

# Tree Automata for Natural Language Processing

Kevin Knight

notes

CS562

fall, 2006

# Chomsky [1957]

- **Distinguish grammatical English from ungrammatical English:**

- John thinks Sara hit the boy
- \* The hit thinks Sara John boy
- John thinks the boy was hit by Sara
- Who does John think Sara hit?
- John thinks Sara hit the boy and the girl
- \* Who does John think Sara hit the boy and?
- John thinks Sara hit the boy with the bat
- What does John think Sara hit the boy with?
- Colorless green ideas sleep furiously.
- \* Green sleep furiously ideas colorless.

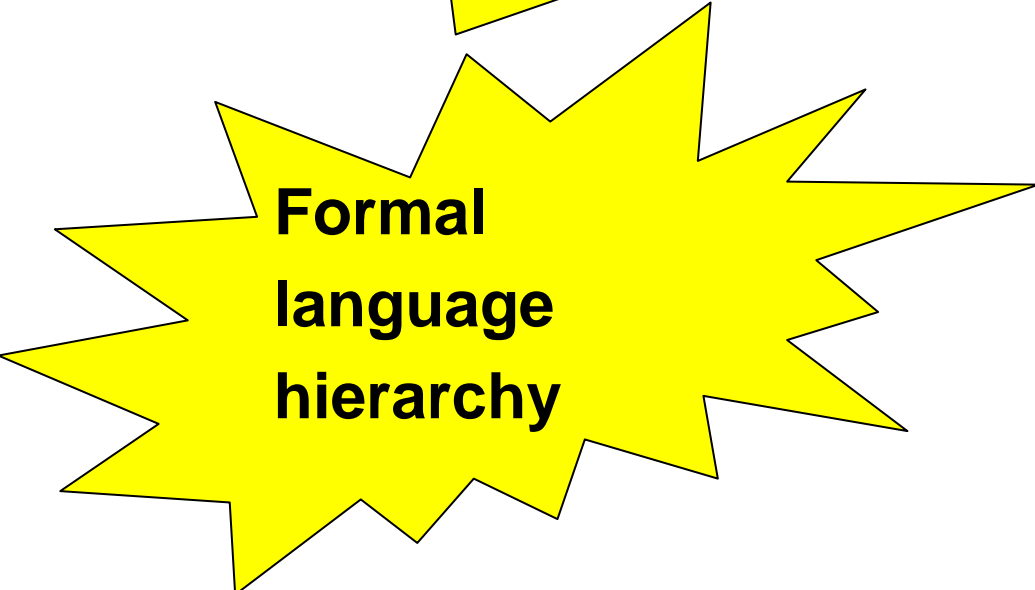
# Chomsky [1957]

- **Can be naturally extended to distinguish sensible vs. nonsense English:**
  - \* Colorless green ideas sleep furiously.
  - John eats bananas.
  - \* Bananas eat John.
- **Can be naturally extended to likelihood someone would say X:**
  - (VERY LIKELY) Hello, how are you?
  - (VERY VERY UNLIKELY) Sara the Sara boy hit hit India
  - (POSSIBLE) That coyote has long hard teeth
  - (VERY UNLIKELY) Garbage isn't in the mind of the ocean.

