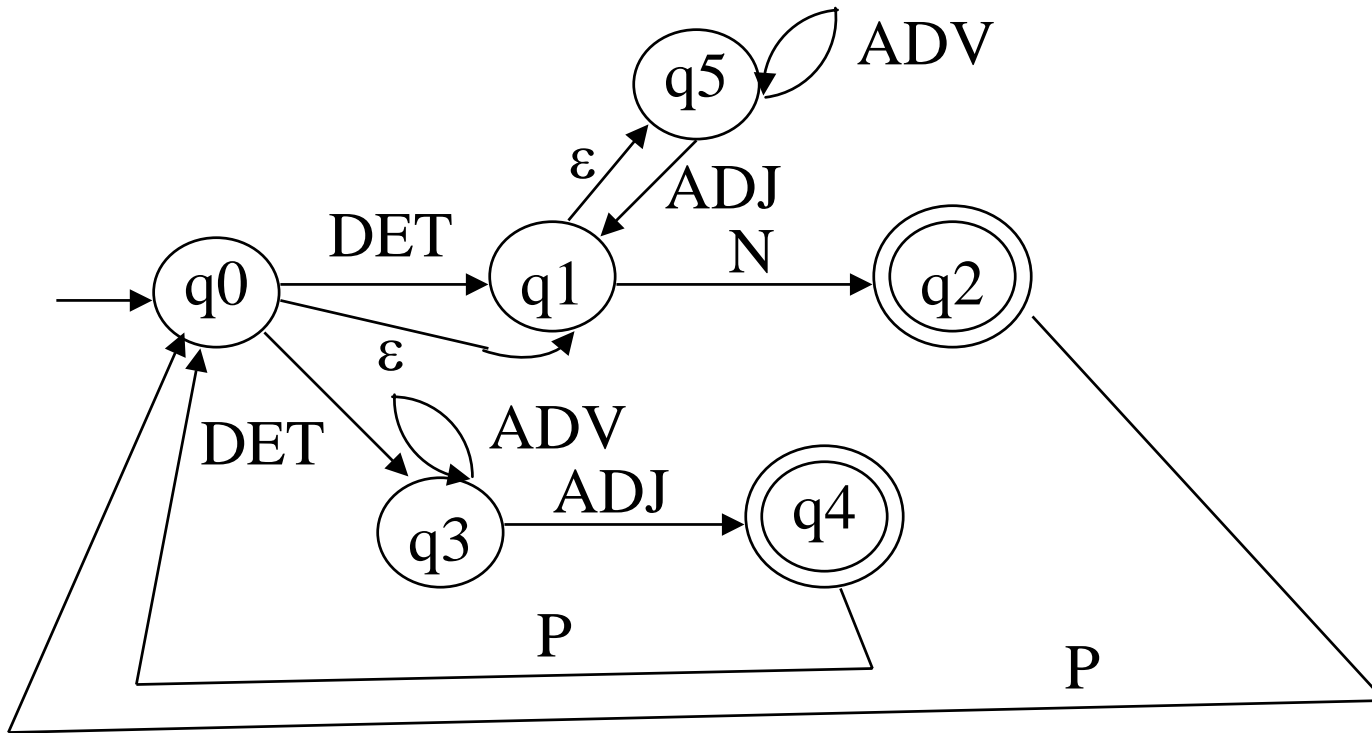


More about noun phrases

- optional adjectives on noun -- *the black cat, the beautiful black cat*
- optional adverbs (ADV) on adjectives-- *the very old cat*

More complications

- nouns modify nouns -- *jet engines, Navy jet engines*
- gerunds, past passive participles modify nouns--
crying children, published papers
- prepositional phrases (PP) consisting of a
preposition (P) followed by a noun phrase modify
nouns-- *the black cat on the dirty old mat*
etc.



Adding prepositional phrases (PP = P NP)
the very beautiful cat on the dirty old mat

Possible trouble:

the cat on the dirty old mat from Wal-Mart

What about simple sentences such as
The beautiful stripy cat on the dirty old mat
yesterday scratched the very old dog
(Note: *yesterday* modifies *scratched*)

Looks like we can do a lot with FSA's suggesting that
'English' may be a finite state language (fsl) (regular
language)

More serious trouble:

the dog chased the cat

the cat the dog chased bit the mouse

the mouse the cat the dog chased bit ate the cheese

etc.

Are there languages that are not fsl (non-regular)?

$$\mathbf{L = \{a^n b^n \mid n \geq 1\}}$$

**L contains strings with equal number of a's and b's,
a's preceding b's**

L is not a fsl.

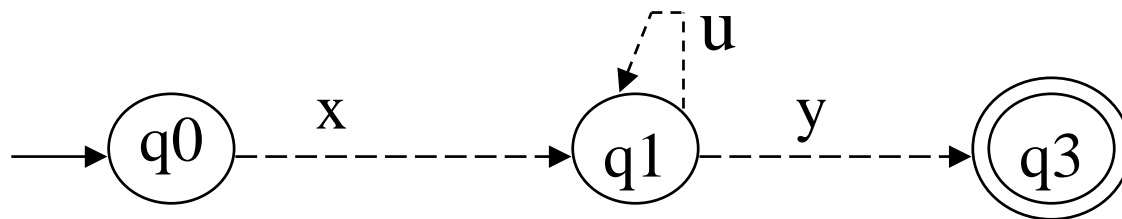
FSA's cannot *count*, i.e., up to an arbitrary number!

Pumping lemma for fsl's

-- a bit formal way of showing why a language is not an fsl--Section 17.2.1

If L is fsl (regular) then for all sufficiently long strings in L we have the following the property.

-- $w \in L$ can be segmented into three parts, say, x, u, y , i.e., $w = x u y$ and all strings of the form $x u^i y \in L$, where u^i means i iterations of u , $i \geq 0$



The loop involving u may include several states.

If L is fsl (regular) then for all sufficiently long strings in L we have the following the property.

-- $w \in L$ can be segmented into three parts, say, x, u, y , i.e., $w = x u y$ and all strings of the form $x u^i y \in L$, where u^i means i iterations of u , $i \geq 0$

$$L = \{a^n b^n \mid n \geq 1\}$$

Try locating the u segment in various places in the string $a a \dots b b \dots$

**In each case the string obtained by iterating u is not in L
Hence, L is not a fsl.**

**There are two other ways of characterizing fsl's
or regular languages**

--finite state grammars --Section 17.3

-- regular expressions -- Section 17.2

Finite state grammars (type 3 grammars, right linear)

A grammar $G = (V_T, V_N, S, R)$ consists of

V_T = terminal vocabulary (lexical items, words)

V_N = non-terminal vocabulary (phrases)

S = start symbol, $S \in V_N$

R = productions, a finite set of rewrite rules

The rewrite rules are of the following form

$A \rightarrow a B$ where $A, B \in V_N$ and $a \in V_T$

$A \rightarrow a$

$$G = (V_T, V_N, S, R)$$

The rewrite rules are of the following form

$A \rightarrow a B$ where $A, B \in V_N$ and $a \in V_T$

$A \rightarrow a$

Derivation starts with S. Since the right hand side of a rule has at most one non-terminal there is only one non-terminal (if any) that can be rewritten at each step.

Derivation stops when there no more non-terminals to be rewritten

$L(G)$ = language derived by G = set of all strings of terminal strings derived in G starting from S

$G = (V_T, V_N, S, R)$

$V_T = \{\text{John, roasted, peanuts, likes}\}$

$V_N = \{S, A, B, C\}$

R:

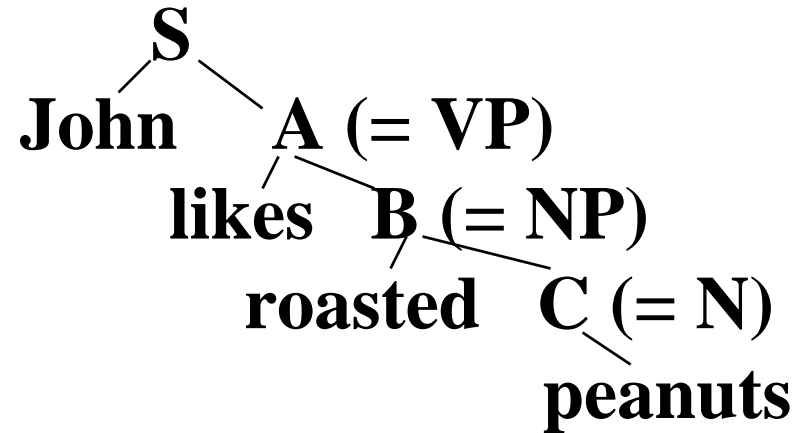
$S \rightarrow \text{John A}$

$A \rightarrow \text{likes B}$

$B \rightarrow \text{roasted C}$

$C \rightarrow \text{peanuts}$

Derivation:



$$G = (V_T, V_N, S, R)$$

The rewrite rules are of the following form

$A \rightarrow a B$ where $A, B \in V_N$ and $a \in V_T$

$A \rightarrow a$

We can associate a FSA, M with G .

