# Language and Computation

## week 7, Tuesday, February 25, 2014

*Tamás Biró*
*Yale University*
`tamas.biro@yale.edu`
`http://www.birot.hu/courses/2014-LC/`

# Practical matters

- **Post-reading:** Chapters 12 and 16.
- **Pre-reading:** Section 13.1
- **Sections**
- Feedback on **homework 2**.
- **Homework 3** posted, due 03/04.

# Today

**Hidden Markov models** and Ferguson's three problems:

- The Viterbi Algorithm
- The Forward Algorithm
- The Forward-Backward Algorithm

As well as introduction to grammars and CFGs.

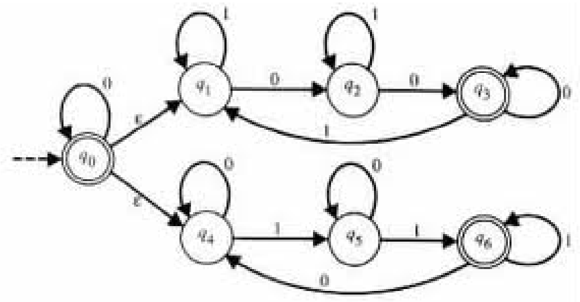
# Hidden Markov Models

# Markov Models: sextuple $(Q, \Sigma, q_0, q_F, A, B)$

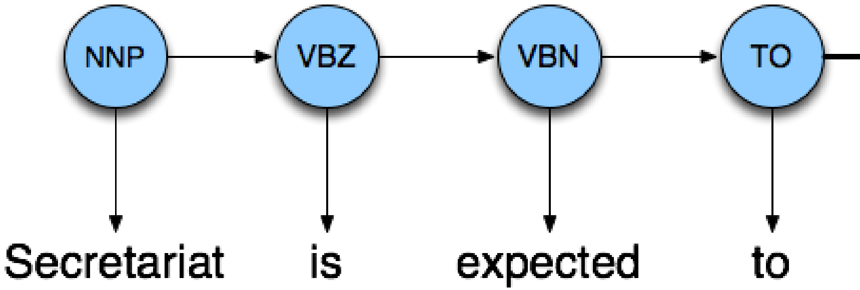
- $Q$ finite set of states $q_1, q_2, \ldots, q_N$.
  $\Sigma$ set of possible observations (finite? not finite?)
- $q_0$ start state (or probability distribution $\pi$ over $Q$)
  $q_F$ end (final) state (or $F \subseteq Q$?)
- $A$ **transition probability matrix**: $\forall i : \sum_{j=1}^{N} a_{ij} = 1$
- $B$ **emission probabilities**: $\forall i : \sum_{o \in \Sigma} b_i(o) = 1$

# Some intro remarks

- Visual representations of FSAs vs. MMs:



(a)




- Probabilities in a ProbFSA

- Markov Chain: “First-order observable Markov Model”

# (Hidden) Markov Models

| | |
|---|---|
| $Q = q_1 q_2 \dots q_N$ | a set of **states** |
| $A = a_{01} a_{02} \dots a_{n1} \dots a_{nn}$ | a **transition probability matrix** $A$, each $a_{ij}$ representing the probability of moving from state $i$ to state $j$, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$ |
| $O = o_1 o_2 \dots o_N$ | a set of **observations**, each one drawn from a vocabulary $V = v_1, v_2, ..., v_V$. |
| $B = b_i(o_t)$ | a set of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation $o_t$ being generated from a state $i$ |
| $q_0, q_{end}$ | special **start and end states** that are not associated with observations |

