# 6.891: Lecture 8 (October 1st, 2003)

# Log-Linear Models for Parsing, and the EM Algorithm Part I

# Overview

- Ratnaparkhi's Maximum-Entropy Parser

- The EM Algorithm Part I

# Log-Linear Taggers: Independence Assumptions

- The input sentence $S$, with length $n = S.length$, has $|\mathcal{T}|^n$ possible tag sequences.

- Each tag sequence $T$ has a conditional probability

$$P(T \mid S) = \prod_{j=1}^{n} P(T(j) \mid S, j, T(1) \ldots T(j-1)) \quad \text{Chain rule}$$

$$= \prod_{j=1}^{n} P(T(j) \mid S, j, T(j-2), T(j-1)) \quad \text{Independence assumptions}$$

- Estimate $P(T(j) \mid S, j, T(j-2), T(j-1))$ using log-linear models

- Use the Viterbi algorithm to compute

$$\text{argmax}_{T \in \mathcal{T}^n} \log P(T \mid S)$$

# A General Approach: (Conditional) History-Based Models

- We've shown how to define $P(T \mid S)$ where $T$ is a tag sequence

- How do we define $P(T \mid S)$ if $T$ is a parse tree (or another structure)?

# A General Approach: (Conditional) History-Based Models

- Step 1: represent a tree as a sequence of **decisions** $d_1 \ldots d_m$

  $$T = \langle d_1, d_2, \ldots d_m \rangle$$

  $m$ is **not** necessarily the length of the sentence

- Step 2: the probability of a tree is

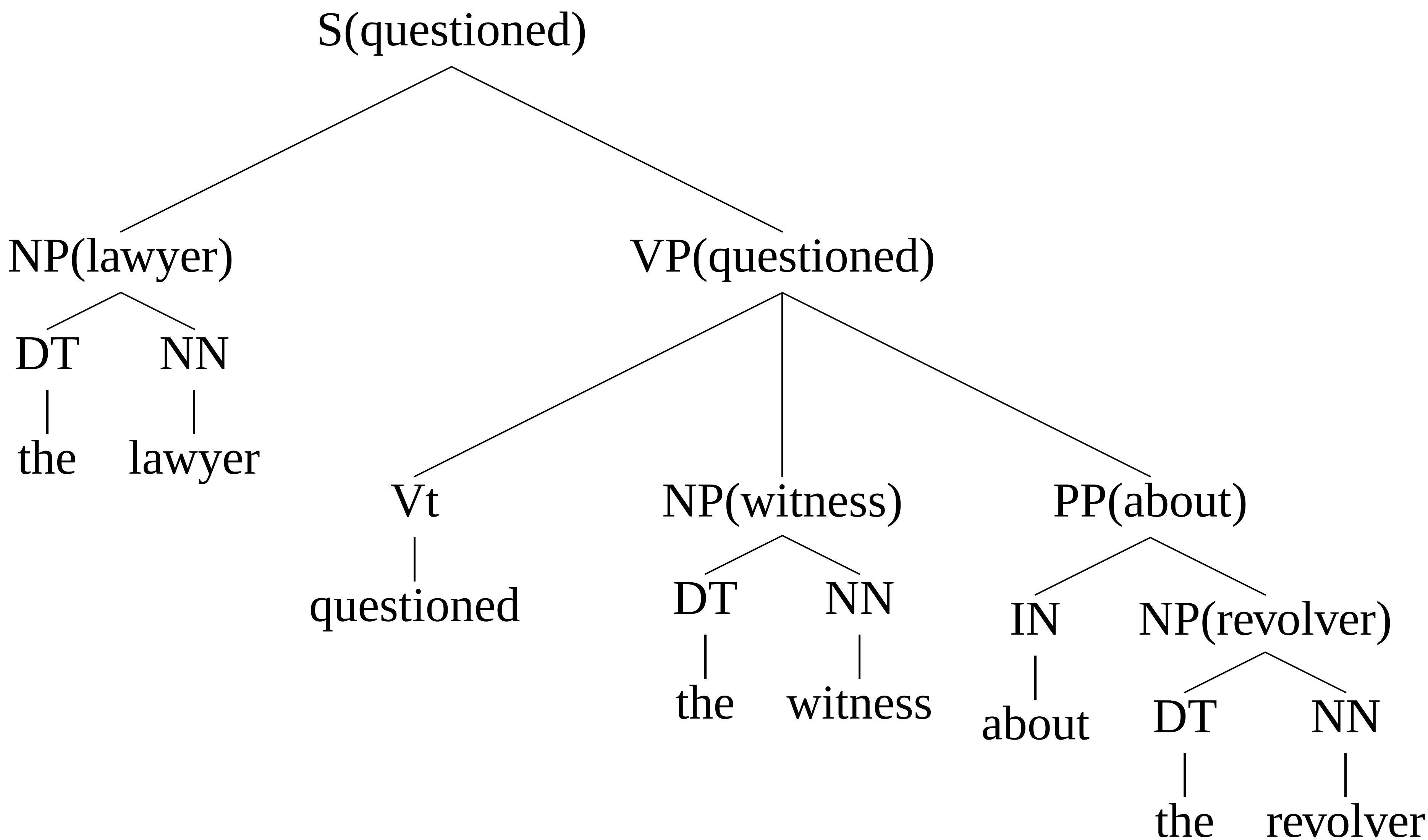
  $$P(T \mid S) = \prod_{i=1}^{m} P(d_i \mid d_1 \ldots d_{i-1}, S)$$

- Step 3: Use a log-linear model to estimate

  $$P(d_i \mid d_1 \ldots d_{i-1}, S)$$

- Step 4: Search?? (answer we'll get to later: beam or heuristic search)

# An Example Tree



# Ratnaparkhi’s Parser: Three Layers of Structure

1. Part-of-speech tags
2. Chunks
3. Remaining structure

# Layer 1: Part-of-Speech Tags

| DT | NN | Vt | DT | NN | IN | DT | NN |
|---|---|---|---|---|---|---|---|
| the | lawyer | questioned | the | witness | about | the | revolver |

---

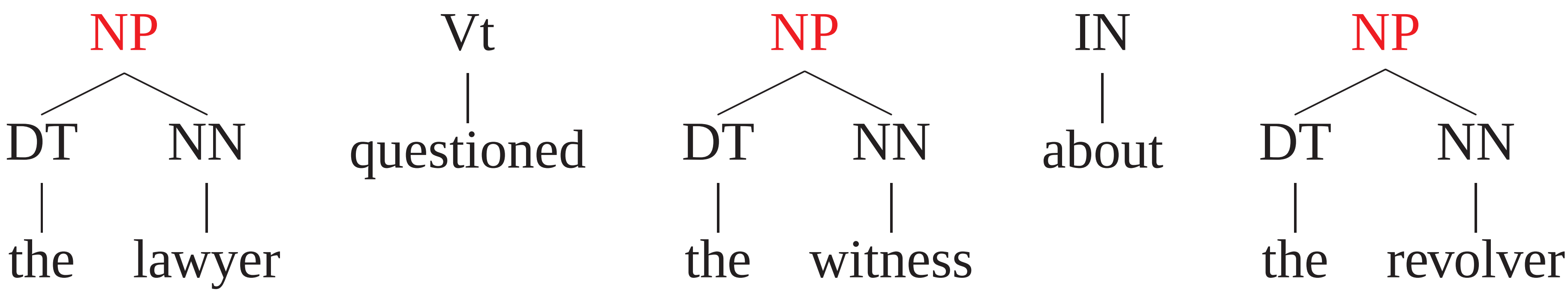
- Step 1: represent a tree as a sequence of **decisions** $d_1 \dots d_m$

$$T = \langle d_1, d_2, \dots d_m \rangle$$

- First $n$ decisions are tagging decisions

  $\langle d_1 \dots d_n \rangle = \langle$ DT, NN, Vt, DT, NN, IN, DT, NN $\rangle$

# Layer 2: Chunks




**Chunks are defined as any phrase where all children are part-of-speech tags**

(Other common chunks are `ADJP`, `QP`)

# Layer 2: Chunks

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Start(NP) | Join(NP) | Other | Start(NP) | Join(NP) | Other | Start(NP) | Join(NP) |
| DT | NN | Vt | DT | NN | IN | DT | NN |
| the | lawyer | questioned | the | witness | about | the | revolver |