-- Treat the non-terminals as states of M

-- The rule $A \rightarrow a B$ corresponds to a transition from state A to state B on the input symbol a

-- The rule $A \rightarrow a$ corresponds to a transition from state A to a final state of M on the input symbol a

FSG's and FSA's are equivalent, i.e., for every FSG, G there is an FSA, M such that the language generated by G is exactly the language accepted by M and vice versa.

‘Behavioral’ characterization of finite state languages

- Regular expressions and finite state (regular) languages

Given Σ , say, $\{a, b\}$

- a is a regular expression (RE) denoting the language $\{a\}$

- b is a RE denoting the language $\{b\}$

- ϵ is a RE denoting the language $\{\epsilon\}$, the language containing the null string

- If A and B are RE’s then so are

$A \cup B$ denoting $\{A\} \cup \{B\}$

AB denoting $\{A\}.\{B\}$

A^* denoting $\{\epsilon\} \cup \{A\} \cup \{A\}.\{A\} \cup \{A\}.\{A\}.\{A\} \dots$

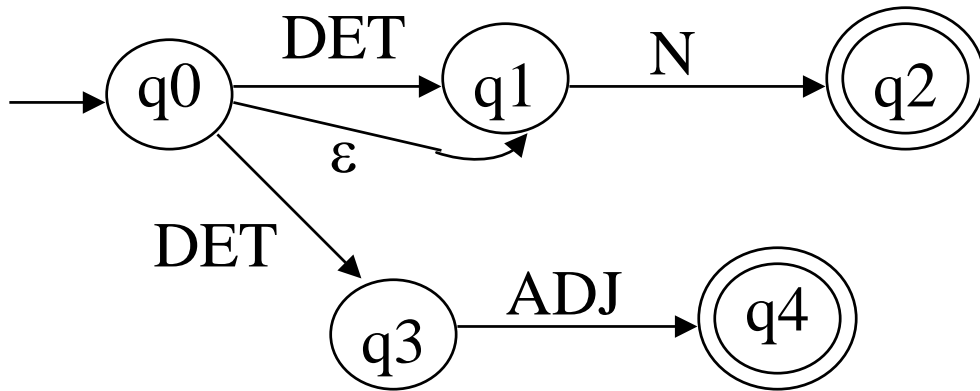
- ϕ is a RE denoting the empty set (language containing no strings)

- nothing else

- **Every RE denotes a finite state language**
- **For every finite state language there is a RE which denotes it**

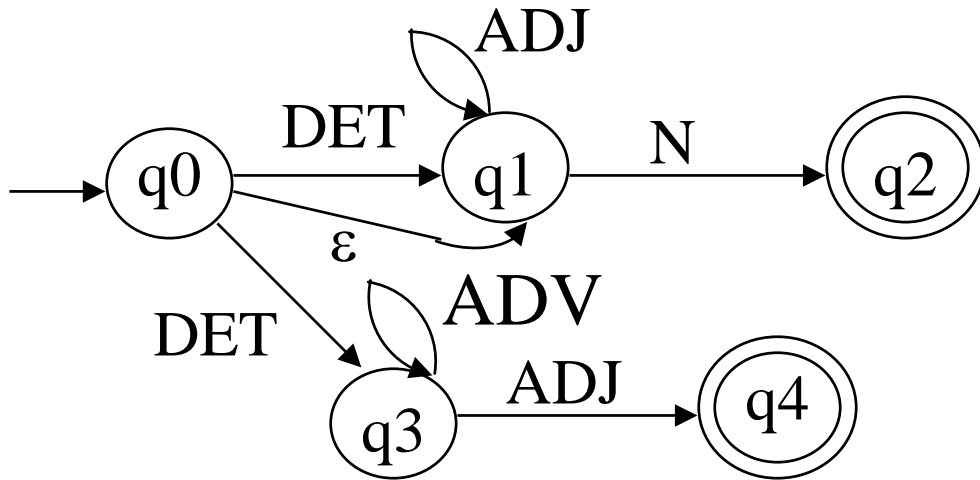
So now we have the following wonderful result:

Finite state machines, finite state grammars (type 3 grammars), and regular expressions all characterize exactly the same set of languages



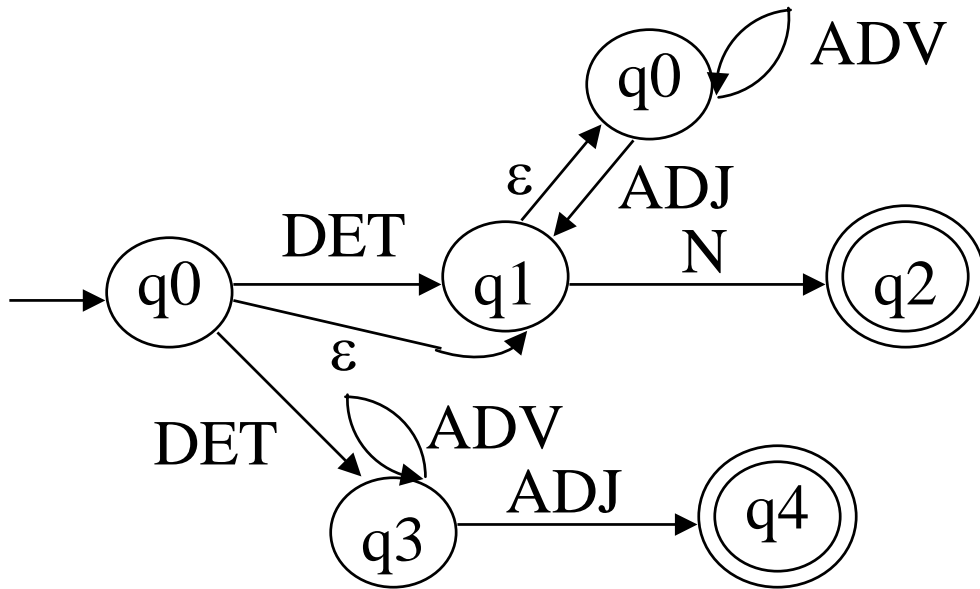
RE denoting the language accepted by the finite state machine shown above

$$\mathbf{DET (N \cup ADJ) \cup N}$$



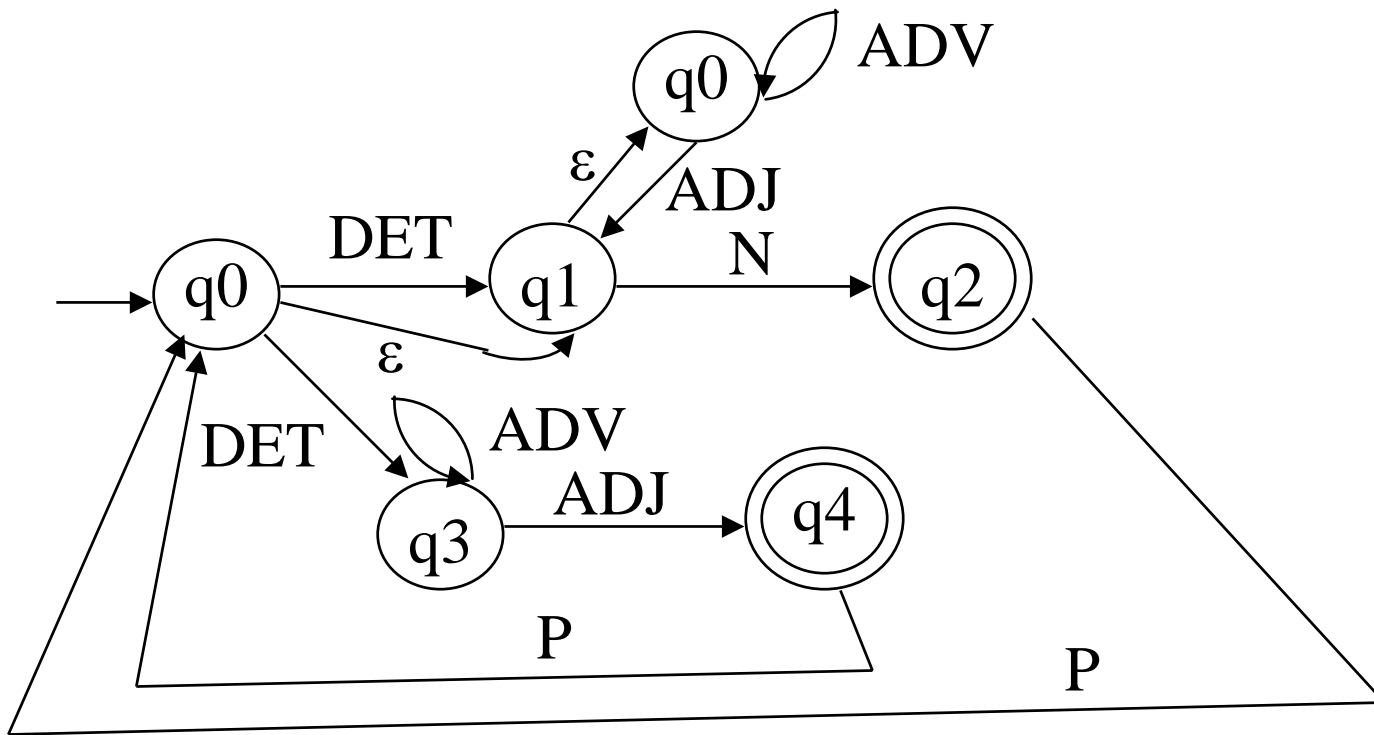
Corresponding RE:

$$((\text{DET} \cup \varepsilon) \text{ADJ}^* \text{N}) \cup (\text{DET} \text{ADV}^* \text{ADJ})$$



RE:

$$((\mathbf{DET} \cup \boldsymbol{\varepsilon}) ((\mathbf{ADV}^* \mathbf{ADJ})^* \mathbf{N}) \cup (\mathbf{DET} \mathbf{ADV}^* \mathbf{ADJ}))$$



RE ?

Why finite state machines or finite state grammars may be inadequate to describe languages?

The rat the cat the dog chased bit ate the cheese

The rat *ate the cheese*
the cat *bit*
the dog chased

These strings have the form

$$L = \{a^n b^n \mid n \geq 1\}$$

**L contains strings with equal number of a's and b's,
a's preceding b's**

-- this language is not a finite state language

Iterations are fine

Mary left

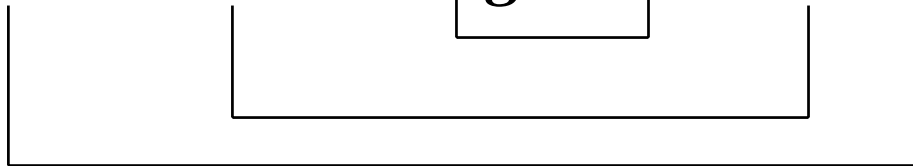
John knows Mary left

Bill thinks John knows Mary left

Harry believes Bill thinks John knows Mary left

*The cheese was eaten by the rat bit by the cat chased
by the dog*

The rat the cat the dog chased bit ate the cheese



**Center embeddings lead to nested dependencies, which
FSA's are unable to handle**

Extending finite state grammars and finite state machines

Finite state grammars

A grammar $G = (V_T, V_N, S, R)$ consists of

V_T = terminal vocabulary (lexical items, words)

V_N = non-terminal vocabulary (phrases)

S = start symbol, $S \in V_N$

R = productions, a finite set of rewrite rules

The rewrite rules are of the following form

$A \rightarrow a B$ where $A, B \in V_N$ and $a \in V_T$

$A \rightarrow a$

Context-free grammars (type 2 grammars)

A grammar $G = (V_T, V_N, S, R)$ consists of

V_T = terminal vocabulary (lexical items, words)

V_N = non-terminal vocabulary (phrases)

S = start symbol, $S \in V_N$

R = productions, a finite set of rewrite rules

The rewrite rules are of the following form

**$A \rightarrow w$ where w is a string of terminal and
non-terminal symbols, i.e., $w \in (V_T \cup V_N)$**

Example:

$G = (V_T, V_N, S, R)$

$V_T = \{\text{the, cat, mat, scratched}\}$

$V_N = \{S, NP, DET, N, VP, V\}$

$S = \text{start symbol, } S \in V_N$

$R = \text{productions, a finite set of rewrite rules}$

$S \rightarrow NP VP$

$V \rightarrow \text{scratched}$

$VP \rightarrow V NP$

$N \rightarrow \text{cat}$

$NP \rightarrow DET$

$N \rightarrow \text{mat}$

N

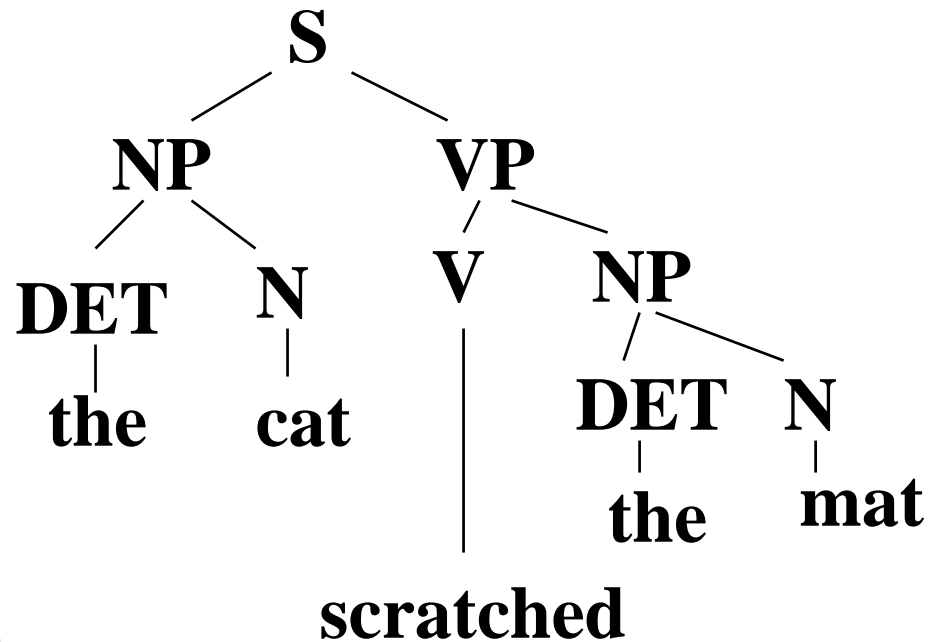
$DET \rightarrow \text{the}$

A derivation

S → **NP VP**
VP → **V NP**
NP → **DET**
N

V → **scratched**
N → **cat**
N → **mat**
DET → **the**

S
NP VP
DET N VP
DET cat VP
DET cat V NP
the cat V DET NP
the cat scratched DET N
the cat scratched DET mat
the cat scratched the mat



Context-free grammars can handle nested dependencies

$$L = \{a^n b^n \mid n \geq 1\}$$

**L contains strings with equal number of a's and b's,
a's preceding b's**

$$G = (V_T, V_N, S, R)$$

$$V_T = \{a, b\}$$

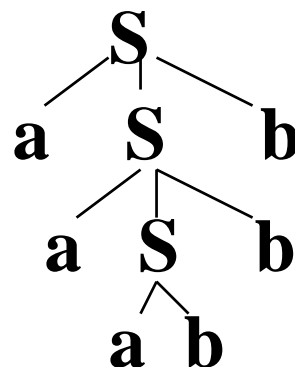
$$V_N = \{S\}$$

S = start symbol, $S \in V_N$

R = productions, a finite set of rewrite rules

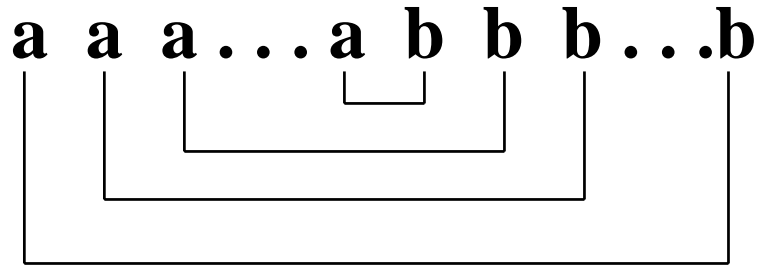
$$S \rightarrow a S b$$

$$S \rightarrow ab$$



a a a b b b

Context-free grammars are adequate to describe nested dependencies



A language L is a context-free language if there is a context-free grammar, G , such that the language of the grammar, $L(G) = L$

-- Finite state grammars are special cases of context-free grammars. Hence,

-- finite state languages \checkmark context-free languages

Are context-free grammars adequate to describe natural languages?