# (Hidden) Markov Models

**Markov assumption:** $P(q[t_i])$ only depends on $q[t_{i-1}]$, and not on previous states or previous outputs.

**Output independence:** $P(o[t_i])$ only depends on $q[t_i]$ and not on previous states or previous outputs.

# (Hidden) Markov Models

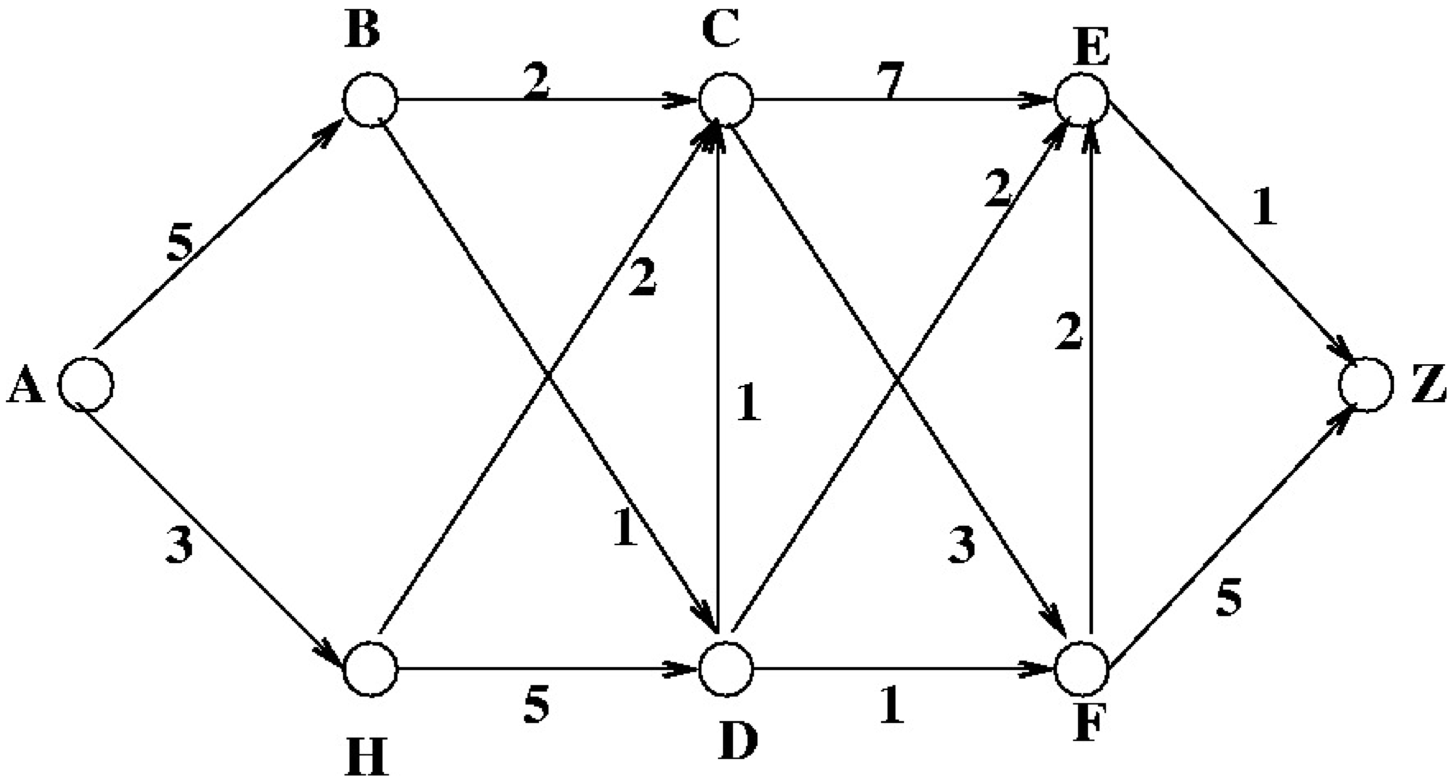
- Given MM $\lambda = (A, B)$, generate series of observation: trivial.

- Given MM $\lambda = (A, B)$, given observation sequence $O$ determine:
  - likelihood $P(O|\lambda)$: **forward algorithm**
  - find most probable sequence of states: **Viterbi algorithm**

- Given an observation sequence $O$, learn $A$ and $B$: **forward-backward algorithm** (aka Baum-Welch algorithm, special case of Expectation-Maximization/EM algorithm).