---

- Step 1: represent a tree as a sequence of **decisions** $d_1 \ldots d_n$

$$T = \langle d_1, d_2, \ldots d_n \rangle$$

- First $n$ decisions are tagging decisions
  Next $n$ decisions are chunk tagging decisions

$$\langle d_1 \ldots d_{2n} \rangle = \langle \text{ DT, NN, Vt, DT, NN, IN, DT, NN, Start(NP), Join(NP), Other, Start(NP), Join(NP), Other, Start(NP), Join(NP)} \rangle$$

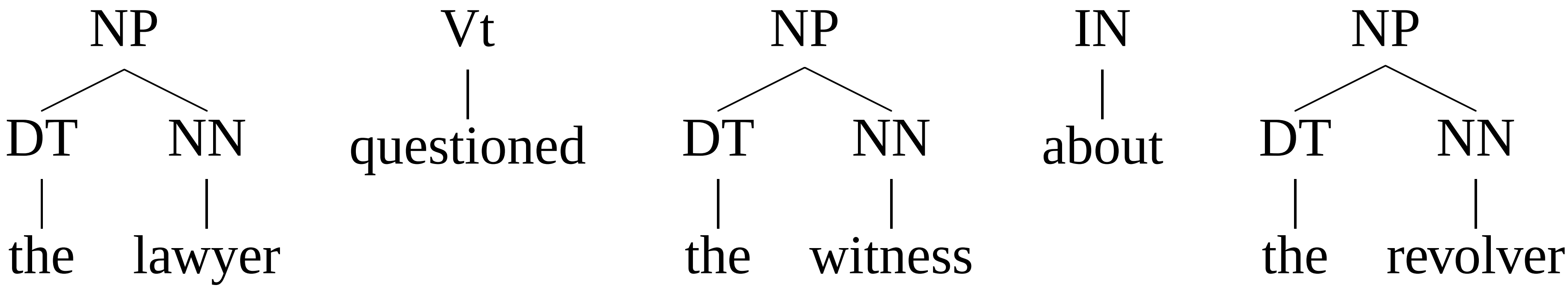
# Layer 3: Remaining Structure

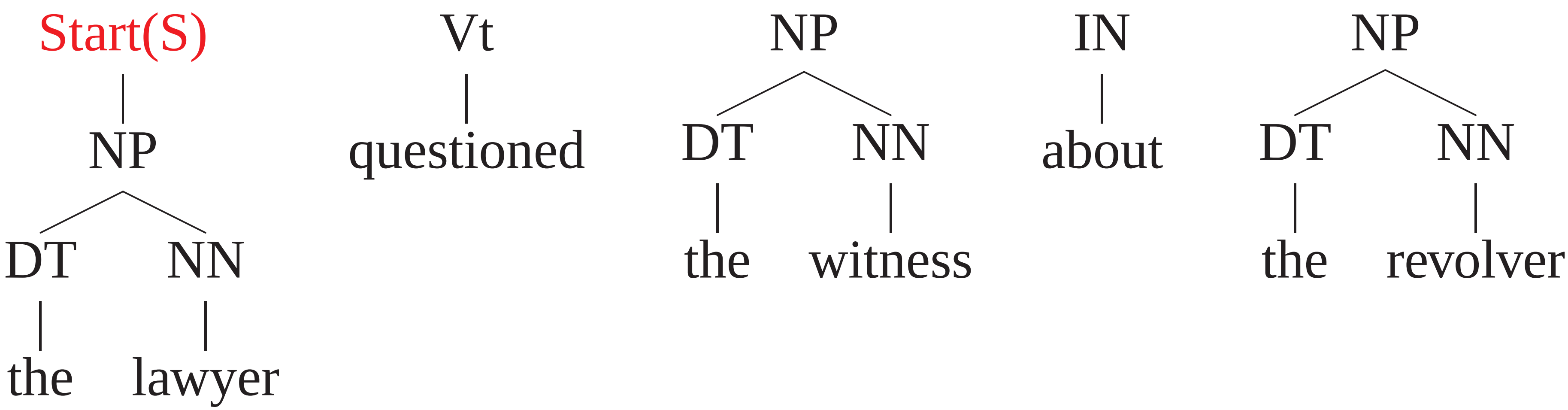
**Alternate Between Two Classes of Actions:**

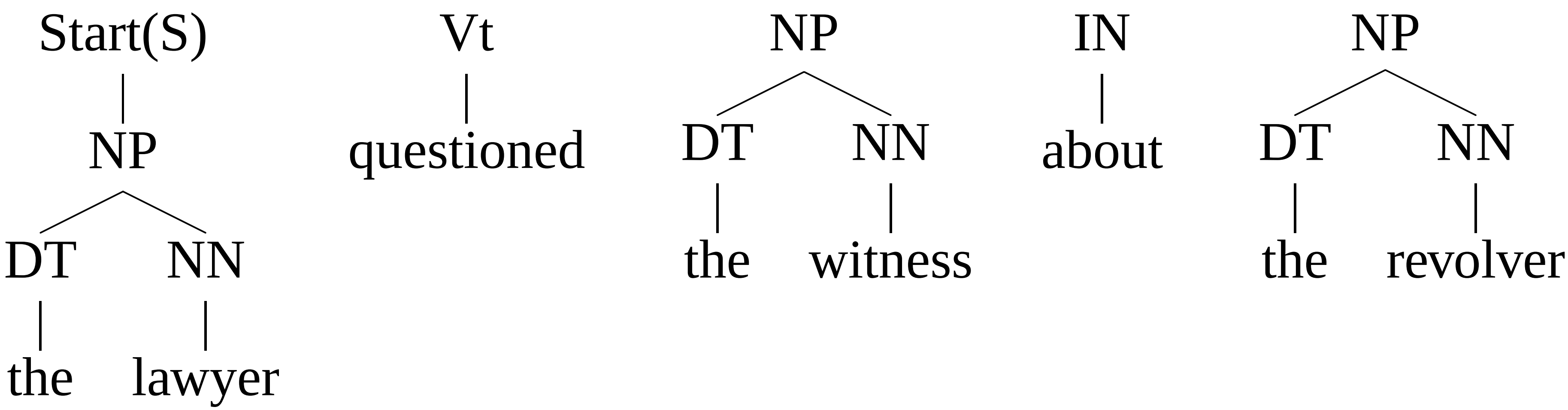
- Join(X) or Start(X), where X is a label (NP, S, VP etc.)
- Check=YES or Check=NO

**Meaning of these actions:**

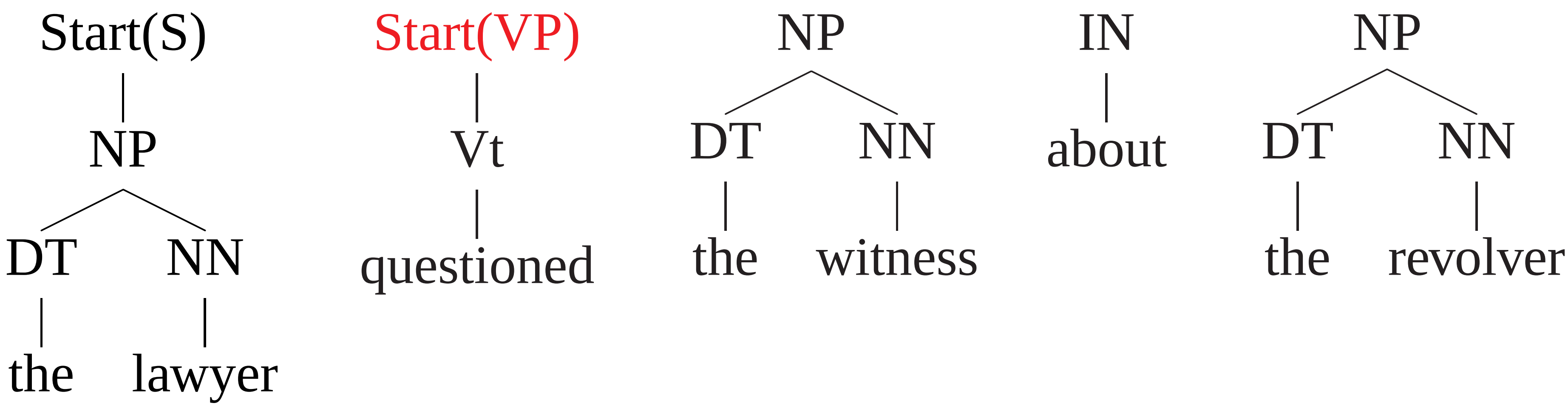
- Start(X) starts a new constituent with label X
  (always acts on leftmost constituent with no start or join label above it)
- Join(X) continues a constituent with label X
  (always acts on leftmost constituent with no start or join label above it)
- Check=NO does nothing
- Check=YES takes previous Join or Start action, and converts it into a completed constituent