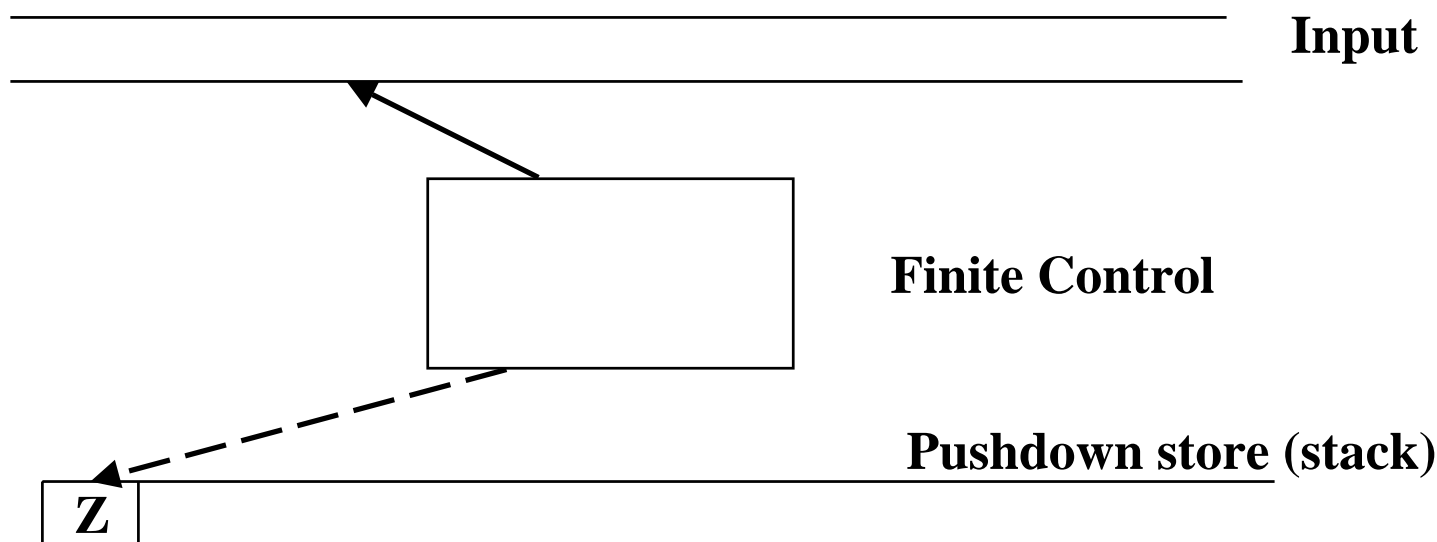
Finite state machines are equivalent to finite state grammars, i.e., for every finite state language there is finite state machine that accepts exactly the same language and vice versa.

Are there machines that are equivalent to context-free grammars, i.e., for every context-free language does there exist a machine that accepts exactly the same language and vice versa?

YES!

Pushdown Automata (PDA)

Pushdown Automaton (PDA)



Move: input symbol, state \rightarrow state, { push/pop }, { move right/stay on the input }

Stack head is on the topmost symbol of the stack.

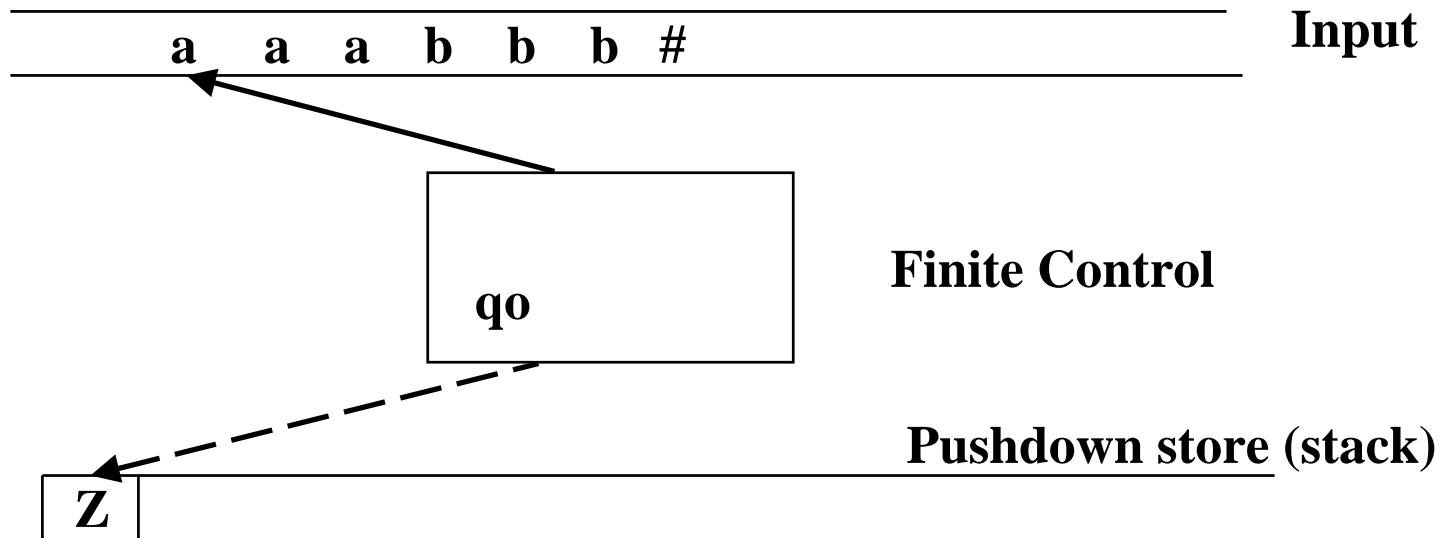
$M = (K, \Sigma, \Gamma, \delta, q_0, F)$ $K, \Sigma, \delta, q_0, F$ as before, Γ is the pushdown alphabet

Z is a special symbol in Γ , which marks the bottom of the stack

A special symbol, $\#$, will mark the end of the input, so the input is $\Sigma \cup \{ \# \}$

Pushdown Automaton (PDA)

Example: Let $\Sigma = \{a, b, \#$, $K = \{q_0, q_1, q_2, q_3\}$, $\Gamma = (A)$, $F = \{q_3\}$
 $L = \{a^n b^n \mid n \geq 1\}$

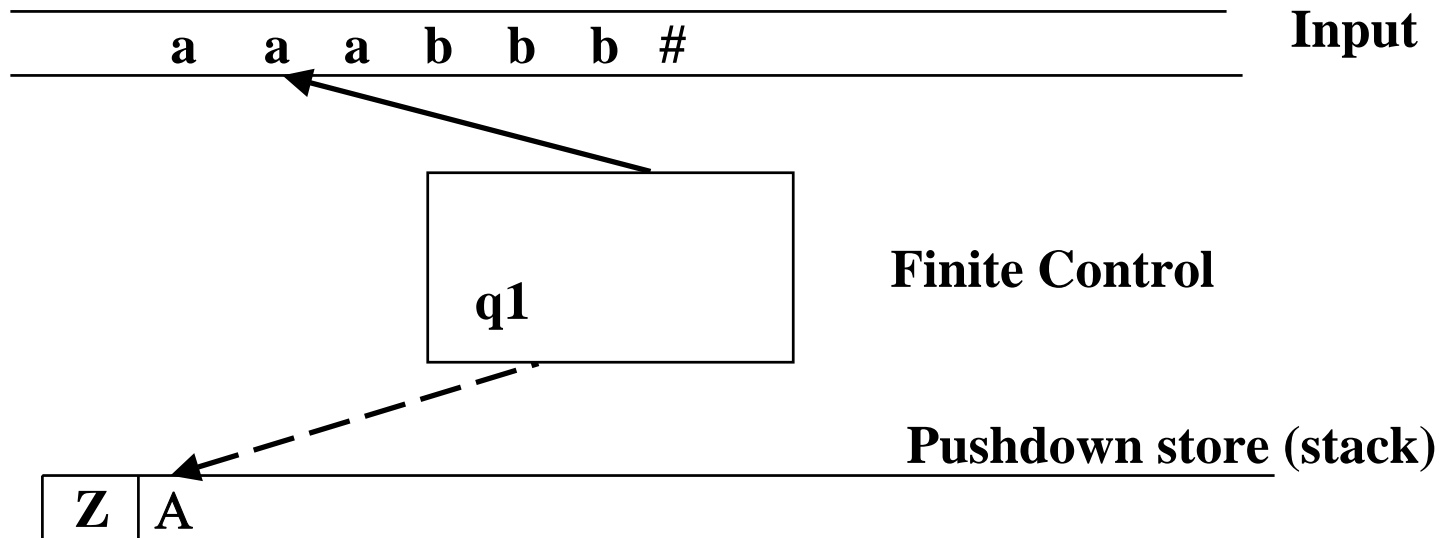


$\delta :$

- $(q_0, a, Z) \rightarrow (q_1, \text{push } A, \text{move right on input})$ ←
- $(q_1, a, A) \rightarrow (q_1, \text{push } A, \text{move right on input})$
- $(q_1, b, A) \rightarrow (q_2, \text{pop } A, \text{move right on input})$
- $(q_2, b, A) \rightarrow (q_2, \text{pop } A, \text{move right on input})$
- $(q_2, \#, Z) \rightarrow (q_3, \text{push } \varepsilon, \text{stay})$