# Viterbi algorithm

# Dynamic programing




Source: http://lcm.csa.iisc.ernet.in/dsa/node163.html

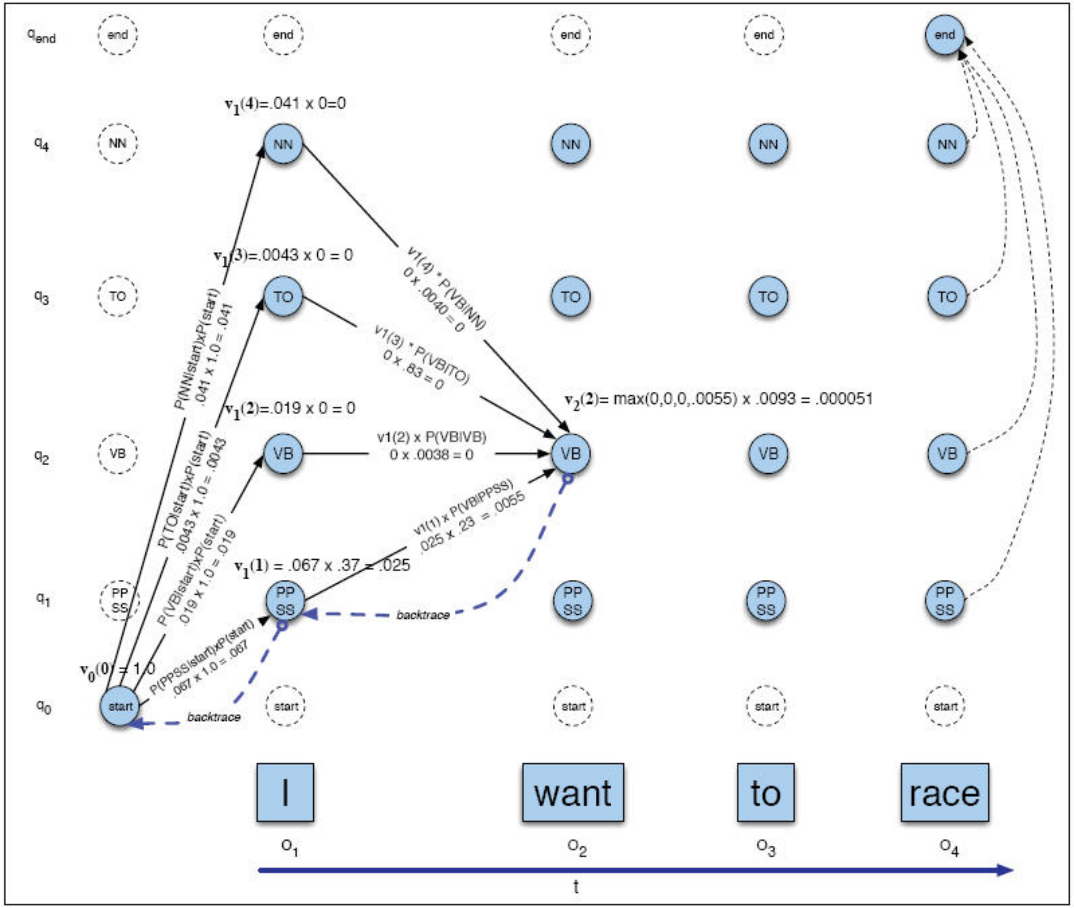
# Viterbi algorithm

Problem: **Decoding**

Given as input an HMM $\lambda = (A, B)$ and a sequence of observations $O = o_1, o_2, \ldots, o_T$, find the most probable sequence of states $Q = q_1, q_2, \ldots, q_T$.