NP
DT
NN
the
lawyer
Vt
questioned
NP
DT
NN
the
witness
IN
about
NP
DT
NN
the
revolver

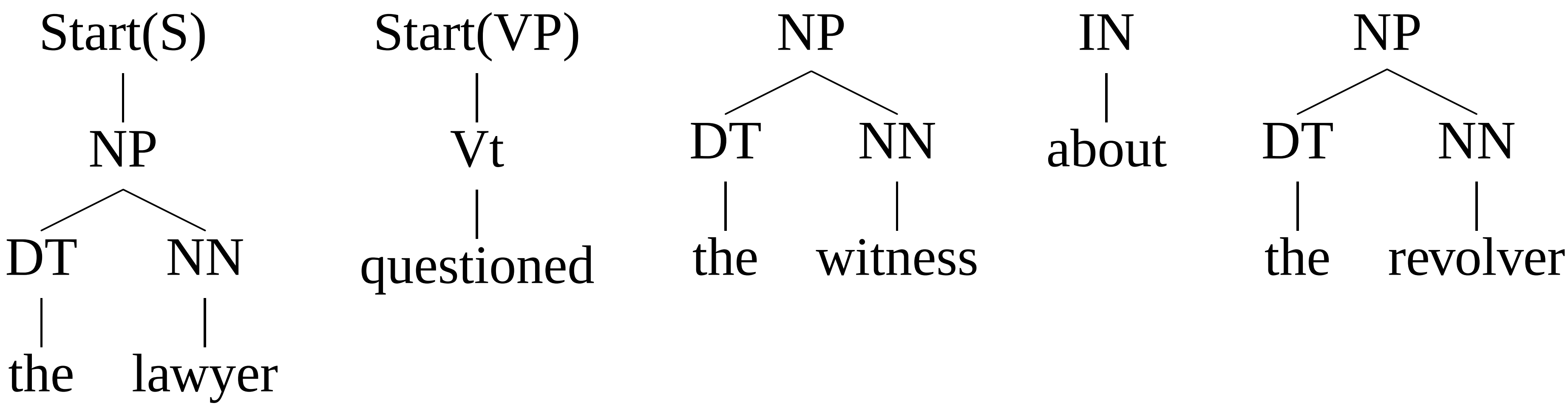

Start(S)
NP
DT
NN
the
lawyer
Vt
questioned
NP
DT
NN
the
witness
IN
about
NP
DT
NN
the
revolver


Start(S)
NP
DT
NN
the
lawyer
Vt
questioned
NP
DT
NN
the
witness
IN
about
NP
DT
NN
the
revolver

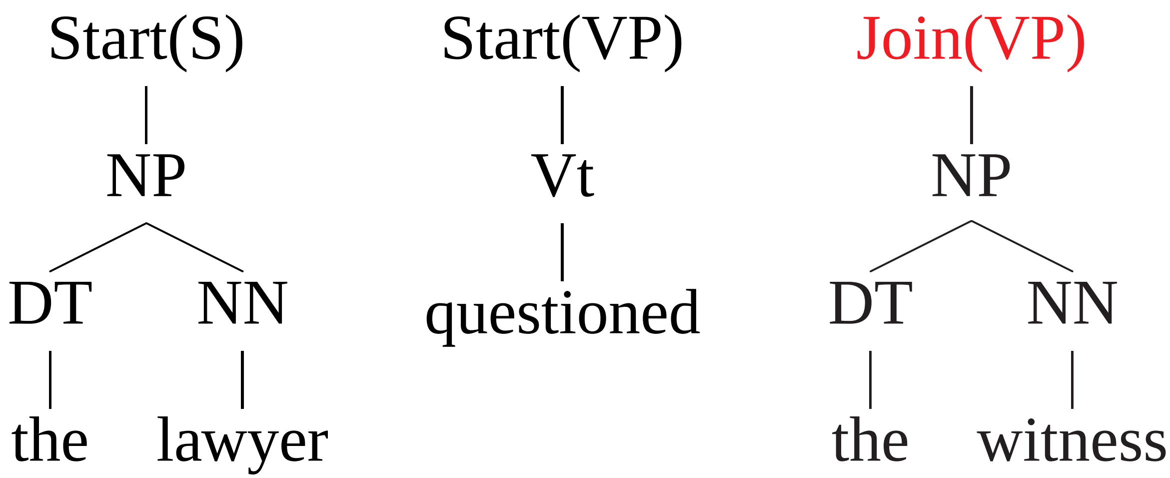
Check=NO

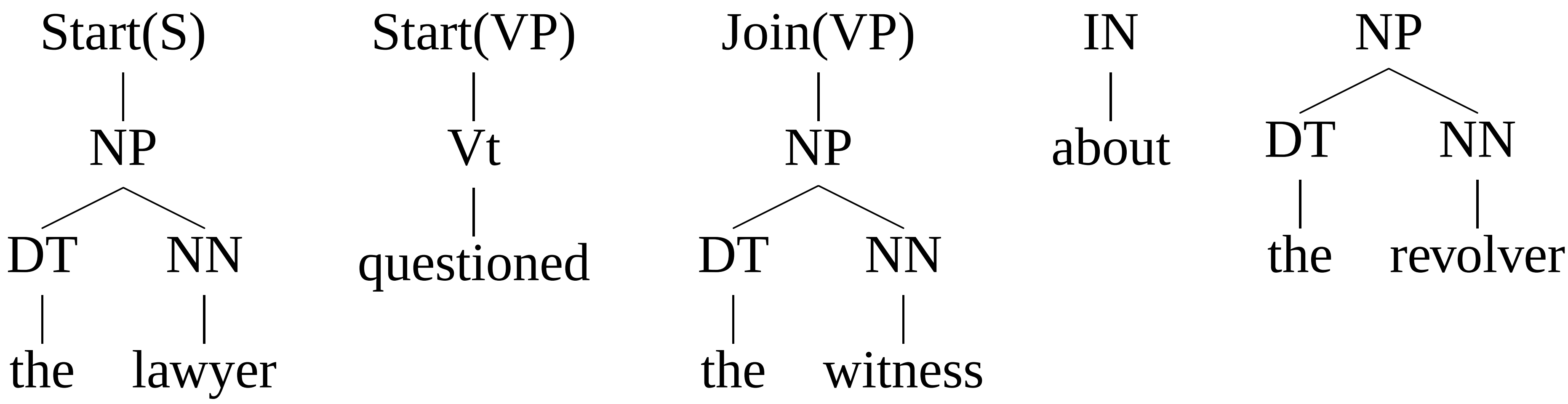


Start(S)
NP
DT
NN
the
lawyer
Start(VP)
Vt
questioned
NP
DT
NN
the
witness
IN
about
NP
DT
NN
the
revolver

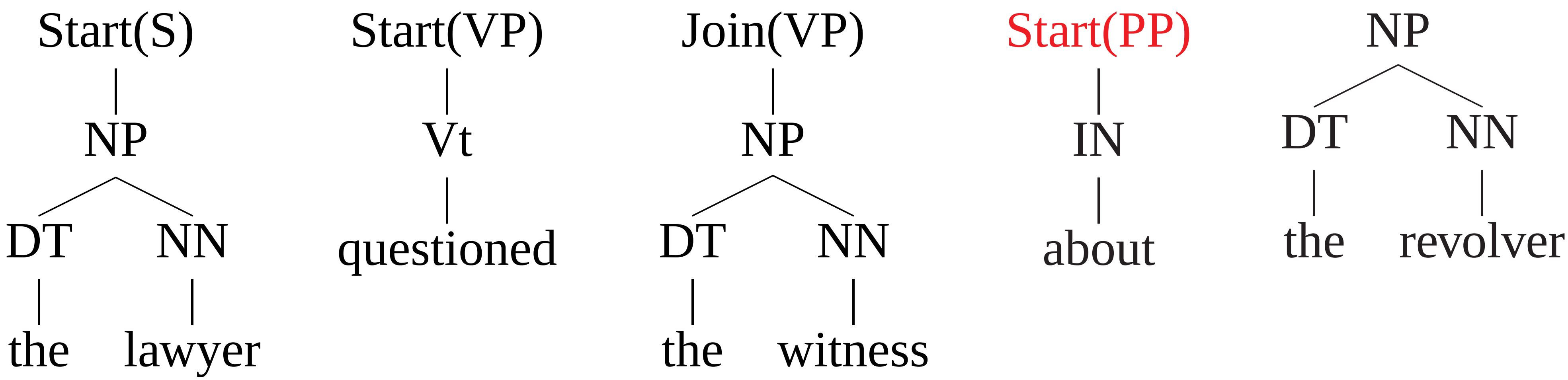

Start(S)
NP
DT
NN
the
lawyer
Start(VP)
Vt
questioned
NP
DT
NN
the
witness
IN
about
NP
DT
NN
the
revolver