# This Research Program has Contributed Powerful Ideas!



**Context-free grammar**



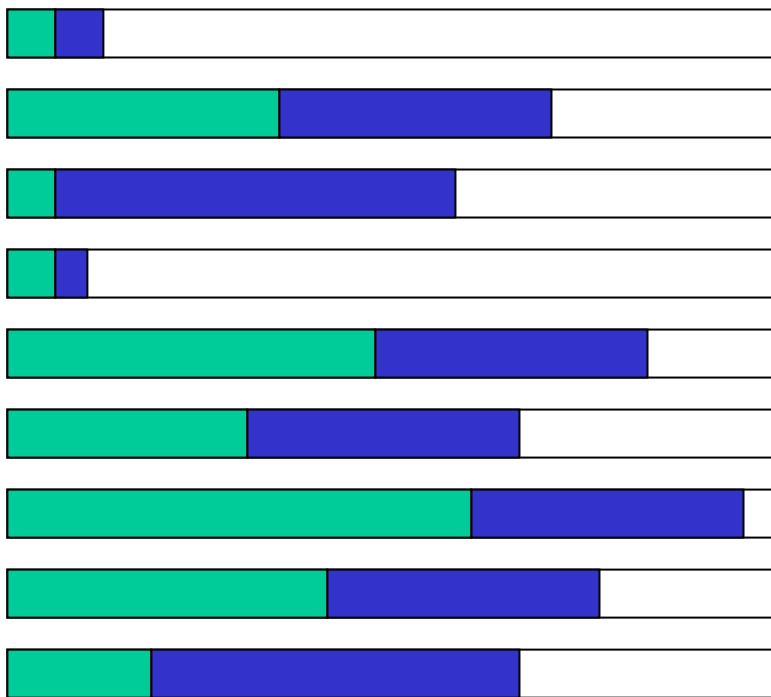
**Formal  
language  
hierarchy**



**Syntax,  
Phonology...**

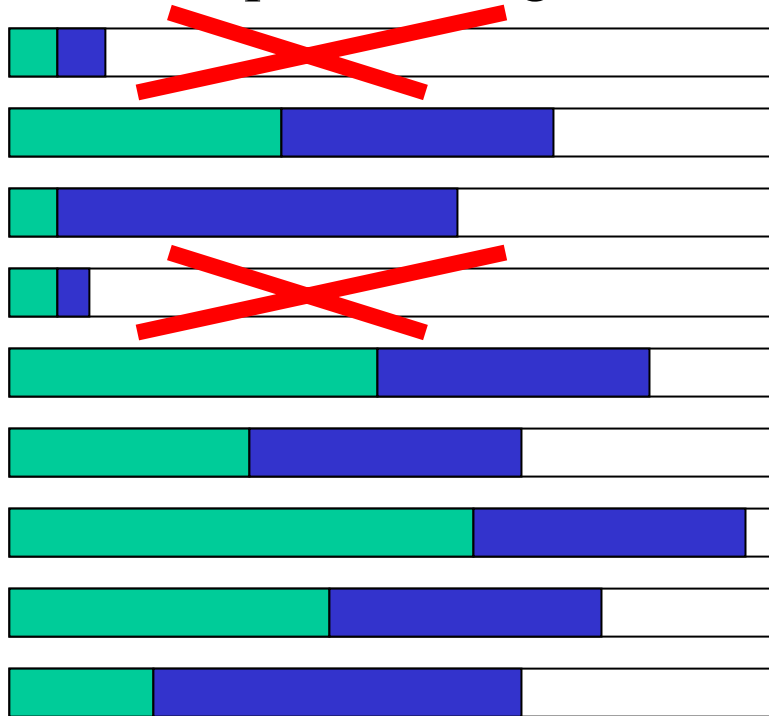
# This Research Program has NLP Technology Applications!

Alternative speech recognition or MT outputs:



# This Research Program has NLP Technology Applications!

Alternative speech recognition or MT outputs:



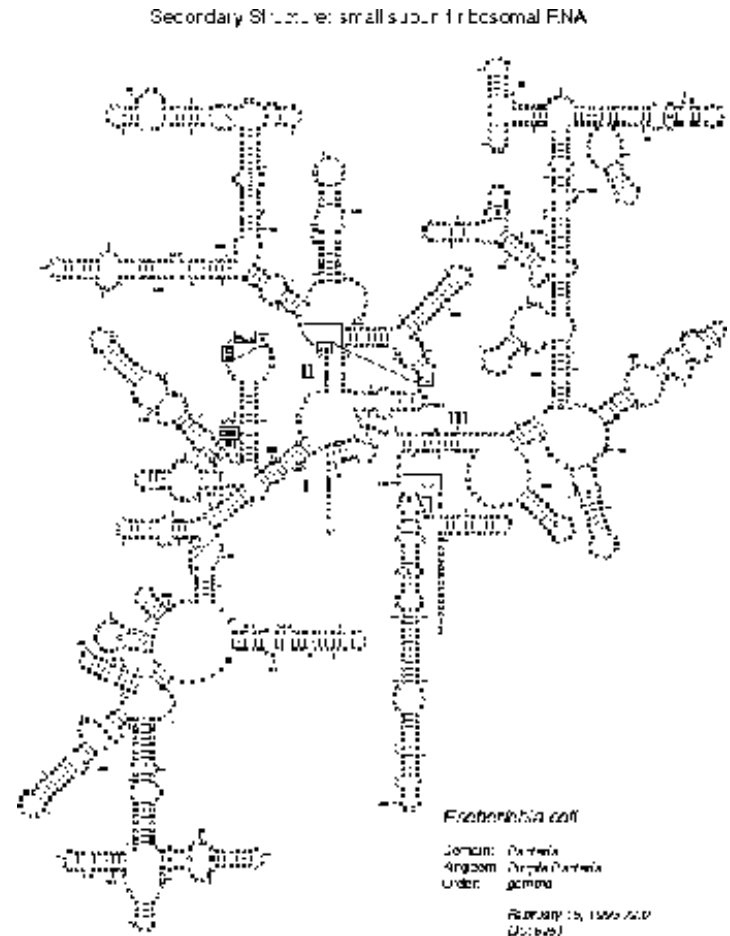
Pick this one!

# This Research Program Has Wider Reach than Was Expected!

- What makes a legal RNA sequence?
- What is the structure of a given RNA sequence?

For example:

Yasubumi Sakakibara, Michael Brown, Richard Hughey, I. Saira Mian, Kimmen Sjolander, Rebecca C. Underwood and David Haussler. **Stochastic Context-Free Grammars for tRNA**, *Modeling Nucleic Acids Research*, 22(23):5112-5120, 1994.



# This Research Program is Really Unfinished!

Type in your English sentence here:

Is this grammatical?

Is this sensible?



# This Research Program is Really Unfinished!

Type in your English sentence here:

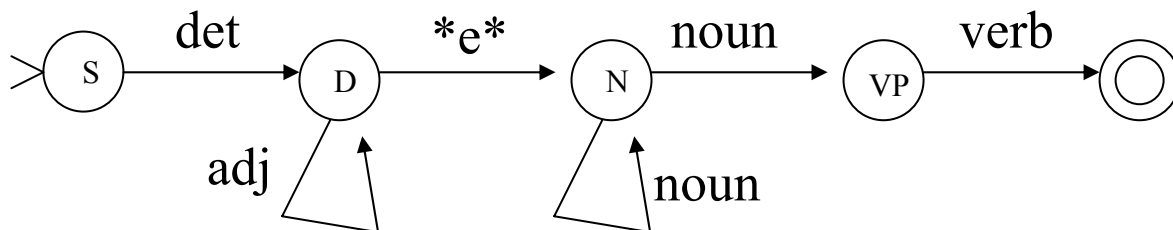
Is this grammatical?