Solution:

**Viterbi algorithm:** dynamic programming, similar to the minimum edit distance algorithm, using a trellis.



$q_{end}$
$q_4$
$q_3$
$q_2$
$q_1$
$q_0$
end
NN
TO
VB
PP SS
start
$v_1(4)$=.041 x 0=0
$v_1(3)$=.0043 x 0 = 0
$v_1(2)$=.019 x 0 = 0
$v_1(1)$ = .067 x .37 = .025
$v_0(0)$ = 1.0
$v_2(2)$= max(0,0,0,.0055) x .0093 = .000051
P(NN|start)xP(start) .041 x 1.0 = .041
P(TO|start)xP(start) .0043 x 1.0 = .0043
P(VB|start)xP(start) .019 x 1.0 = .019
P(PPSS|start)xP(start) .067 x 1.0 = .067
v1(4) * P(VB|NN) 0 x .0040 = 0
v1(3) * P(VB|TO) 0 x .83 = 0
v1(2) x P(VB|VB) 0 x .0038 = 0
v1(1) x P(VB|PPSS) .025 x .23 = .0055
backtrace
backtrace
I
want
to
race
$o_1$
$o_2$
$o_3$
$o_4$
t

# Viterbi algorithm

| | |
|---|---|
| $v_{t-1}(i)$ | the **previous Viterbi path probability** from the previous time step |
| $a_{ij}$ | the **transition probability** from previous state $q_i$ to current state $q_j$ |
| $b_j(o_t)$ | the **state observation likelihood** of the observation symbol $o_t$ given the current state $j$ |

$$\forall j : v_t(j) = \max_{i=1}^{n} \left( v_{t-1}(i) \cdot a_{ij} \cdot b_j(o_t) \right)$$

# Viterbi algorithm

**function** VITERBI(*observations* of len $T$,*state-graph* of len $N$) **returns** *best-path*

create a path probability matrix *viterbi[N+2,T]*
**for** each state $s$ **from** 1 **to** $N$ **do** ; initialization step
$viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$
$backpointer[s,1] \leftarrow 0$
**for** each time step $t$ **from** 2 **to** $T$ **do** ; recursion step
**for** each state $s$ **from** 1 **to** $N$ **do**

$$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s', t-1] * a_{s',s} * b_s(o_t)$$

$$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s', t-1] * a_{s',s}$$

$$viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$$ ; termination step

$$backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$$ ; termination step

**return** the backtrace path by following backpointers to states back in time from $backpointer[q_F,T]$

# The Forward Algorithm

# Forward algorithm

Problem: **Likelihood**

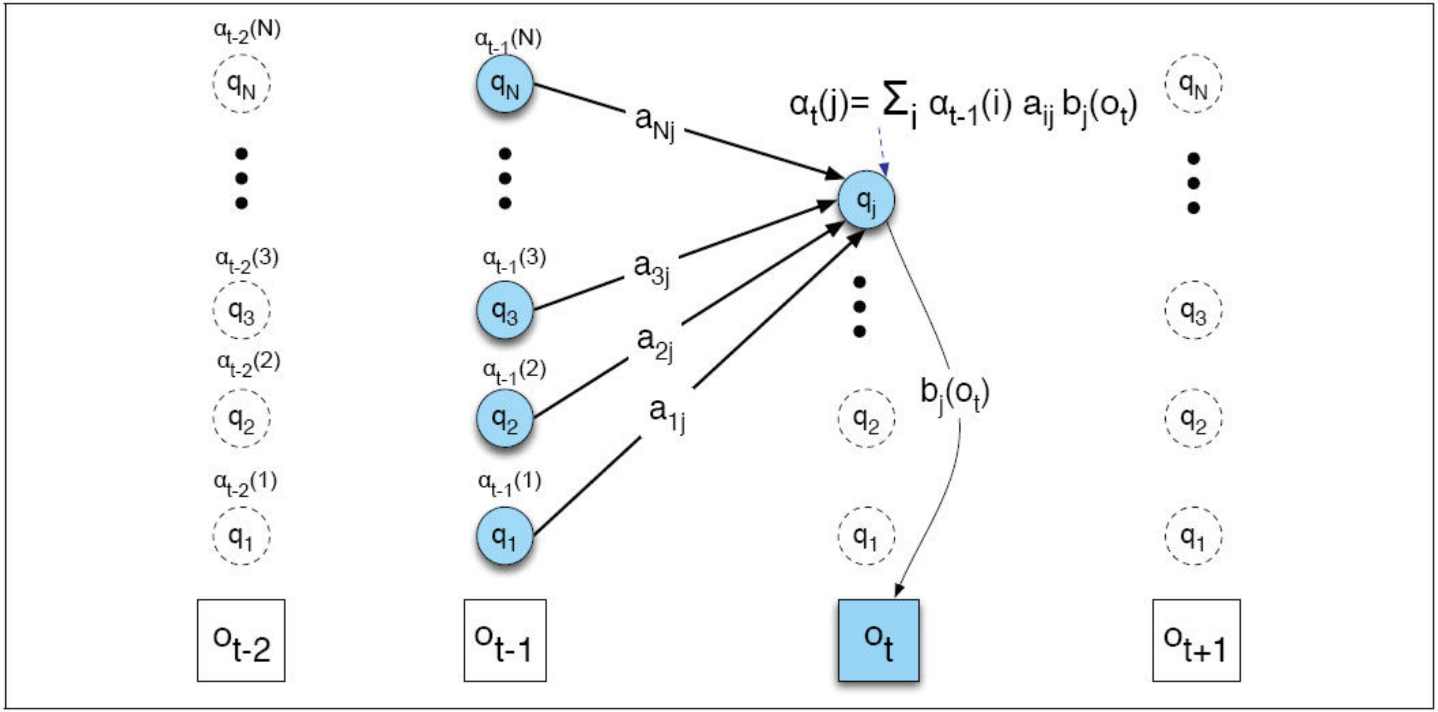
Given as input an HMM $\lambda = (A, B)$ and a sequence of observations $O = o_1, o_2, \ldots, o_T$, determine the **likelihood** $P(O|\lambda)$, the probability that HMM $\lambda$ emits series $O$.