Check=NO

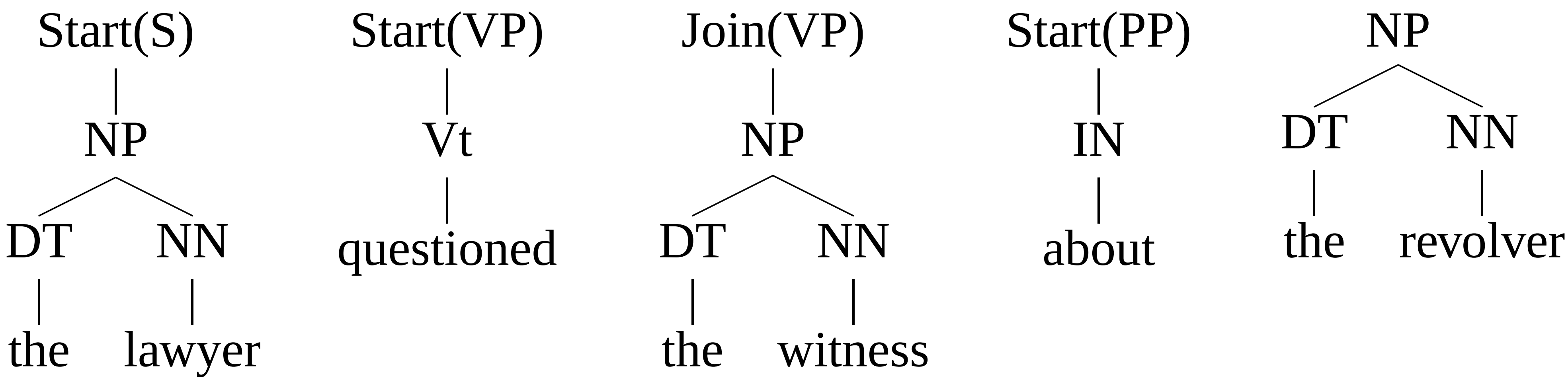

Start(S)
NP
DT
NN
the
lawyer
Start(VP)
Vt
questioned
Join(VP)
NP
DT
NN
the
witness
IN
about
NP
DT
NN
the
revolver


Start(S)
NP
DT
NN
the
lawyer
Start(VP)
Vt
questioned
Join(VP)
NP
DT
NN
the
witness
IN
about
NP
DT
NN
the
revolver

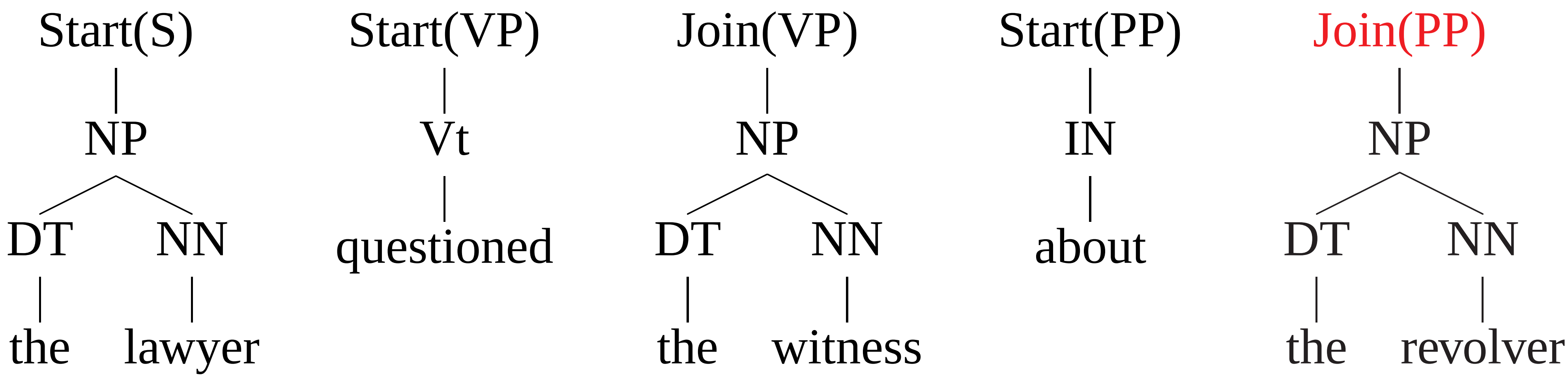
Check=NO

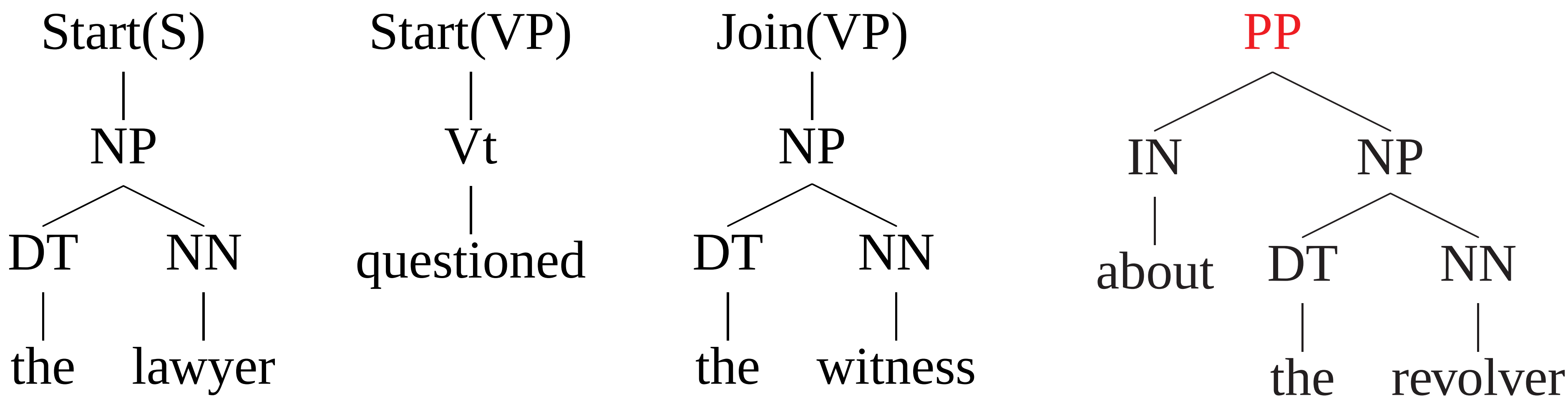


Start(S)
NP
DT
NN
the
lawyer
Start(VP)
Vt
questioned
Join(VP)
NP
DT
NN
the
witness
Start(PP)
IN
about
NP
DT
NN
the
revolver