Is this sensible?

Or, check out machine translation output  
→ it grammatical not is really!

# Regular Grammar

- Example
  - $S \rightarrow \text{det } D$
  - $D \rightarrow \text{adj } D$
  - $D \rightarrow N$
  - $N \rightarrow \text{noun } N$
  - $N \rightarrow \text{noun } VP$
  - $VP \rightarrow \text{verb}$
- Defines a set of *strings*
- Language described by regular grammar can also be described by a finite-state acceptor (FSA):



# Regular Grammar

- Can't capture long-distance relationships with a reasonable-sized grammar
  - The *toys* in that closet *are/is* fun to play with.
  - The *toys* in that closet are fun to *play* with.
- Can't handle  $a^n b^n$  constructions?
  - John, Bill, ..., and Jack enjoy  
swimming, kayaking, ..., and slacking, respectively.
- Can't handle deep recursion?
  - The rat died.
  - The rat [that the cat ate] died.

^can a verb come next? yes...

# Context-Free Grammar

- Example:

- $S \rightarrow NP VP$  [p=1.0]
- $NP \rightarrow DET N$  [p=0.7]
- $NP \rightarrow NP PP$  [p=0.3]
- $PP \rightarrow P NP$  [p=1.0]
- $VP \rightarrow V NP$  [p=0.4]
- $DET \rightarrow the$  [p=1.0]
- $N \rightarrow boy$  [p=1.0]
- $V \rightarrow saw$  [p=1.0]
- $P \rightarrow with$  [p=1.0]

## Generative Process:

S

- Defines a set of *strings*
- Language described by CFG can also be described by a push-down acceptor

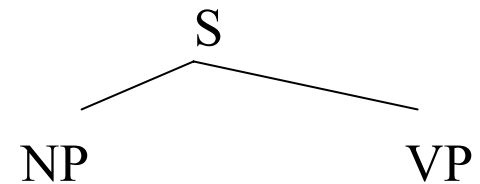
# Context-Free Grammar

- Example:

- $S \rightarrow NP VP$  [p=1.0]
- $NP \rightarrow DET N$  [p=0.7]
- $NP \rightarrow NP PP$  [p=0.3]
- $PP \rightarrow P NP$  [p=1.0]
- $VP \rightarrow V NP$  [p=0.4]
- $DET \rightarrow the$  [p=1.0]
- $N \rightarrow boy$  [p=1.0]
- $V \rightarrow saw$  [p=1.0]
- $P \rightarrow with$  [p=1.0]

- Defines a set of *strings*
- Language described by CFG can also be described by a push-down acceptor

## Generative Process:



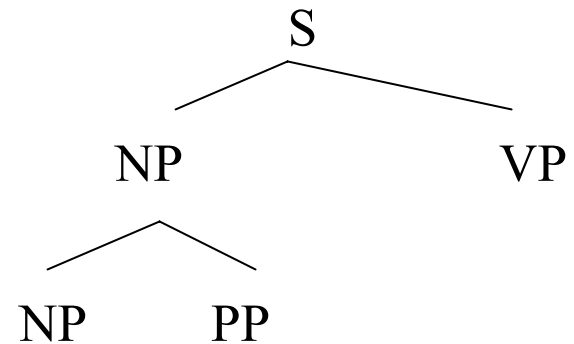
# Context-Free Grammar

- Example:

- $S \rightarrow NP VP$  [p=1.0]
- $NP \rightarrow DET N$  [p=0.7]
- $NP \rightarrow NP PP$  [p=0.3]
- $PP \rightarrow P NP$  [p=1.0]
- $VP \rightarrow V NP$  [p=0.4]
- $DET \rightarrow the$  [p=1.0]
- $N \rightarrow boy$  [p=1.0]
- $V \rightarrow saw$  [p=1.0]
- $P \rightarrow with$  [p=1.0]

- Defines a set of *strings*
- Language described by CFG can also be described by a push-down acceptor

## Generative Process:



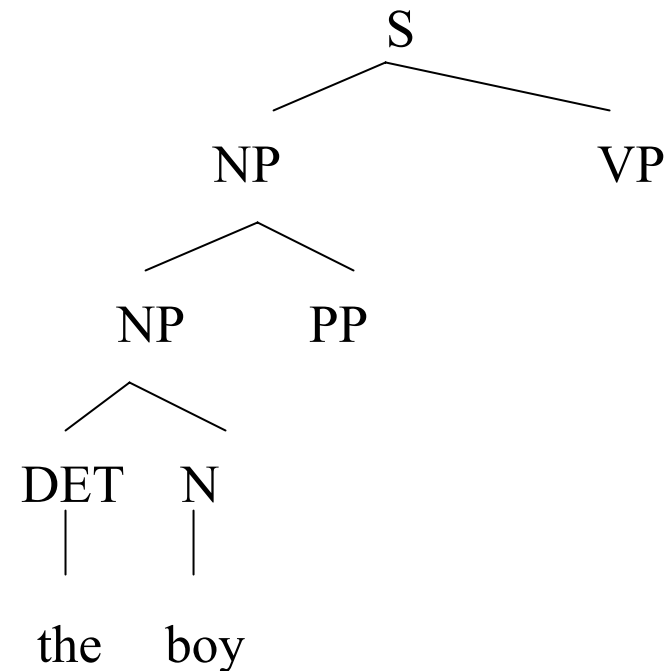
# Context-Free Grammar

- Example:

- $S \rightarrow NP VP$  [p=1.0]
- $NP \rightarrow DET N$  [p=0.7]
- $NP \rightarrow NP PP$  [p=0.3]
- $PP \rightarrow P NP$  [p=1.0]
- $VP \rightarrow V NP$  [p=0.4]
- $DET \rightarrow the$  [p=1.0]
- $N \rightarrow boy$  [p=1.0]
- $V \rightarrow saw$  [p=1.0]
- $P \rightarrow with$  [p=1.0]

- Defines a set of *strings*
- Language described by CFG can also be described by a push-down acceptor

## Generative Process:



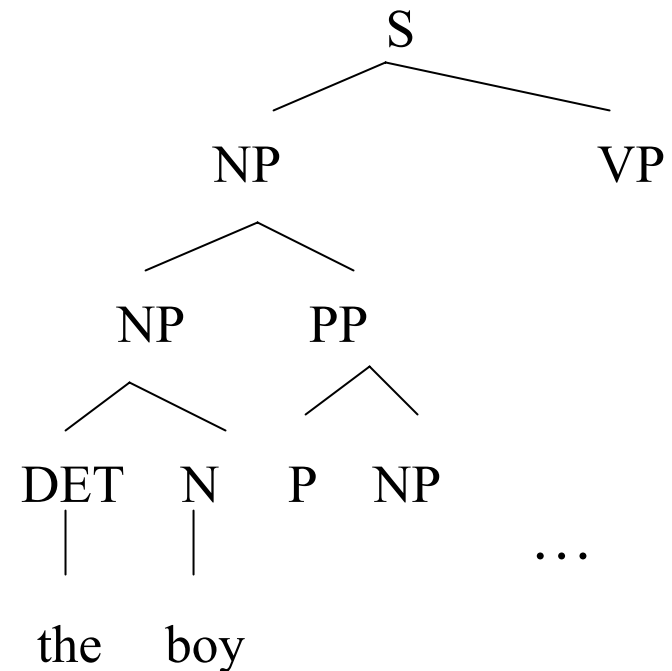
# Context-Free Grammar

- Example:

- $S \rightarrow NP VP$  [p=1.0]
- $NP \rightarrow DET N$  [p=0.7]
- $NP \rightarrow NP PP$  [p=0.3]
- $PP \rightarrow P NP$  [p=1.0]
- $VP \rightarrow V NP$  [p=0.4]
- $DET \rightarrow the$  [p=1.0]
- $N \rightarrow boy$  [p=1.0]
- $V \rightarrow saw$  [p=1.0]
- $P \rightarrow with$  [p=1.0]

- Defines a set of *strings*
- Language described by CFG can also be described by a push-down acceptor

## Generative Process:





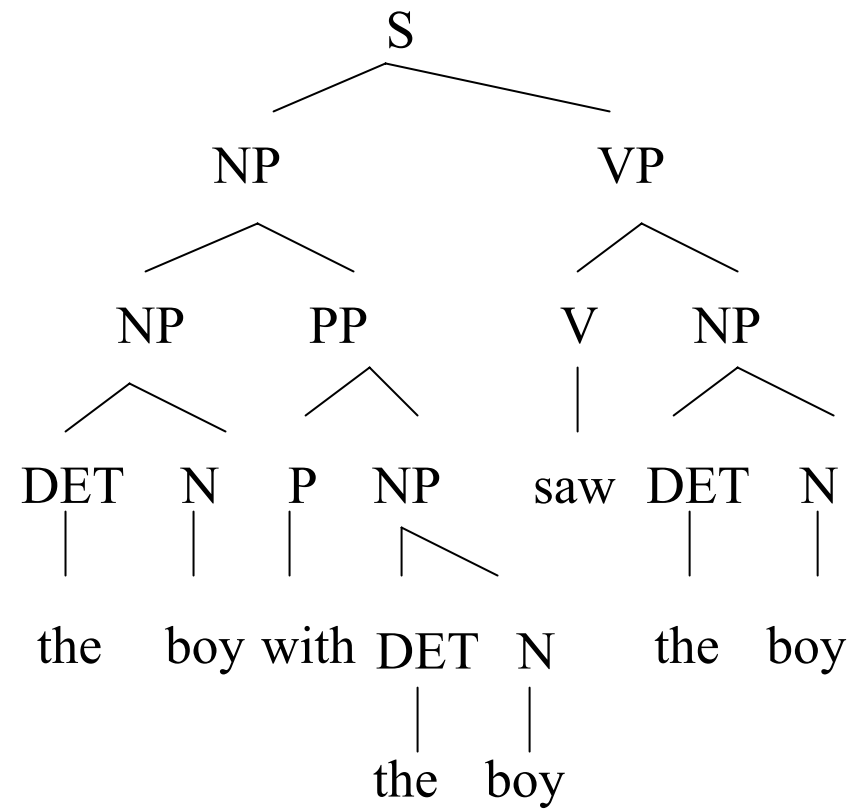
# Context-Free Grammar

- Example:

- $S \rightarrow NP VP$  [p=1.0]
- $NP \rightarrow DET N$  [p=0.7]
- $NP \rightarrow NP PP$  [p=0.3]
- $PP \rightarrow P NP$  [p=1.0]
- $VP \rightarrow V NP$  [p=0.4]
- $DET \rightarrow the$  [p=1.0]
- $N \rightarrow boy$  [p=1.0]
- $V \rightarrow saw$  [p=1.0]
- $P \rightarrow with$  [p=1.0]