$$P(O|\lambda) = \sum_{q[t_1],\ldots,q[t_T]} P(o_1, \ldots, o_T \ | \ q[t_1], \ldots, q[t_T], \lambda)$$

Solution:

**Forward algorithm:** dynamic programming, similar to the minimum edit distance algorithm, using a trellis.

# Forward algorithm



# Forward algorithm

| | |
|---|---|
| $\alpha_{t-1}(i)$ | the **previous forward path probability** from the previous time step |
| $a_{ij}$ | the **transition probability** from previous state $q_i$ to current state $q_j$ |
| $b_j(o_t)$ | the **state observation likelihood** of the observation symbol $o_t$ given the current state $j$ |

$$\forall j : \alpha_t(j) = \sum_{i=1}^{n} \alpha_{t-1}(i) \cdot a_{ij} \cdot b_j(o_t)$$

# Forward algorithm

**function** FORWARD(*observations* of len $T$, *state-graph* of len $N$) **returns** *forward-prob*

create a probability matrix *forward[N+2,T]*
**for** each state $s$ **from** 1 **to** $N$ **do** ; initialization step
$\quad forward[s,1] \leftarrow a_{0,s} * b_s(o_1)$
**for** each time step $t$ **from** 2 **to** $T$ **do** ; recursion step
$\quad$ **for** each state $s$ **from** 1 **to** $N$ **do**

$$forward[s,t] \leftarrow \sum_{s'=1}^{N} forward[s',t-1] * a_{s',s} * b_s(o_t)$$

$$forward[q_F,\mathrm{T}] \leftarrow \sum_{s=1}^{N} forward[s,T] * a_{s,q_F}$$ ; termination step

**return** $forward[q_F,T]$

# The Forward-Backward Algorithm

# Forward-Backward algorithm

Problem:

Given as input an observation sequence $O = o_1, o_2, \ldots, o_T$ and the set of possible states in the HMM, **learn** the HMM parameters $A$ and $B$.

Solution: **Forward-Backward algorithm:**

a.k.a. **Baum-Welch Algorithm**, a special case of the **Expectation-Maximization** (EM) algorithm.

an example of **unsupervised learning**!