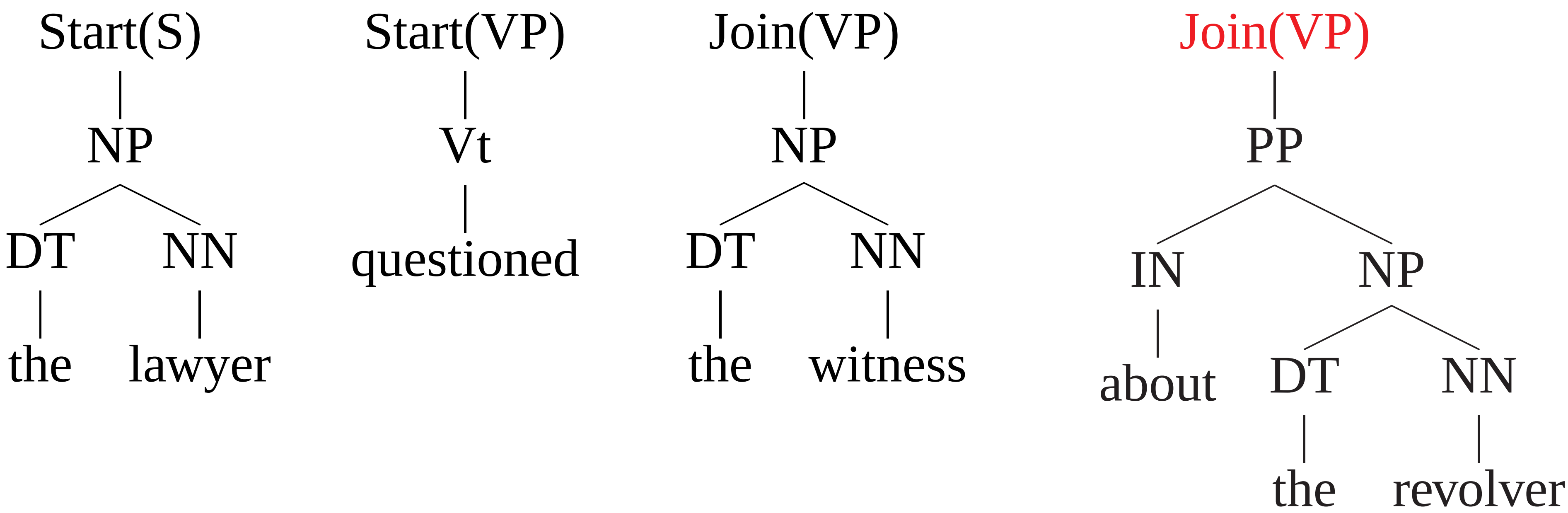

Start(S)
NP
DT
NN
the
lawyer
Start(VP)
Vt
questioned
Join(VP)
NP
DT
NN
the
witness
Start(PP)
IN
about
NP
DT
NN
the
revolver

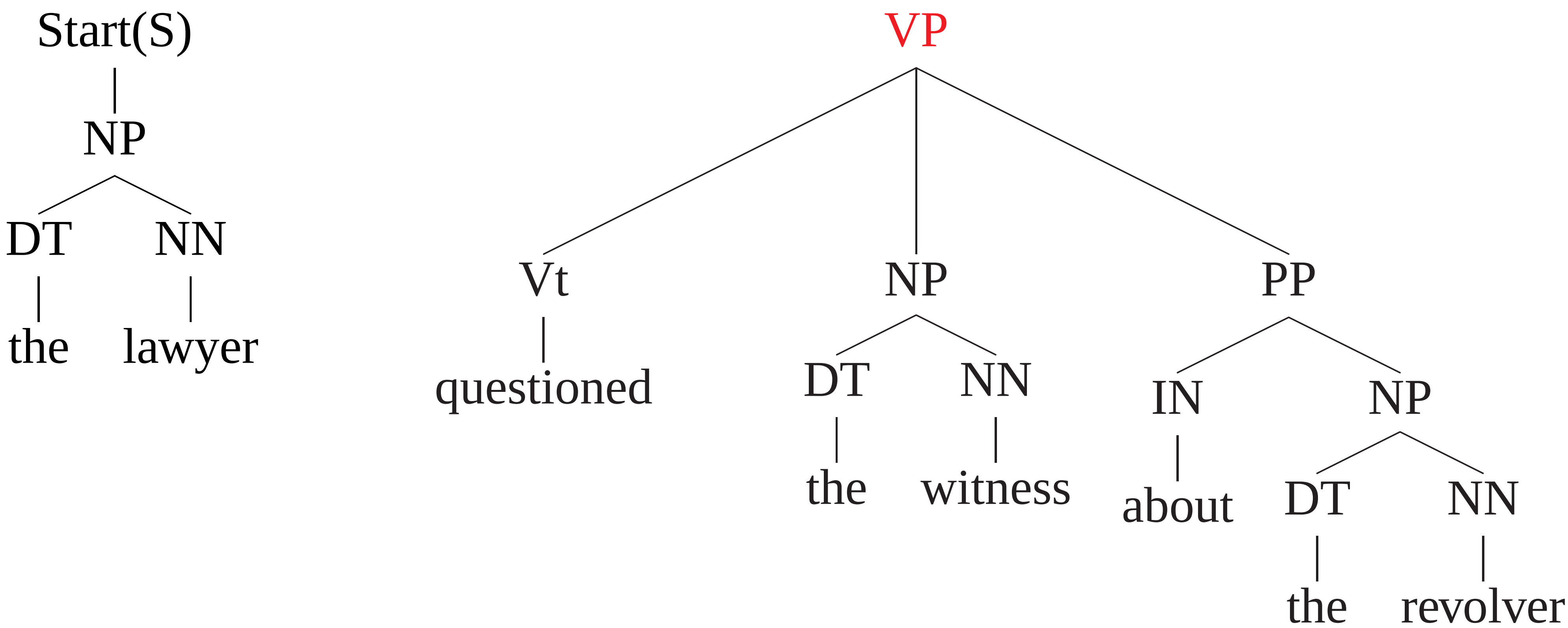
Check=NO


Start(S)
NP
DT
NN
the
lawyer
Start(VP)
Vt
questioned
Join(VP)
NP
DT
NN
the
witness
Start(PP)
IN
about
Join(PP)
NP
DT
NN
the
revolver

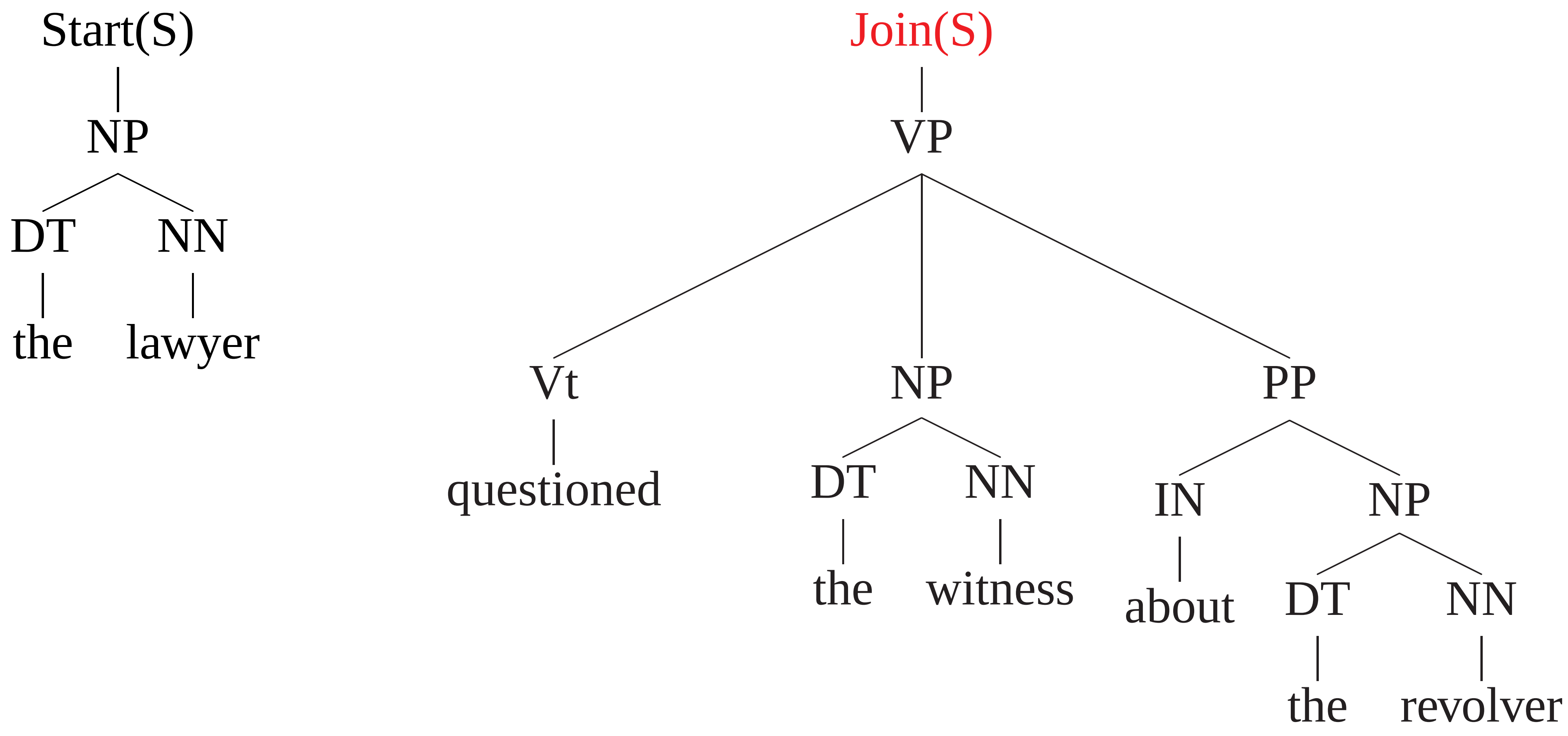

Start(S)
NP
DT
NN
the
lawyer
Start(VP)
Vt
questioned
Join(VP)
NP
DT
NN
the
witness
PP
IN
NP
about
DT
NN
the
revolver