- Defines a set of *strings*
- Language described by CFG can also be described by a push-down acceptor

## Generative Process:



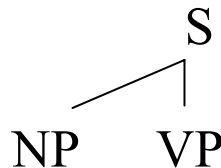
# Context-Free Grammar

- Is this what lies behind modern parsers like Collins and Charniak?
- No... they do not have a finite list of productions, but rather infinite.
  - “Markovized” grammar
- Generative process is head-out:

$$\begin{array}{c} S \\ | \\ VP \end{array}$$
$$P_h(VP \mid S)$$

# Context-Free Grammar

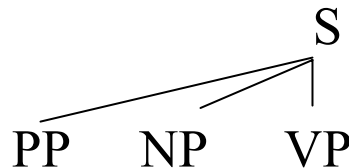
- Is this what lies behind modern parsers like Collins and Charniak?
- No... they do not have a finite list of productions, but rather infinite.
  - “Markovized” grammar
- Generative process is head-out:



$$P_{\text{left}}(\text{NP} \mid \text{VP}, \text{S})$$

# Context-Free Grammar

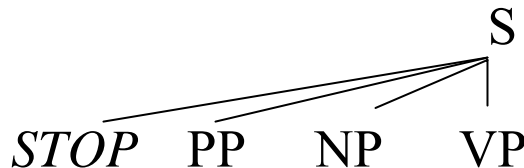
- Is this what lies behind modern parsers like Collins and Charniak?
- No... they do not have a finite list of productions, but rather infinite.
  - “Markovized” grammar
- Generative process is head-out:



$$P_{\text{left}}(\text{PP} \mid \text{VP}, \text{S})$$

# Context-Free Grammar

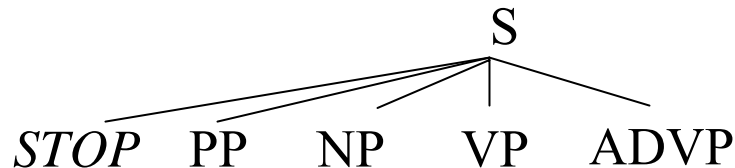
- Is this what lies behind modern parsers like Collins and Charniak?
- No... they do not have a finite list of productions, but rather infinite.
  - “Markovized” grammar
- Generative process is head-out:



$$P_{\text{left}}(STOP \mid VP, S)$$

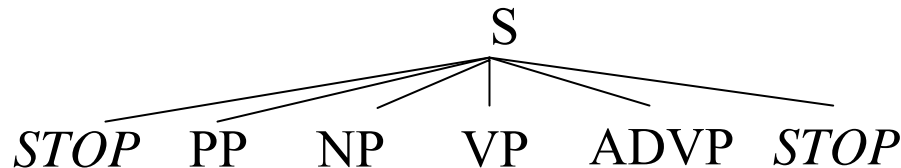
# Context-Free Grammar

- Is this what lies behind modern parsers like Collins and Charniak?
- No... they do not have a finite list of productions, but rather infinite.
  - “Markovized” grammar
- Generative process is head-out:



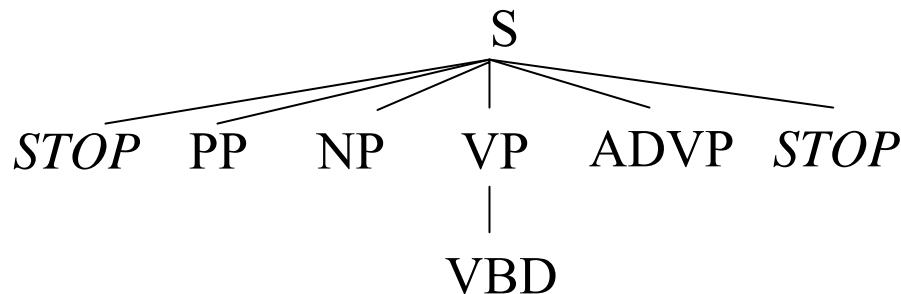
# Context-Free Grammar

- Is this what lies behind modern parsers like Collins and Charniak?
- No... they do not have a finite list of productions, but rather infinite.
  - “Markovized” grammar
- Generative process is head-out:



# Context-Free Grammar

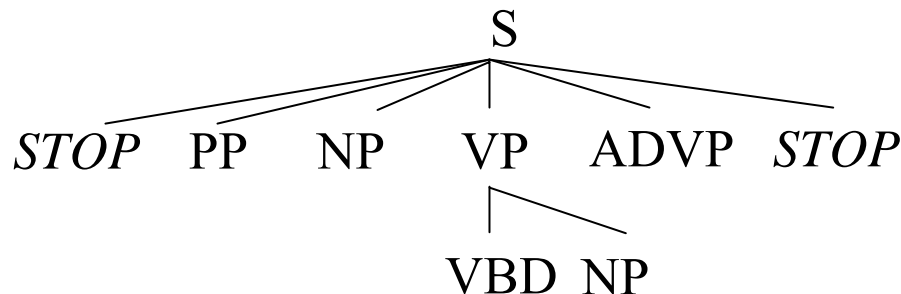
- Is this what lies behind modern parsers like Collins and Charniak?
- No... they do not have a finite list of productions, but rather infinite.
  - “Markovized” grammar
- Generative process is head-out:





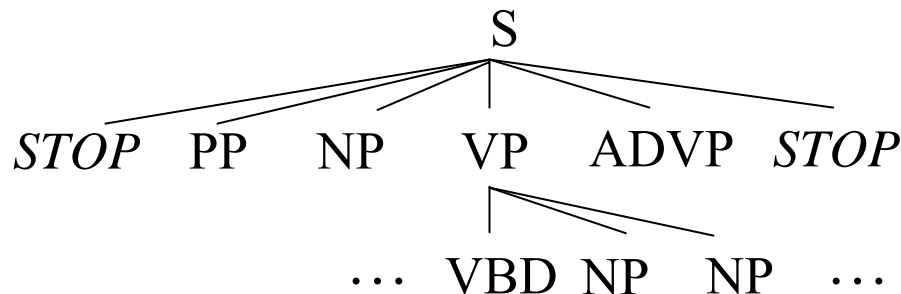
# Context-Free Grammar

- Is this what lies behind modern parsers like Collins and Charniak?
- No... they do not have a finite list of productions, but rather infinite.
  - “Markovized” grammar
- Generative process is head-out:



# Context-Free Grammar

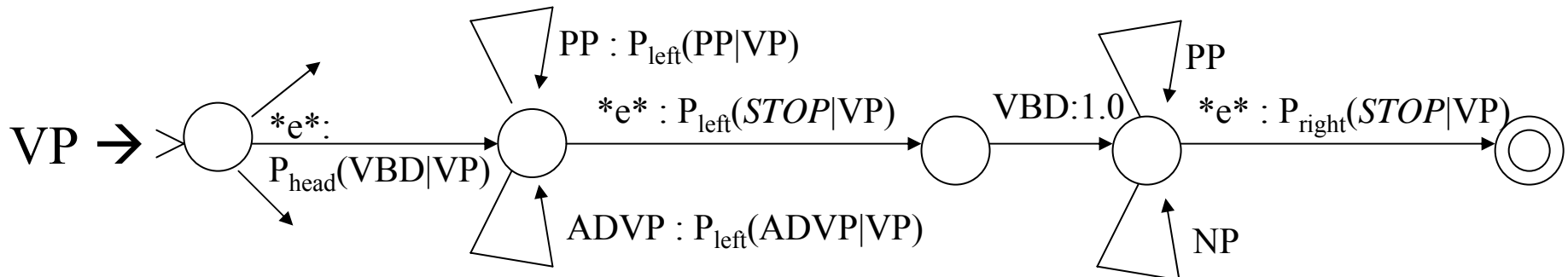
- Is this what lies behind modern parsers like Collins and Charniak?
- No... they do not have a finite list of productions, but rather infinite.
  - “Markovized” grammar
- Generative process is head-out:



# Extended Context-Free Grammar

## [Thatcher, 1967]

- Example:
  - $S \rightarrow \text{ADV}^* \text{NP VP PP}^*$
  - $\text{VP} \rightarrow \text{VBD NP} [\text{NP}] \text{PP}^*$
- Defines a set of *strings*
- Can model Charniak and Collins:
  - $\text{VP} \rightarrow (\text{ADV} \mid \text{PP})^* \text{VBD} (\text{NP} \mid \text{PP})^*$
- Can even model probabilistic version:



# Extended Context-Free Grammar

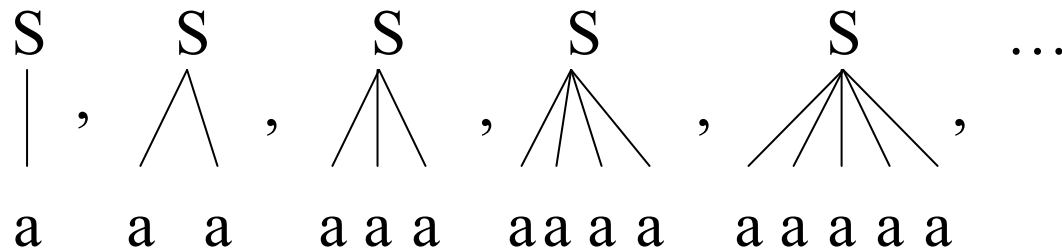
## [Thatcher, 1967]

- Is ECFG more powerful than CFG?
- It is possible to build a CFG that generates the same set of strings as your ECFG
  - As long as right-hand side of every rule is regular
  - Or even context-free ☺
- BUT:
  - the CFG won't generate the same *derivation trees* as the ECFG