# Forward-Backward algorithm

- **Forward probability** $\alpha_t(i)$: probability of seeing the observations from time beginning to $t$, given that we are in state $i$ at time $t$, and given HMM.
  *To compute as described in the Forward Algorithm.*
- **Backward probability** $\beta_t(i)$: probability of seeing the observations from time $t+1$ to the end, given that we are in state $i$ at time $t$, and given HMM.
  *To compute by analogy to the Forward Algorithm.*

$$\alpha_t(i) \cdot a_{ij} \cdot b_j(o_{t+1}) \cdot \beta_{t+1}(j)$$

# Forward-Backward algorithm

1. Observation $O$ is given.
   Initialize $A$ and $B$ (in a clever way)

2. Iterate until convergence

   (a) **E-step:** given $O$ and given current HMM ($A$ and $B$), compute $\forall t, j$
      i. forward probability $\alpha_t(j)$ and backward probability $\beta_t(j)$
      ii. expected state occupancy count $\gamma_t(j)$: probability of being in state $j$ at time $t$ (by using $\alpha_t(j)$ and $\beta_t(j)$)
      iii. expected state transition count $\xi_t(i, j)$: probability of being in state $i$ at time $t$ and state $j$ at time $t+1$ (by using $\alpha_t(j)$, $\beta_t(j)$)

   (b) **M-step:** recompute $A$ and $B$ probabilities, given current $\xi$ and $\gamma$.

3. Return final values of $A$ and $B$.

**function** FORWARD-BACKWARD(*observations* of len $T$, *output vocabulary* $V$, *hidden state set* $Q$) **returns** *HMM=(A,B)*

**initialize** $A$ and $B$
**iterate** until convergence
**E-step**

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)} \; \forall\, t \text{ and } j$$

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{ij} b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(N)} \; \forall\, t,\ i, \text{ and } j$$

**M-step**

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \xi_t(i,j)}$$

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t.\ O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