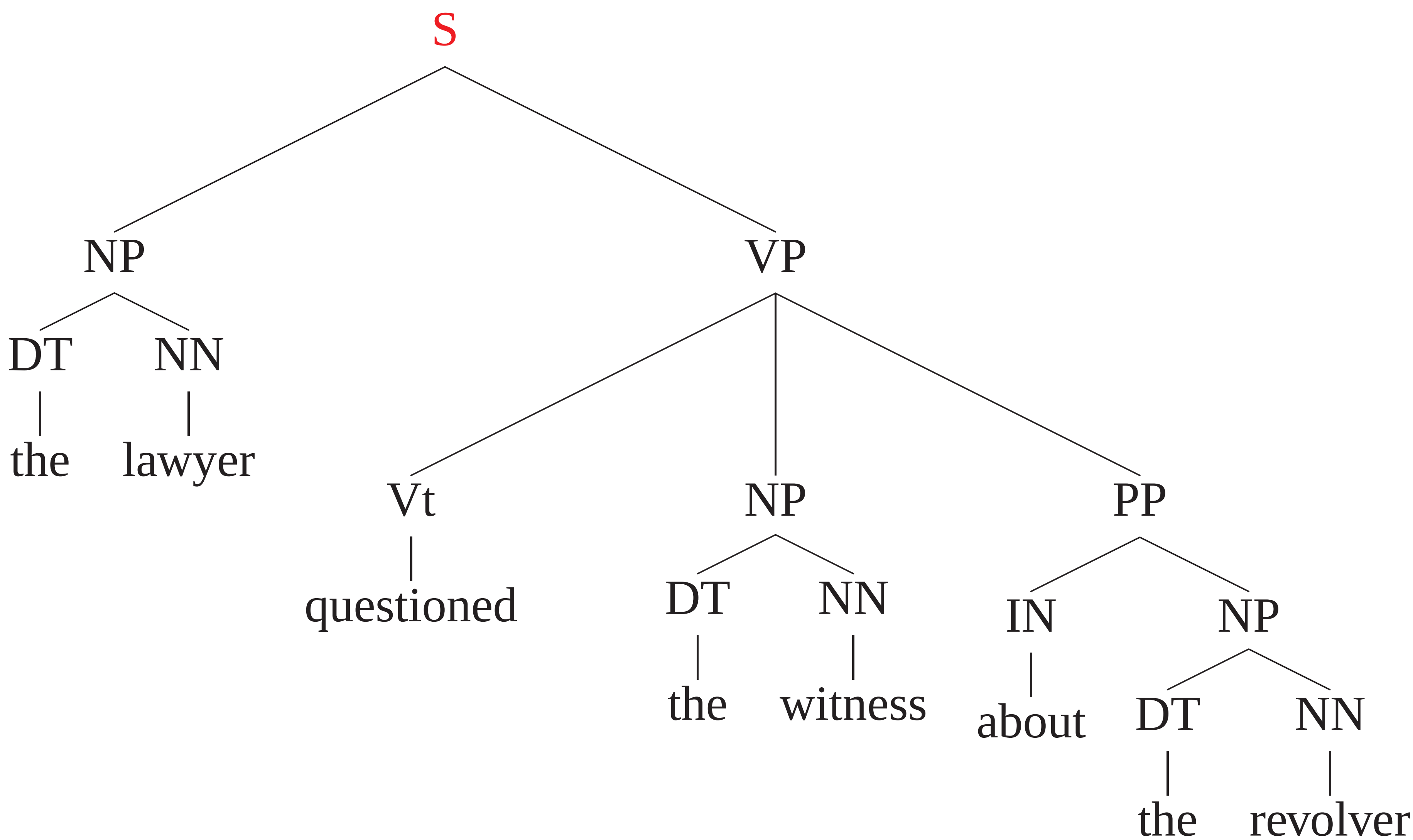
Check=YES


Start(S)
NP
DT
NN
the
lawyer
Start(VP)
Vt
questioned
Join(VP)
NP
DT
NN
the
witness
Join(VP)
PP
IN
NP
about
DT
NN
the
revolver


Start(S)
NP
DT
NN
the
lawyer
VP
Vt
NP
PP
questioned
DT
NN
IN
NP
the
witness
about
DT
NN
the
revolver

Check=YES


Start(S)
NP
DT
NN
the
lawyer
Join(S)
VP
Vt
NP
PP
questioned
DT
NN
the
witness
IN
NP
about
DT
NN
the
revolver


S
NP
VP
DT
NN
the
lawyer
Vt
NP
PP
questioned
DT
NN
IN
NP
the
witness
about
DT
NN
the
revolver

Check=YES

# The Final Sequence of decisions

$\langle d_1 \ldots d_{2n} \rangle = \langle$ DT, NN, Vt, DT, NN, IN, DT, NN,
Start(NP), Join(NP), Other, Start(NP), Join(NP),
Other, Start(NP), Join(NP),
Start(S), Check=NO, Start(VP), Check=NO,
Join(VP), Check=NO, Start(PP), Check=NO,
Join(PP), Check=YES, Join(VP), Check=YES,
Join(S), Check=YES $\rangle$

# A General Approach: (Conditional) History-Based Models

- Step 1: represent a tree as a sequence of **decisions** $d_1 \ldots d_m$

$$T = \langle d_1, d_2, \ldots d_m \rangle$$

  $m$ is **not** necessarily the length of the sentence

- Step 2: the probability of a tree is

$$P(T \mid S) = \prod_{i=1}^{m} P(d_i \mid d_1 \ldots d_{i-1}, S)$$

- Step 3: Use a log-linear model to estimate

$$P(d_i \mid d_1 \ldots d_{i-1}, S)$$

- Step 4: Search?? (answer we'll get to later: beam or heuristic search)

# Applying a Log-Linear Model

- Step 3: Use a log-linear model to estimate

$$P(d_i \mid d_1 \dots d_{i-1}, S)$$

- A reminder:

$$P(d_i \mid d_1 \dots d_{i-1}, S) = \frac{e^{\phi(\langle d_1 \dots d_{i-1}, S \rangle, d_i) \cdot \mathbf{W}}}{\sum_{d \in \mathcal{A}} e^{\phi(\langle d_1 \dots d_{i-1}, S \rangle, d) \cdot \mathbf{W}}}$$

where:

| | |
|---|---|
| $\langle d_1 \dots d_{i-1}, S \rangle$ | is the history |
| $d_i$ | is the outcome |
| $\phi$ | maps a history/outcome pair to a feature vector |
| $\mathbf{W}$ | is a parameter vector |
| $\mathcal{A}$ | is set of possible actions (may be context dependent) |

# Reminder: Implementing FEATURE_VECTOR

- Intermediate step: map history/tag pair to set of **feature strings**

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/Vt from which Spain expanded its empire into the rest of the Western Hemisphere .

e.g., Ratnaparkhi's features:

"TAG=Vt;Word=base"
"TAG=Vt;TAG-1=JJ"
"TAG=Vt;TAG-1=JJ;TAG-2=DT"
"TAG=Vt;SUFF1=e"
"TAG=Vt;SUFF2=se"
"TAG=Vt;SUFF3=ase"
"TAG=Vt;WORD-1=important"
"TAG=Vt;WORD+1=from"

# Reminder: Implementing FEATURE_VECTOR

- Next step: match strings to integers through a hash table

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/Vt from which Spain expanded its empire into the rest of the Western Hemisphere .

e.g., Ratnaparkhi's features:

| | |
|---|---|
| "TAG=Vt;Word=base" | $\rightarrow$ 1315 |
| "TAG=Vt;TAG-1=JJ" | $\rightarrow$ 17 |
| "TAG=Vt;TAG-1=JJ;TAG-2=DT" | $\rightarrow$ 32908 |
| "TAG=Vt;SUFF1=e" | $\rightarrow$ 459 |
| "TAG=Vt;SUFF2=se" | $\rightarrow$ 1000 |
| "TAG=Vt;SUFF3=ase" | $\rightarrow$ 1509 |
| "TAG=Vt;WORD-1=important" | $\rightarrow$ 1806 |
| "TAG=Vt;WORD+1=from" | $\rightarrow$ 300 |

In this case, sparse array is:
$A.length = 8, A(1...8) = \{1315, 17, 32908, 459, 1000, 1509, 1806, 300\}$