Pushdown Automaton (PDA)

Example: Let $\Sigma = \{a, b, \#\}$, $K = \{q_0, q_1, q_2, q_3\}$, $\Gamma = (A)$, $F = \{q_3\}$
 $L = \{a^n b^n \mid n \geq 1\}$

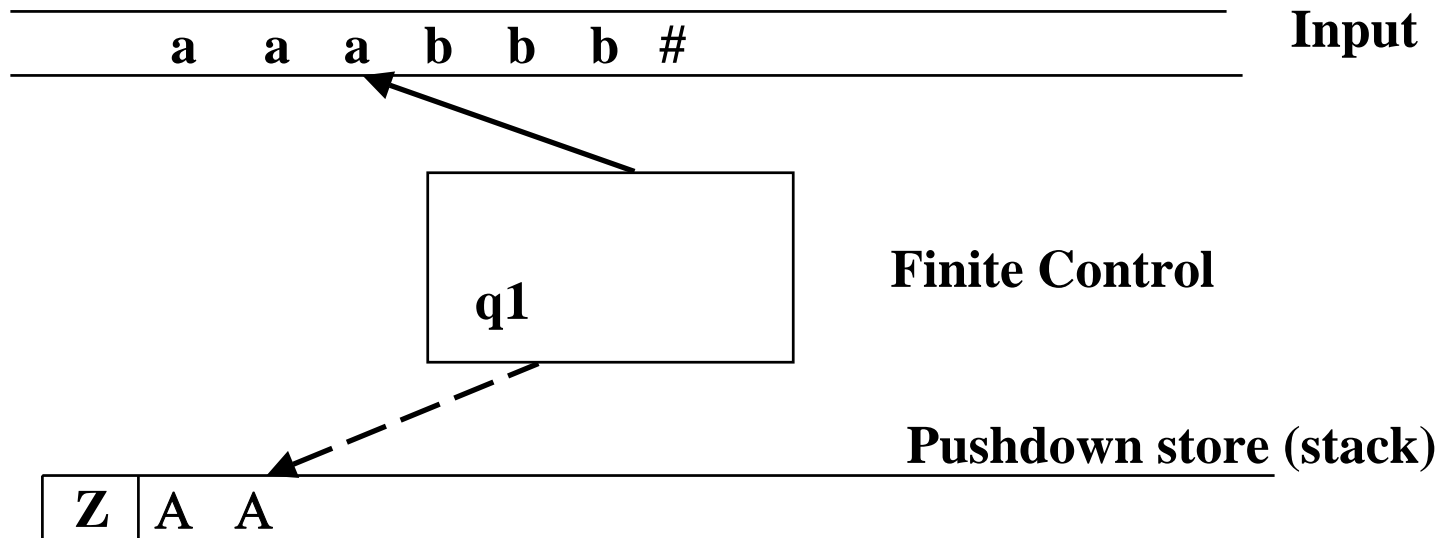


$\delta :$

- $(q_0, a, Z) \rightarrow (q_1, \text{push } A, \text{move right on input})$
- $(q_1, a, A) \rightarrow (q_1, \text{push } A, \text{move right on input})$ ←
- $(q_1, b, A) \rightarrow (q_2, \text{pop } A, \text{move right on input})$
- $(q_2, b, A) \rightarrow (q_2, \text{pop } A, \text{move right on input})$
- $(q_2, \#, Z) \rightarrow (q_3, \text{push } \varepsilon, \text{stay})$

Pushdown Automaton (PDA)

Example: Let $\Sigma = \{a, b, \#\}$, $K = \{q_0, q_1, q_2, q_3\}$, $\Gamma = (A)$, $F = \{q_3\}$
 $L = \{a^n b^n \mid n \geq 1\}$

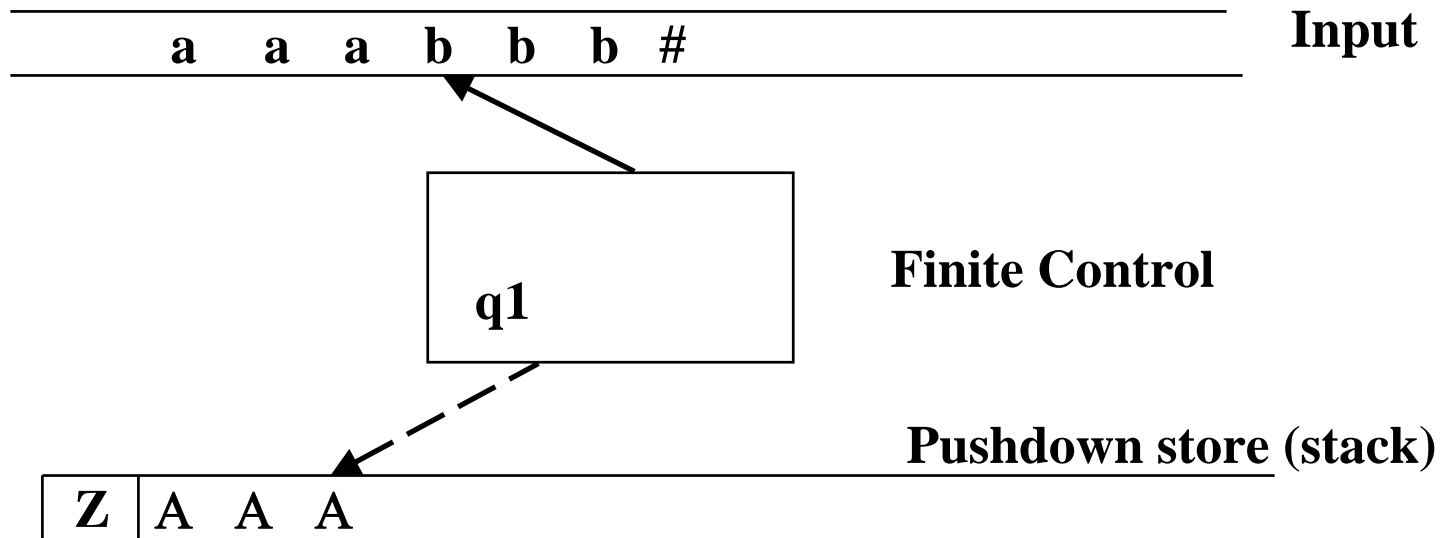


$\delta :$

- $(q_0, a, Z) \rightarrow (q_1, \text{push } A, \text{move right on input})$
- $(q_1, a, A) \rightarrow (q_1, \text{push } A, \text{move right on input})$ ←
- $(q_1, b, A) \rightarrow (q_2, \text{pop } A, \text{move right on input})$
- $(q_2, b, A) \rightarrow (q_2, \text{pop } A, \text{move right on input})$
- $(q_2, \#, Z) \rightarrow (q_3, \text{push } \varepsilon, \text{stay})$

Pushdown Automaton (PDA)

Example: Let $\Sigma = \{a, b, \#\}$, $K = \{q_0, q_1, q_2, q_3\}$, $\Gamma = (A)$, $F = \{q_3\}$
 $L = \{a^n b^n \mid n \geq 1\}$