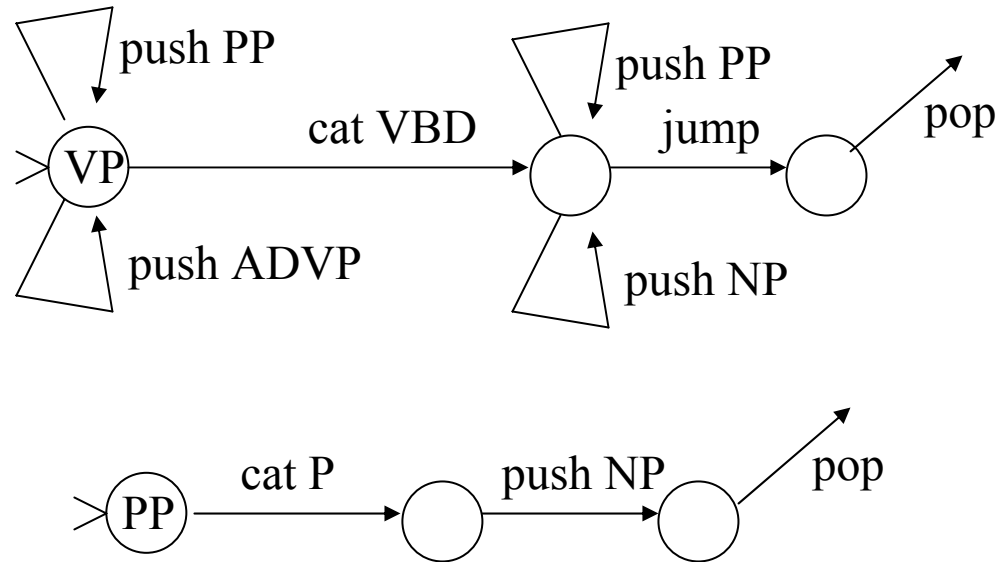
# Extended Context-Free Grammar

## [Thatcher, 1967]

- For example:
  - ECFG can (of course) accept the string language  $a^*$
  - Therefore, we can build a CFG that also accepts  $a^*$
  - But ECFG can do it via this set of derivation trees, if so desired:



# Recursive Transition Networks (RTNs)



Collection of separate networks.

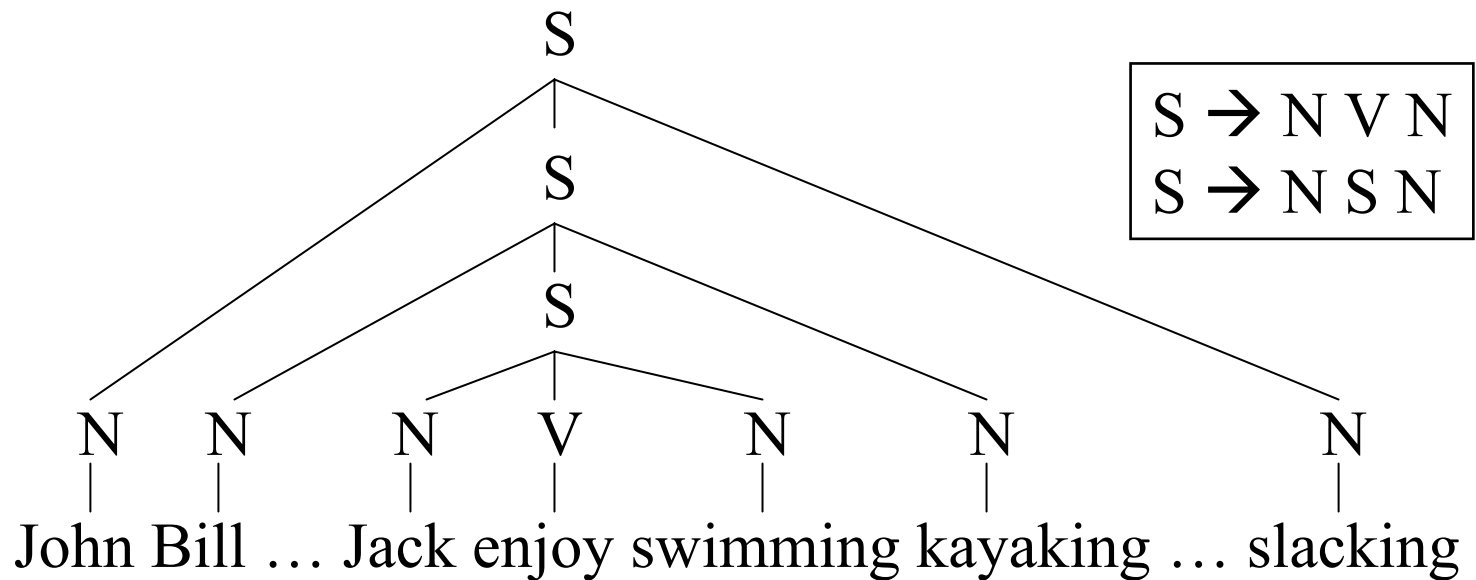
Defines a set of *strings*.

Usually no one thinks about derivation trees.

Augmented RTNs (ATNs) store pieces of text in register variables as recognition proceeds.

# More Trees vs Strings

- Regular grammars can't handle  $a^n b^n$  constructions?
- CFGs can:



- We can generate the string ...
- But that's not necessarily the derivation tree we want

# Tree Generating Systems

- What if the trees are important?
  - Parsing, RNA structure prediction
  - Tree transformation (active  $\rightarrow$  passive, MT, SBLM, ...)
- We can view CFG or ECFG as a tree-generating system, if we squint real hard ...
- Or, we can look at real tree-generating systems:
  - Regular tree grammar (RTG)
    - More recently in our field called tree substitution grammar (TSG)
  - Tree-adjoining grammar (TAG)
- And at tree-transforming systems:
  - Top-down tree transducers (R, RL, RLN, ...)
  - Bottom-up tree transducers (F, FL, FLN, ...)