# Applying a Log-Linear Model

- Step 3: Use a log-linear model to estimate

$$P(d_i \mid d_1 \ldots d_{i-1}, S) = \frac{e^{\phi(\langle d_1 \ldots d_{i-1}, S\rangle, d_i) \cdot \mathbf{W}}}{\sum_{d \in \mathcal{A}} e^{\phi(\langle d_1 \ldots d_{i-1}, S\rangle, d) \cdot \mathbf{W}}}$$

- The big question: how do we define $\phi$?
- Ratnaparkhi's method defines $\phi$ differently depending on whether next decision is:
  - A tagging decision
    (same features as before for POS tagging!)
  - A chunking decision
  - A start/join decision after chunking
  - A check=no/check=yes decision

# Layer 2: Chunks

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Start(NP) | Join(NP) | Other | Start(NP) | Join(NP) | IN | DT | NN |
| DT | NN | Vt | DT | NN | about | the | revolver |
| the | lawyer | questioned | the | witness | | | |

⇒ "TAG=Join(NP);Word0=witness;POS0=NN"
"TAG=Join(NP);POS0=NN"
"TAG=Join(NP);Word+1=about;POS+1=IN"
"TAG=Join(NP);POS+1=IN"
"TAG=Join(NP);Word+2=the;POS+2=DT"
"TAG=Join(NP);POS+2=IN"
"TAG=Join(NP);Word-1=the;POS-1=DT;TAG-1=Start(NP)"
"TAG=Join(NP);POS-1=DT;TAG-1=Start(NP)"
"TAG=Join(NP);TAG-1=Start(NP)"
"TAG=Join(NP);Word-2=questioned;POS-2=Vt;TAG-2=Other"
. . .

# Layer 3: Join or Start

- Looks at head word, constituent (or POS) label, and start/join annotation of $n$'th tree relative to the decision, where $n = -2, -1$

- Looks at head word, constituent (or POS) label of $n$'th tree relative to the decision, where $n = 0, 1, 2$

- Looks at bigram features of the above for (-1,0) and (0,1)

- Looks at trigram features of the above for (-2,-1,0), (-1,0,1) and (0, 1, 2)

- The above features with all combinations of head words excluded

- Various punctuation features

# Layer 3: Check=NO or Check=YES

- A variety of questions concerning the proposed constituent

# The Search Problem

- In POS tagging, we could use the Viterbi algorithm because

$$P(T(j) \mid S, j, T(1) \ldots T(j{-}1)) = P(T(j) \mid S, j, T(j{-}2) \ldots T(j{-}1))$$

- Now: Decision $d_i$ could depend on arbitrary decisions in the "past" $\Rightarrow$ no chance for dynamic programming

- Instead, Ratnaparkhi uses a beam search method

# Overview

- Ratnaparkhi's Maximum-Entropy Parser

- The EM Algorithm Part I

# An Experiment/Some Intuition

- I have one coin in my pocket,

  Coin 0 has probability $\lambda$ of heads

- I toss the coin 10 times, and see the following sequence:

  HHTTHHHTHH

  (7 heads out of 10)

- What would you guess $\lambda$ to be?

# An Experiment/Some Intuition

- I have three coins in my pocket,

  Coin 0 has probability $\lambda$ of heads;
  Coin 1 has probability $p_1$ of heads;
  Coin 2 has probability $p_2$ of heads

- For each trial I do the following:

  First I toss Coin 0
  If Coin 0 turns up **heads**, I toss **coin 1** three times
  If Coin 0 turns up **tails**, I toss **coin 2** three times

  I don't tell you whether Coin 0 came up heads or tails,
  or whether Coin 1 or 2 was tossed three times,
  but I do tell you how many heads/tails are seen at each trial

- You see the following sequence:

$$\langle HHH \rangle, \langle TTT \rangle, \langle HHH \rangle, \langle TTT \rangle, \langle HHH \rangle$$

What would you estimate as the values for $\lambda$, $p_1$ and $p_2$?

# Maximum Likelihood Estimation

- We have data points $X_1, X_2, \ldots X_n$ drawn from some (finite or countable) set $\mathcal{X}$

- We have a parameter vector $\Theta$

- We have a parameter space $\Omega$

- We have a distribution $P(X \mid \Theta)$ for any $\Theta \in \Omega$, such that

$$\sum_{X \in \mathcal{X}} P(X \mid \Theta) = 1 \text{ and } P(X \mid \Theta) \geq 0 \text{ for all X}$$

- We assume that our data points $X_1, X_2, \ldots X_n$ are drawn at random (independently, identically distributed) from a distribution $P(X \mid \Theta^*)$ for some $\Theta^* \in \Omega$

# A First Example: Coin Tossing

- $\mathcal{X} = \{\mathsf{H}, \mathsf{T}\}$. Our data points $X_1, X_2, \ldots X_n$ are a sequence of heads and tails, e.g.

$$\mathsf{HHTTHHHTHH}$$

- Parameter vector $\Theta$ is a single parameter, i.e., the probability of coin coming up heads

- Parameter space $\Omega = [0, 1]$

- Distribution $P(X \mid \Theta)$ is defined as

$$P(X \mid \Theta) = \begin{cases} \Theta & \text{If } X = \mathsf{H} \\ 1 - \Theta & \text{If } X = \mathsf{T} \end{cases}$$

# Log-Likelihood

- We have data points $X_1, X_2, \ldots X_n$ drawn from some (finite or countable) set $\mathcal{X}$

- We have a parameter vector $\Theta$, and a parameter space $\Omega$

- We have a distribution $P(X \mid \Theta)$ for any $\Theta \in \Omega$

- The likelihood is

$$Likelihood(\Theta) = P(X_1, X_2, \ldots X_n \mid \Theta) = \prod_{i=1}^{n} P(X_i \mid \Theta)$$

- The log-likelihood is

$$L(\Theta) = \log Likelihood(\Theta) = \sum_{i=1}^{n} \log P(X_i \mid \Theta)$$

# Maximum Likelihood Estimation

- Given a sample $X_1, X_2, \ldots X_n$, choose

$$\Theta_{ML} = \text{argmax}_{\Theta \in \Omega} L(\Theta) = \text{argmax}_{\Theta \in \Omega} \sum_i \log P(X_i \mid \Theta)$$

- For example, take the coin example:
  say $X_1 \ldots X_n$ has $Count(H)$ heads, and $(n - Count(H))$ tails
  $\Rightarrow$

$$\begin{aligned} L(\Theta) &= \log\left(\Theta^{Count(H)} \times (1-\Theta)^{n-Count(H)}\right) \\ &= Count(H)\log\Theta + (n - Count(H))\log(1-\Theta) \end{aligned}$$

- We now have

$$\Theta_{ML} = \frac{Count(H)}{n}$$

# A Second Example: Probabilistic Context-Free Grammars

- $\mathcal{X}$ is the set of all parse trees generated by the underlying context-free grammar. Our sample is $n$ trees $T_1 \ldots T_n$ such that each $T_i \in \mathcal{X}$.

- $R$ is the set of rules in the context free grammar
  $N$ is the set of non-terminals in the grammar

- $\Theta_r$ for $r \in R$ is the parameter for rule $r$

- Let $R(\alpha) \subset R$ be the rules of the form $\alpha \rightarrow \beta$ for some $\beta$

- The parameter space $\Omega$ is the set of $\Theta \in [0,1]^{|R|}$ such that

$$\text{for all } \alpha \in N \sum_{r \in R(\alpha)} \Theta_r = 1$$

- We have

$$P(T \mid \Theta) = \prod_{r \in R} \Theta_r^{Count(T,r)}$$

where $Count(T, r)$ is the number of times rule $r$ is seen in the tree $T$

$$\Rightarrow \quad \log P(T \mid \Theta) = \sum_{r \in R} Count(T, r) \log \Theta_r$$

# Maximum Likelihood Estimation for PCFGs

- We have

$$\log P(T \mid \Theta) = \sum_{r \in R} Count(T, r) \log \Theta_r$$

where $Count(T, r)$ is the number of times rule $r$ is seen in the tree $T$

- And,

$$L(\Theta) = \sum_i \log P(T_i \mid \Theta) = \sum_i \sum_{r \in R} Count(T_i, r) \log \Theta_r$$

- Solving $\Theta_{ML} = \text{argmax}_{\Theta \in \Omega} L(\Theta)$ gives

$$\Theta_r = \frac{\sum_i Count(T_i, r)}{\sum_i \sum_{s \in R(\alpha)} Count(T_i, s)}$$

where $r$ is of the form $\alpha \rightarrow \beta$ for some $\beta$

# Models with Hidden Variables

- Now say we have two sets $\mathcal{X}$ and $\mathcal{Y}$, and a joint distribution $P(X, Y \mid \Theta)$