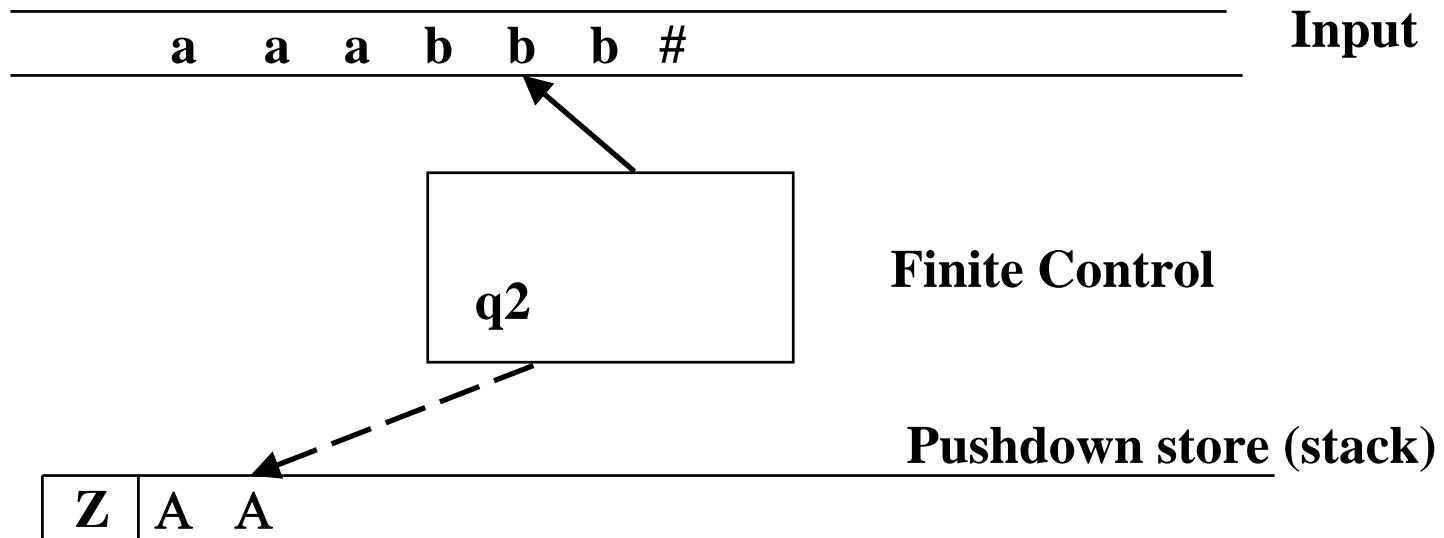


$\delta :$

- $(q_0, a, Z) \rightarrow (q_1, \text{push } A, \text{move right on input})$
- $(q_1, a, A) \rightarrow (q_1, \text{push } A, \text{move right on input})$
- $(q_1, b, A) \rightarrow (q_2, \text{pop } A, \text{move right on input})$ ←
- $(q_2, b, A) \rightarrow (q_2, \text{pop } A, \text{move right on input})$
- $(q_2, \#, Z) \rightarrow (q_3, \text{push } \epsilon, \text{stay})$

Pushdown Automaton (PDA)

Example: Let $\Sigma = \{a, b, \#\}$, $K = \{q_0, q_1, q_2, q_3\}$, $\Gamma = (A)$, $F = \{q_3\}$
 $L = \{a^n b^n \mid n \geq 1\}$

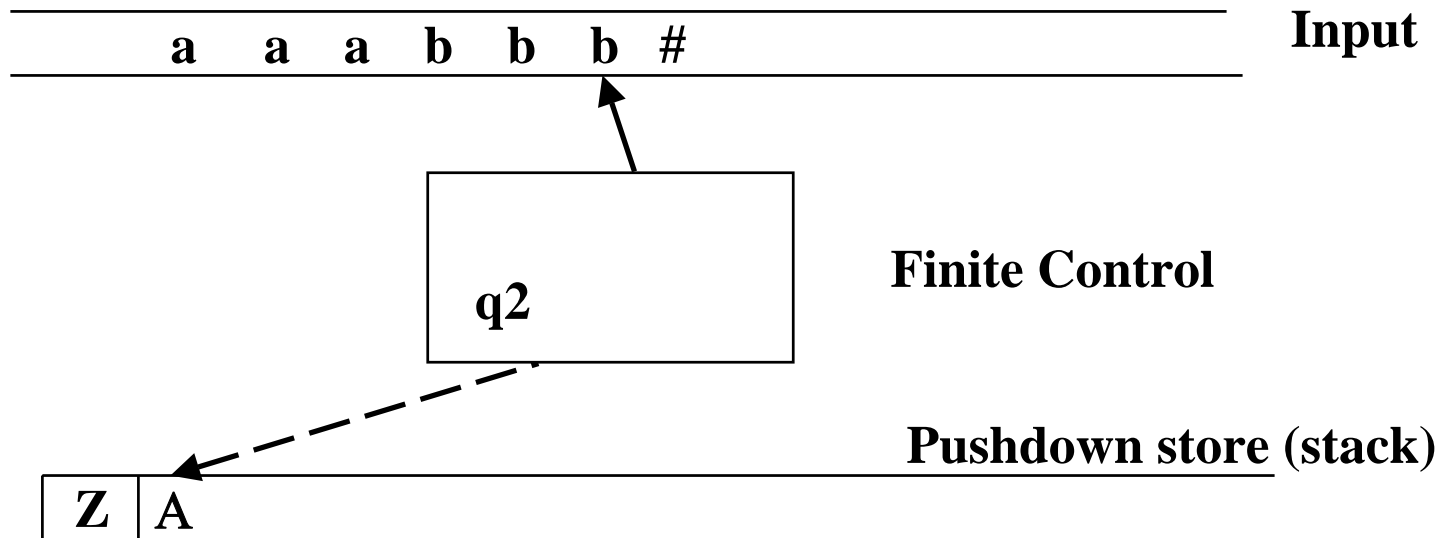


$\delta :$

- $(q_0, a, Z) \rightarrow (q_1, \text{push } A, \text{move right on input})$
- $(q_1, a, A) \rightarrow (q_1, \text{push } A, \text{move right on input})$
- $(q_1, b, A) \rightarrow (q_2, \text{pop } A, \text{move right on input})$
- $(q_2, b, A) \rightarrow (q_2, \text{pop } A, \text{move right on input})$ ←
- $(q_2, \#, Z) \rightarrow (q_3, \text{push } \varepsilon, \text{stay})$

Pushdown Automaton (PDA)

Example: Let $\Sigma = \{a, b, \#\}$, $K = \{q_0, q_1, q_2, q_3\}$, $\Gamma = (A)$, $F = \{q_3\}$
 $L = \{a^n b^n \mid n \geq 1\}$

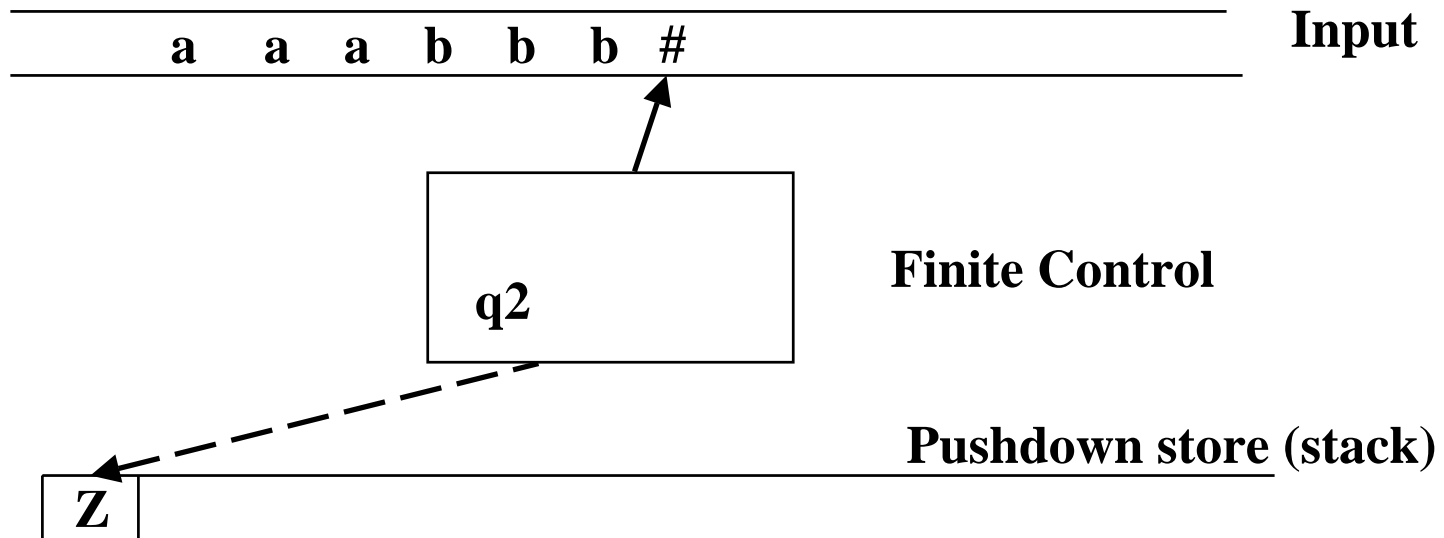


$\delta :$

- $(q_0, a, Z) \rightarrow (q_1, \text{push } A, \text{move right on input})$
- $(q_1, a, A) \rightarrow (q_1, \text{push } A, \text{move right on input})$
- $(q_1, b, A) \rightarrow (q_2, \text{pop } A, \text{move right on input})$
- $(q_2, b, A) \rightarrow (q_2, \text{pop } A, \text{move right on input})$ ←
- $(q_2, \#, Z) \rightarrow (q_3, \text{push } \epsilon, \text{stay})$