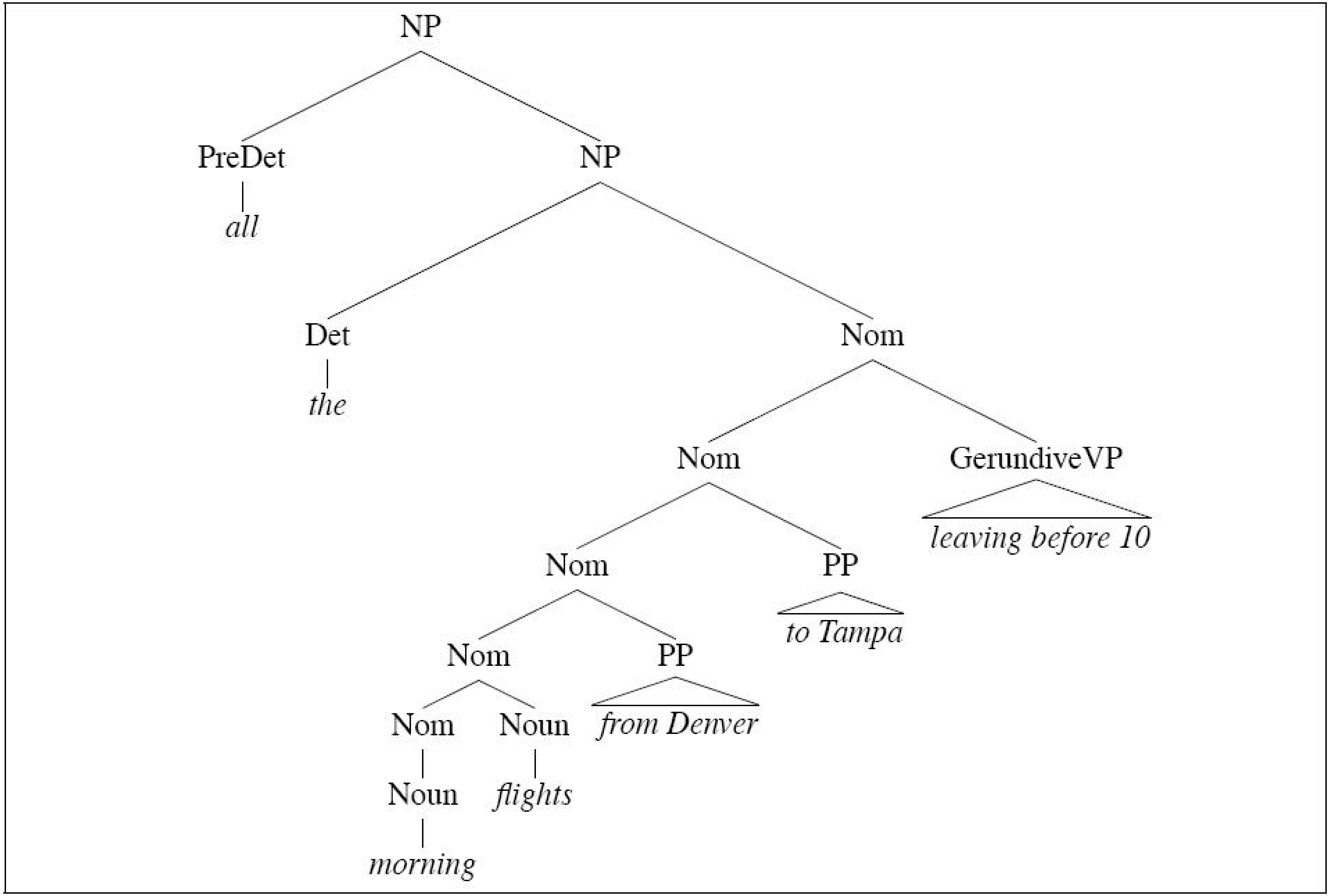
**return** $A$, $B$

# Intro to syntax

# Formal Grammars

| | |
|---|---|
| $N$ | a set of **non-terminal symbols** (or **variables**) |
| $\Sigma$ | a set of **terminal symbols** (disjoint from $N$) |
| $R$ | a set of **rules** or productions, each of the form $A \rightarrow \beta$ , where $A$ is a non-terminal, $\beta$ is a string of symbols from the infinite set of strings $(\Sigma \cup N)*$ |
| $S$ | a designated **start symbol** |

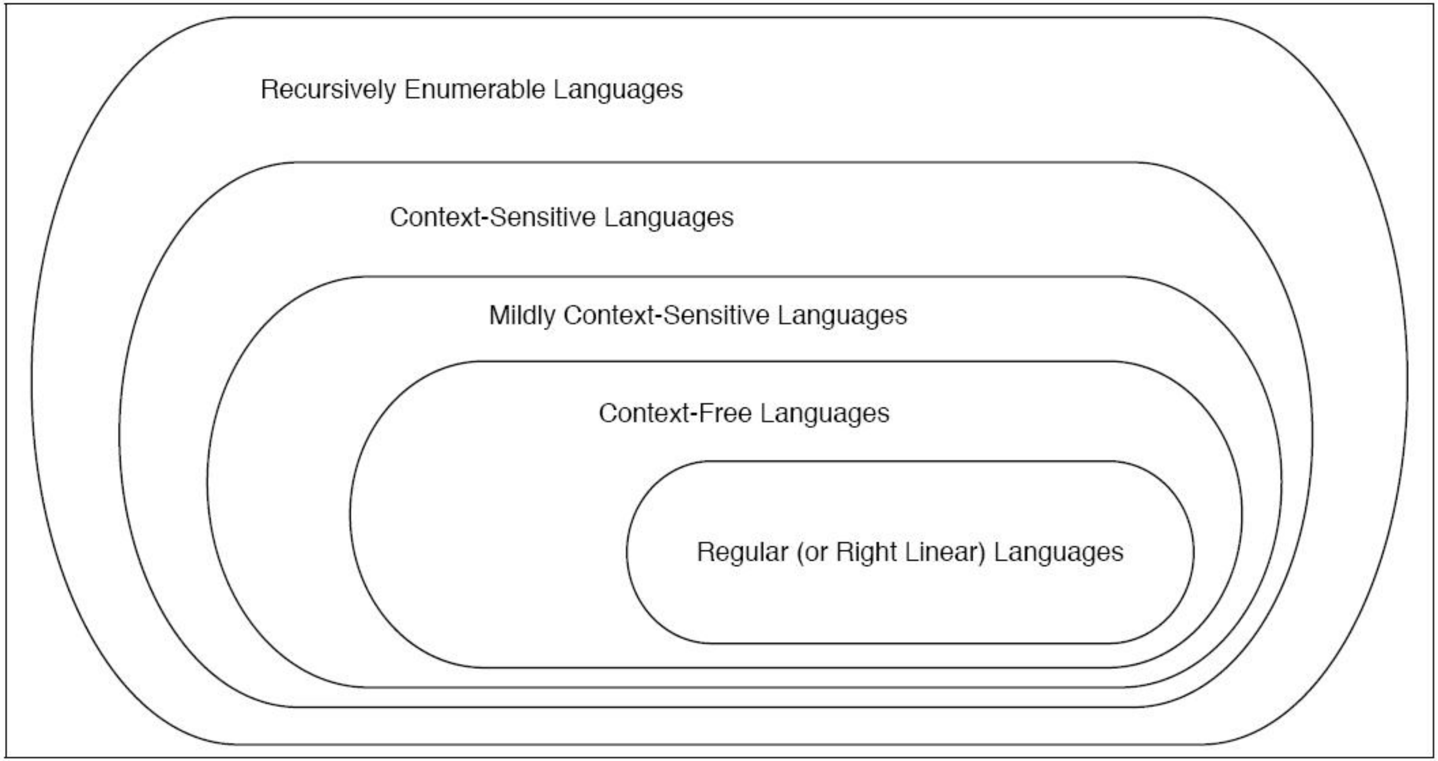
| | |
|---|---|
| Capital letters like $A$, $B$, and $S$ | Non-terminals |
| $S$ | The start symbol |
| Lower-case Greek letters like $\alpha$, $\beta$, and $\gamma$ | Strings drawn from $(\Sigma \cup N)*$ |
| Lower-case Roman letters like $u$, $v$, and $w$ | Strings of terminals |