- If we had **fully observed data**, $(X_i, Y_i)$ pairs, then

$$L(\Theta) = \sum_i \log P(X_i, Y_i \mid \Theta)$$

- If we have **partially observed data**, $X_i$ examples, then

$$\begin{aligned} L(\Theta) &= \sum_i \log P(X_i \mid \Theta) \\ &= \sum_i \log \sum_{Y \in \mathcal{Y}} P(X_i, Y \mid \Theta) \end{aligned}$$

- The **EM (Expectation Maximization) algorithm** is a method for finding

$$\Theta_{ML} = \text{argmax}_{\Theta} \sum_i \log \sum_{Y \in \mathcal{Y}} P(X_i, Y \mid \Theta)$$

# The Three Coins Example

- e.g., in the three coins example:
  $\mathcal{Y} = \{\mathsf{H}, \mathsf{T}\}$
  $\mathcal{X} = \{\mathsf{HHH}, \mathsf{TTT}, \mathsf{HTT}, \mathsf{THH}, \mathsf{HHT}, \mathsf{TTH}, \mathsf{HTH}, \mathsf{THT}\}$
  $\Theta = \{\lambda, p_1, p_2\}$

- and

$$P(X, Y \mid \Theta) = P(Y \mid \Theta) P(X \mid Y, \Theta)$$

where

$$P(Y \mid \Theta) = \begin{cases} \lambda & \text{If } Y = \mathsf{H} \\ 1 - \lambda & \text{If } Y = \mathsf{T} \end{cases}$$

and

$$P(X \mid Y, \Theta) = \begin{cases} p_1^h (1 - p_1)^t & \text{If } Y = \mathsf{H} \\ p_2^h (1 - p_2) t & \text{If } Y = \mathsf{T} \end{cases}$$

where $h =$ number of heads in $X$, $t =$ number of tails in $X$

# The Three Coins Example

- Fully observed data might look like:

$$(\langle HHH \rangle, H), (\langle TTT \rangle, T), (\langle HHH \rangle, H), (\langle TTT \rangle, T), (\langle HHH \rangle, H)$$

- In this case maximum likelihood estimates are:

$$\lambda = \frac{3}{5}$$

$$p_1 = \frac{3}{3}$$

$$p_2 = \frac{0}{3}$$

# The Three Coins Example

- Partially observed data might look like:

$$\langle H H H \rangle, \langle T T T \rangle, \langle H H H \rangle, \langle T T T \rangle, \langle H H H \rangle$$

- How do we find the maximum likelihood parameters?

# The EM Algorithm

- $\Theta^t$ is the parameter vector at $t$'th iteration

- Choose $\Theta^0$ (at random, or using various heuristics)

- Iterative procedure is defined as

$$\Theta^t = \text{argmax}_\Theta Q(\Theta, \Theta^{t-1})$$

where

$$Q(\Theta, \Theta^{t-1}) = \sum_i \sum_{Y \in \mathcal{Y}} P(Y \mid X_i, \Theta^{t-1}) \log P(X_i, Y \mid \Theta)$$

# The EM Algorithm

- Iterative procedure is defined as $\Theta^t = \text{argmax}_\Theta Q(\Theta, \Theta^{t-1})$, where

$$Q(\Theta, \Theta^{t-1}) = \sum_i \sum_{Y \in \mathcal{Y}} P(Y \mid X_i, \Theta^{t-1}) \log P(X_i, Y \mid \Theta)$$

- Key points:
  - Intuition: fill in hidden variables $Y$ according to $P(Y \mid X_i, \Theta)$
  - EM is guaranteed to converge to a local maximum, or saddle-point, of the likelihood function
  - In general, if

$$\text{argmax}_\Theta \sum_i \log P(X_i, Y_i \mid \Theta)$$

has a simple (analytic) solution, then

$$\text{argmax}_\Theta \sum_i \sum_Y P(Y \mid X_i, \Theta) \log P(X_i, Y \mid \Theta)$$

also has a simple (analytic) solution.

# The Three Coins Example

- Partially observed data might look like:

$$\langle HHH \rangle, \langle TTT \rangle, \langle HHH \rangle, \langle TTT \rangle, \langle HHH \rangle$$

- Say $X = \langle HHH \rangle$, current parameters are $\lambda, p_1, p_2$

$$\begin{aligned} P(\langle HHH \rangle) &= P(\langle HHH \rangle, H) + P(\langle HHH \rangle, T) \\ &= \lambda p_1^3 + (1 - \lambda) p_2^3 \end{aligned}$$

and

$$\begin{aligned} P(Y = H \mid \langle HHH \rangle) &= \frac{P(\langle HHH \rangle, H)}{P(\langle HHH \rangle, H) + P(\langle HHH \rangle, T)} \\ &= \frac{\lambda p_1^3}{\lambda p_1^3 + (1 - \lambda) p_2^3} \end{aligned}$$

# The Three Coins Example

- After filling in hidden variables for each example, partially observed data might look like:

$$
\begin{array}{rl}
(\langle HHH\rangle, H) & P(Y = H \mid HHH) = 0.6 \\
(\langle HHH\rangle, T) & P(Y = T \mid HHH) = 0.4 \\
(\langle TTT\rangle, H) & P(Y = H \mid TTT) = 0.3 \\
(\langle TTT\rangle, T) & P(Y = T \mid TTT) = 0.7 \\
(\langle HHH\rangle, H) & P(Y = H \mid HHH) = 0.6 \\
(\langle HHH\rangle, T) & P(Y = T \mid HHH) = 0.4 \\
(\langle TTT\rangle, H) & P(Y = H \mid TTT) = 0.3 \\
(\langle TTT\rangle, T) & P(Y = T \mid TTT) = 0.7 \\
(\langle HHH\rangle, H) & P(Y = H \mid HHH) = 0.6 \\
(\langle HHH\rangle, T) & P(Y = T \mid HHH) = 0.4
\end{array}
$$

# EM for Probabilistic Context-Free Grammars

- A PCFG defines a distribution $P(S, T \mid \Theta)$ over tree/sentence pairs $(S, T)$

- If we had tree/sentence pairs (**fully observed data**) then

$$L(\Theta) = \sum_i \log P(S_i, T_i \mid \Theta)$$

- Say we have sentences only, $S_1 \ldots S_n$
$\Rightarrow$ trees are hidden variables

$$L(\Theta) = \sum_i \log \sum_T P(S_i, T \mid \Theta)$$

# EM for Probabilistic Context-Free Grammars

- Say we have sentences only, $S_1 \ldots S_n$
  $\Rightarrow$ trees are hidden variables

$$L(\Theta) = \sum_i \log \sum_T P(S_i, T \mid \Theta)$$

- EM algorithm is then $\Theta^t = \mathrm{argmax}_\Theta Q(\Theta, \Theta^{t-1})$, where

$$Q(\Theta, \Theta^{t-1}) = \sum_i \sum_T P(T \mid S_i, \Theta^{t-1}) \log P(S_i, T \mid \Theta)$$

- Remember:

$$\log P(S_i, T \mid \Theta) = \sum_{r \in R} Count(S_i, T, r) \log \Theta_r$$

where $Count(S, T, r)$ is the number of times rule $r$ is seen in the sentence/tree pair $(S, T)$

$$\begin{aligned}
\Rightarrow Q(\Theta, \Theta^{t-1}) &= \sum_i \sum_T P(T \mid S_i, \Theta^{t-1}) \log P(S_i, T \mid \Theta) \\
&= \sum_i \sum_T P(T \mid S_i, \Theta^{t-1}) \sum_{r \in R} Count(S_i, T, r) \log \Theta_r \\
&= \sum_i \sum_{r \in R} Count(S_i, r) \log \Theta_r
\end{aligned}$$

where $Count(S_i, r) = \sum_T P(T \mid S_i, \Theta^{t-1}) Count(S_i, T, r)$

**the expected counts**

- Solving $\Theta_{ML} = \text{argmax}_{\Theta \in \Omega} L(\Theta)$ gives

$$\Theta_r = \frac{\sum_i Count(S_i, r)}{\sum_i \sum_{s \in R(\alpha)} Count(S_i, s)}$$

where $r$ is of the form $\alpha \rightarrow \beta$ for some $\beta$

- We'll see next week that there are efficient (dynamic programming) algorithms for computation of

$$Count(S_i, r) = \sum_T P(T \mid S_i, \Theta^{t-1}) Count(S_i, T, r)$$