Pushdown Automaton (PDA)

Example: Let $\Sigma = \{a, b, \#\}$, $K = \{q_0, q_1, q_2, q_3\}$, $\Gamma = (A)$, $F = \{q_3\}$
 $L = \{a^n b^n \mid n \geq 1\}$

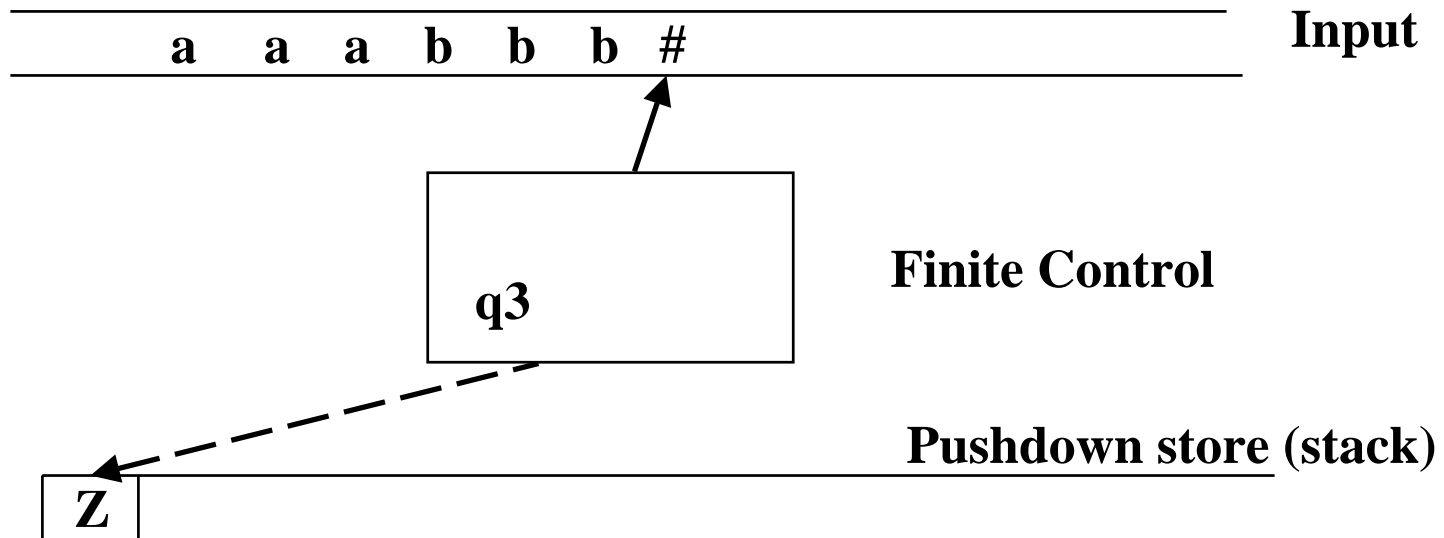


$\delta :$

- $(q_0, a, Z) \rightarrow (q_1, \text{push } A, \text{move right on input})$
- $(q_1, a, A) \rightarrow (q_1, \text{push } A, \text{move right on input})$
- $(q_1, b, A) \rightarrow (q_2, \text{pop } A, \text{move right on input})$
- $(q_2, b, A) \rightarrow (q_2, \text{pop } A, \text{move right on input})$
- $(q_2, \#, Z) \rightarrow (q_3, \text{push } \varepsilon, \text{stay}) \leftarrow$

Pushdown Automaton (PDA)

Example: Let $\Sigma = \{a, b, \#\}$, $K = \{q_0, q_1, q_2, q_3\}$, $\Gamma = (A)$, $F = \{q_3\}$
 $L = \{a^n b^n \mid n \geq 1\}$



$\delta :$

- $(q_0, a, Z) \rightarrow (q_1, \text{push } A, \text{move right on input})$
- $(q_1, a, A) \rightarrow (q_1, \text{push } A, \text{move right on input})$
- $(q_1, b, A) \rightarrow (q_2, \text{pop } A, \text{move right on input})$
- $(q_2, b, A) \rightarrow (q_2, \text{pop } A, \text{move right on input})$
- $(q_2, \#, Z) \rightarrow (q_3, \text{push } \varepsilon, \text{stay}) \leftarrow$

$L = \{ w c w^R \mid w \in \{a, b\}^*, c \text{ is special symbol not in } w, w^R \text{ is the reverse (mirror image) of } w \}$

Construct a PDA, M , that accepts L

What about

$L1 = \{ w w^R \mid w \in \{a, b\}^* \text{ and } w^R \text{ is the reverse (mirror image) of } w \}$

There is a PDA for $L1$ but it is non-deterministic, why?

-- For every context-free language, L , there is a pushdown automaton (PDA), M , possibly, a non-deterministic PDA such that the language accepted by M , i.e., $L(M)$ is exactly L

-- For finite state automata (FSA), for every non-deterministic FSA, M , there is a deterministic FSA, M' , such that $L(M) = L(M')$

-- This correspondence between deterministic and non-deterministic machines does not hold for PDA's

-- Non-deterministic PDA's are strictly more powerful than deterministic PDA's

Examples of languages that are beyond context-free languages, i.e., context-free grammars are not adequate to describe these languages

$$\mathbf{L = \{a^n b^n c^n \mid n \geq 1\}}$$

$\mathbf{L = \{w c w \mid w \in \{a, b\}^*, c \text{ is special symbol not in } w\}}$
(copy language)

$$\mathbf{L = \{a^{2^n} \mid n \geq 1\}}$$

$L = \{a^n b^n c^n \mid n \geq 1\}$ is not a context-free language

There is no pushdown automaton, M , such that M accepts L

$L' = \{a^n b^n \mid n \geq 1\}$ is a context-free language

Is it possible to adapt the pushdown automaton, say, M' that accepts L' for the language L ?

-- suppose for each input a we push two A 's on the stack

-- then for each input b we pop one A , so at the end of the b 's half of the A 's will be popped

-- then for each input c we pop one A

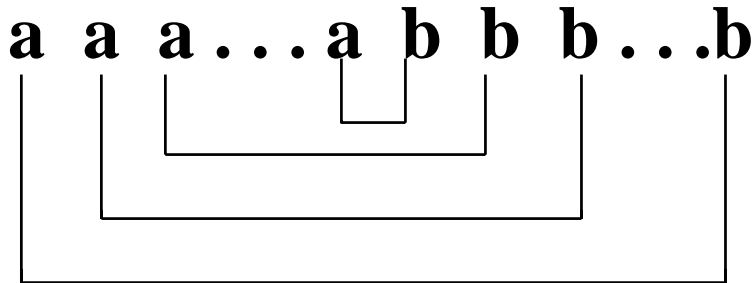
What is wrong with this argument?

Even if a language is context-free, such as for example

$$L = \{a^n b^n \mid n \geq 1\}$$

context-free grammars are inadequate to provide certain kinds of structural descriptions

Context-free grammars are adequate to describe nested dependencies

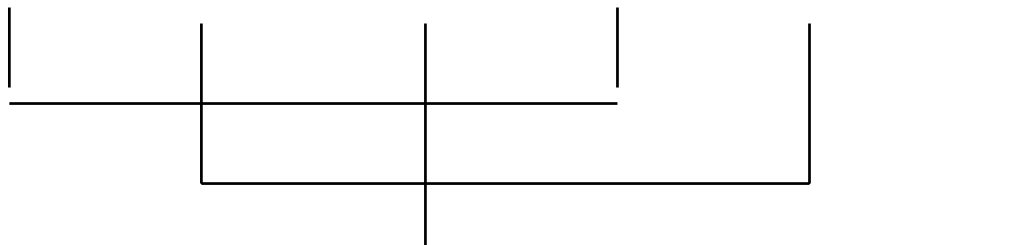


Crossing dependencies

In Dutch in subordinate clause constructions we have crossing dependencies

Jan Piet Marie zag helpen zwemmen

Jan Piet Marie saw help swim



(Jan saw Piet help Marie swim)

A pushdown automaton (PDA) is unable to describe such dependencies, why?