NP
PreDet
all
NP
Det
the
Nom
Nom
GerundiveVP
leaving before 10
Nom
PP
to Tampa
Nom
PP
from Denver
Nom
Noun
Noun
flights
morning

# Formal Grammars

| Frame | Verb | Example |
|---|---|---|
| $\emptyset$ | eat, sleep | I ate |
| *NP* | prefer, find, leave | Find [$_{NP}$ the flight from Pittsburgh to Boston] |
| *NP NP* | show, give | Show [$_{NP}$ me] [$_{NP}$ airlines with flights from Pittsburgh] |
| $PP_{\text{from}}$ $PP_{\text{to}}$ | fly, travel | I would like to fly [$_{PP}$ from Boston] [$_{PP}$ to Philadelphia] |
| *NP* $PP_{\text{with}}$ | help, load | Can you help [$_{NP}$ me] [$_{PP}$ with a flight] |
| *VPto* | prefer, want, need | I would prefer [$_{VPto}$ to go by United airlines] |
| *VPbrst* | can, would, might | I can [$_{VPbrst}$ go from Boston] |
| *S* | mean | Does this mean [$_{S}$ AA has a hub in Boston] |

# Chomsky hierarchy



# Chomsky hierarchy

| Type | Common Name | Rule Skeleton | Linguistic Example |
|---|---|---|---|
| 0 | Turing Equivalent | $\alpha \rightarrow \beta$, s.t. $\alpha \neq \epsilon$ | HPSG, LFG, Minimalism |
| 1 | Context Sensitive | $\alpha A \beta \rightarrow \alpha \gamma \beta$, s.t. $\gamma \neq \epsilon$ | |
| – | Mildly Context Sensitive | | TAG, CCG |
| 2 | Context Free | $A \rightarrow \gamma$ | Phrase-Structure Grammars |
| 3 | Regular | $A \rightarrow xB$ or $A \rightarrow x$ | Finite-State Automata |

NB:

0: Turing machine

1: Linear bounded automaton

2: Non-deterministic push-down automaton

3: Finite-state automaton

See you on Thursday!