Beyond context-free grammars:

-- Context-sensitive grammars (type 1 grammars):

The rewriting rules are of the form

$$\phi A \psi \rightarrow \phi w \psi$$

where $\phi, \psi, w \in \{V_T \cup V_N\}^*$, $w \neq \text{null string } (\epsilon)$

i.e., strings of terminals and

non-terminals including the null string, except for w ,

$A \in V_N$

--The rewriting of A is controlled by the left context ϕ and the right context ψ , unlike the rewriting rules in a context-free grammar, which are of the form $A \rightarrow w$

-- Unrestricted rewriting grammars (type 0 grammars)

The rewriting rules are of the form

$$\phi \rightarrow \psi$$

where $\phi, \psi \in \{V_T \cup V_N\}^*$, i.e., strings of terminals and non-terminals including the null string, ϕ must contain

~~at least one non-terminal~~
There is no restriction on the form of the rewriting rules, except that ϕ must contain at least one non-terminal

-- Note that the right hand side of a rule can be ϵ . Thus the length of a string in the derivation may shrink or grow as the derivation proceeds

Chomsky Hierarchy

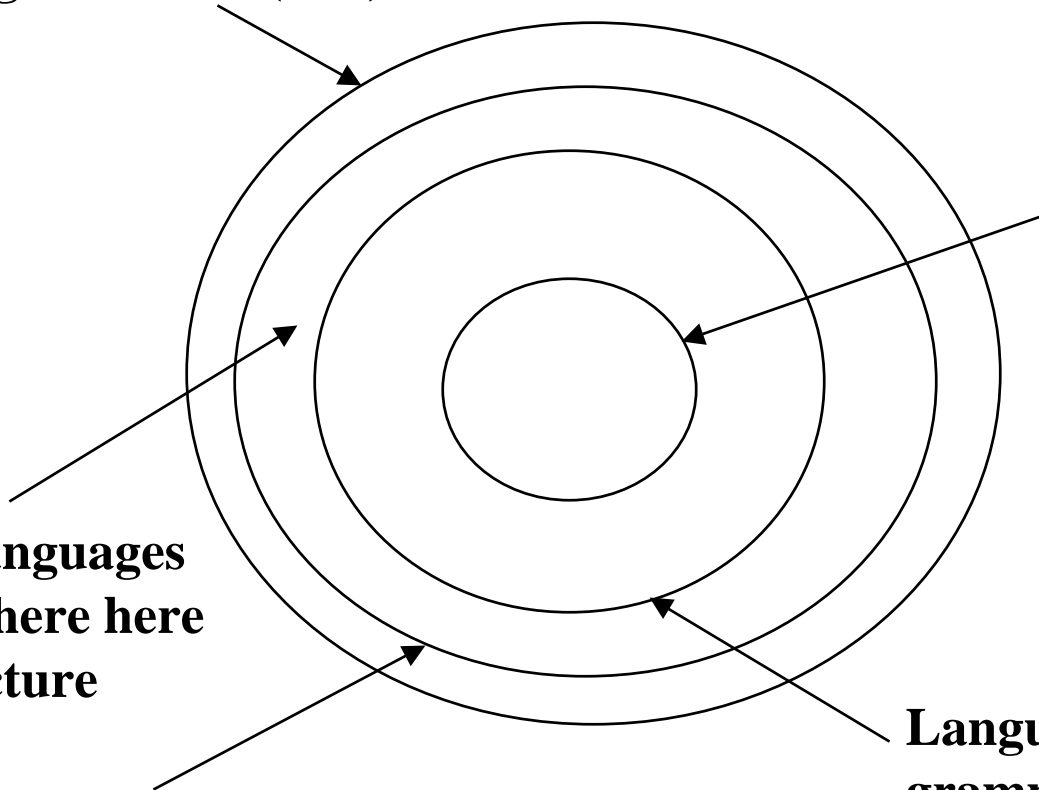
Languages of unrestricted rewriting grammars, type 0 grammars, languages of Turing Machines (TM)

Finite State Languages, regular languages, sets denoted by regular expressions, languages of finite state grammars, type 3 grammars

Natural languages lie somewhere here -- a conjecture

Languages of context-sensitive grammars, type 1 grammars, languages of linear bounded automata

Languages of context-free grammars, type 2 grammars, languages of (non-deterministic) pushdown automata



Problem 1

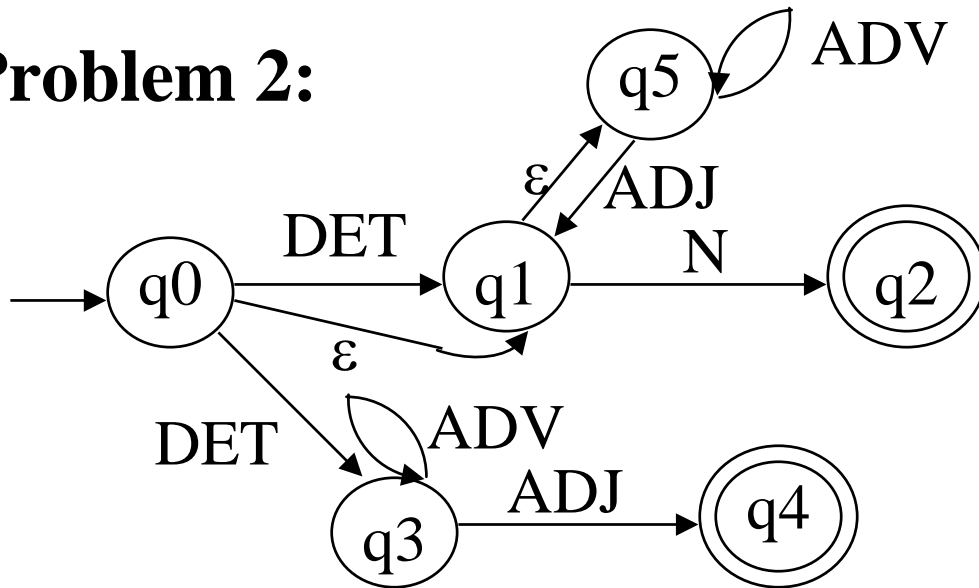
Let $\Sigma = \{0, 1\}$

Let $L =$ the set of all strings of 0's and 1's that contain exactly two 1's

Construct a finite state automaton, M , such that M accepts exactly L

Construct a finite state automaton M' such that M' accepts the complement of L

Problem 2:



-- The finite state automaton, M , for simple noun phrases shown above is non-deterministic, why?

-- Construct a finite state automaton, M' , such that M' is deterministic and M' is equivalent to M

--- explain your construction

Problem 3:

(3a)

$L = \{ w c w^R \mid w \in \{a, b\}^*, c \text{ is special symbol not in } w,$

$w^R \text{ is the reverse (mirror image) of } w\}$

Construct a PDA, M , that accepts

L

(3b)

**$L1 = \{ w w^R \mid w \in \{a, b\}^* \text{ and } w^R \text{ is the}$
reverse (mirror image) of } w\}**

Construct a PDA, M' , that accepts $L1$

M' is necessarily non-deterministic, why?

Processing: computational models and human processing

Parsing

- Sources of complexity**
 - size of the grammar**
 - size of the lexicon**
 - complexity of the rules**
 - sources of ambiguities**

Sources of ambiguities

-- lexical

-- word sense: bank (river)/bank (finance)

-- part of speech: plan (noun)/plan (verb)

clear (adjective)/clear (verb)

duck(noun)/duck (verb)

sent (past verb)/sent (past pass. part.)

-- structural

They are annoying children

((They) (are annoying) (children))

((They) (are) (annoying children))

Sources of ambiguities

-- scopal

Every student knows two languages

- (a) Each student knows two languages. It is not necessary that all students know the same two languages**
- (b) There are two languages that are known by all students**

Sources of ambiguities

-- pragmatic

Can you pass the salt?

-- yes/no question

-- a request

Sources of ambiguities

-- attachment (a kind of structural ambiguity)

The servant of the actress on the balcony

-- *on the balcony* can be attached to *actress*
or to *servant*

The cop saw the man with the binoculars

-- *with the binoculars* can be attached to *man* or to *saw*

Low attachment/ High attachment

Attachment ambiguities

Are there any preferences? High/Low

Factors affecting attachment

-- lexical items

-- distance from the site of attachment

-- context

Tom said that Bill left yesterday

Tom said that Bill left very loudly

Tom said to all the people that Bill left yesterday

The policewoman saw the man with her binoculars

Statistical computational models of attachment

The cop saw the man with the binoculars

Cooccurrence statistics over the quadruples

saw	man	with	binoculars
V	N1	P	N2

Cognitive implications

Local and Global Ambiguities

-- Local ambiguities

**-- an ambiguity that arises at some local level
but it is resolved at the global level**

Have the students take the examination?

Have the students taken the examination?

The manger sent presents to all employees

The package sent yesterday has reached NSF

The manager sent abroad for training has returned

The horse raced past the barn fell

(Garden path sentences)

Global ambiguities

- the entire sentence can be analyzed in more than one way**

- we have seen many examples already**

Here is one more

We saw her duck

- resolution of global ambiguities**

- what kinds of information will be useful?**

- context**

- domain knowledge**

- statistical information**