# What We'd Like to Do

- Device or grammar  $D$  for compactly representing possibly-infinite set of trees (possibly with weights)
- Want to support operations like:
  - Membership testing: Is tree  $X$  in the set  $D$ ?
  - Equality testing: Do  $D1$  and  $D2$  contain exactly the same trees?
  - Intersection: Compute (possibly-infinite) set of trees in both  $D1$  and  $D2$
  - Weighted intersection
    - Re-score all weighted trees in  $D1$  by intersecting with weighted set  $D2$
    - E.g.:  $D1$  is a set of candidate translations
    - E.g.:  $D2$  is a language model [Chomsky 1957]

# Regular Tree Grammar (RTG)

## Generative Process:

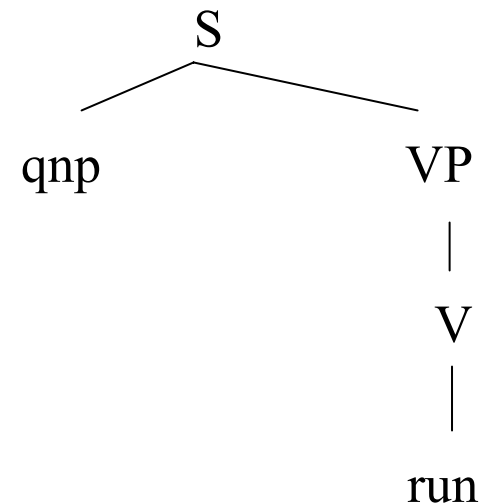
- Example:
  - $q \rightarrow S(qnp, VP(V(run)))$  [p=1.0]
  - $qnp \rightarrow NP(qdet, qn)$  [p=0.7]
  - $qnp \rightarrow NP(qnp, qpp)$  [p=0.3]
  - $qpp \rightarrow PP(qprep, qnp)$  [p=1.0]
  - $qdet \rightarrow DET(the)$  [p=1.0]
  - $qprep \rightarrow PREP(of)$  [p=1.0]
  - $qn \rightarrow N(sons)$  [p=0.5]
  - $qn \rightarrow N(daughters)$  [p=0.5]
- Defines a set of *trees*

q

# Regular Tree Grammar (RTG)

- Example:
  - $q \rightarrow S(qnp, VP(V(run)))$  [p=1.0]
  - $qnp \rightarrow NP(qdet, qn)$  [p=0.7]
  - $qnp \rightarrow NP(qnp, qpp)$  [p=0.3]
  - $qpp \rightarrow PP(qprep, qnp)$  [p=1.0]
  - $qdet \rightarrow DET(the)$  [p=1.0]
  - $qprep \rightarrow PREP(of)$  [p=1.0]
  - $qn \rightarrow N(sons)$  [p=0.5]
  - $qn \rightarrow N(daughters)$  [p=0.5]

Generative Process:

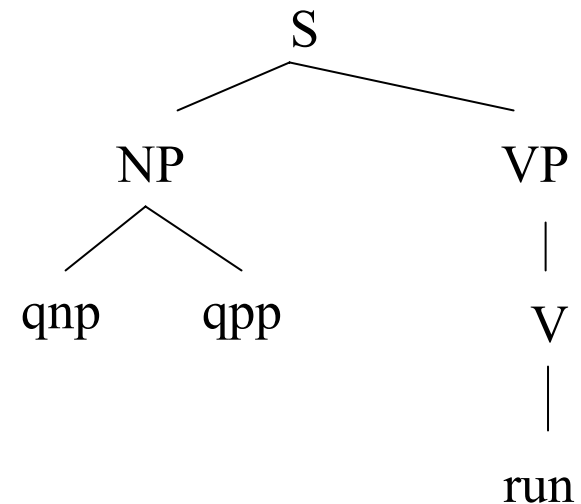


- Defines a set of *trees*

# Regular Tree Grammar (RTG)

- Example:
  - $q \rightarrow S(qnp, VP(V(run)))$  [p=1.0]
  - $qnp \rightarrow NP(qdet, qn)$  [p=0.7]
  - $qnp \rightarrow NP(qnp, qpp)$  [p=0.3]
  - $qpp \rightarrow PP(qprep, qnp)$  [p=1.0]
  - $qdet \rightarrow DET(the)$  [p=1.0]
  - $qprep \rightarrow PREP(of)$  [p=1.0]
  - $qn \rightarrow N(sons)$  [p=0.5]
  - $qn \rightarrow N(daughters)$  [p=0.5]

Generative Process:



- Defines a set of *trees*

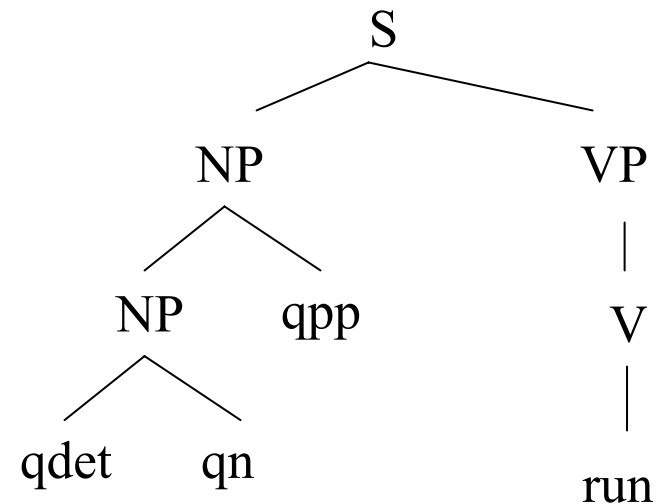
# Regular Tree Grammar (RTG)

- Example:

- $q \rightarrow S(qnp, VP(V(run)))$  [p=1.0]
- $qnp \rightarrow NP(qdet, qn)$  [p=0.7]
- $qnp \rightarrow NP(qnp, qpp)$  [p=0.3]
- $qpp \rightarrow PP(qprep, qnp)$  [p=1.0]
- $qdet \rightarrow DET(the)$  [p=1.0]
- $qprep \rightarrow PREP(of)$  [p=1.0]
- $qn \rightarrow N(sons)$  [p=0.5]
- $qn \rightarrow N(daughters)$  [p=0.5]

- Defines a set of *trees*

## Generative Process:



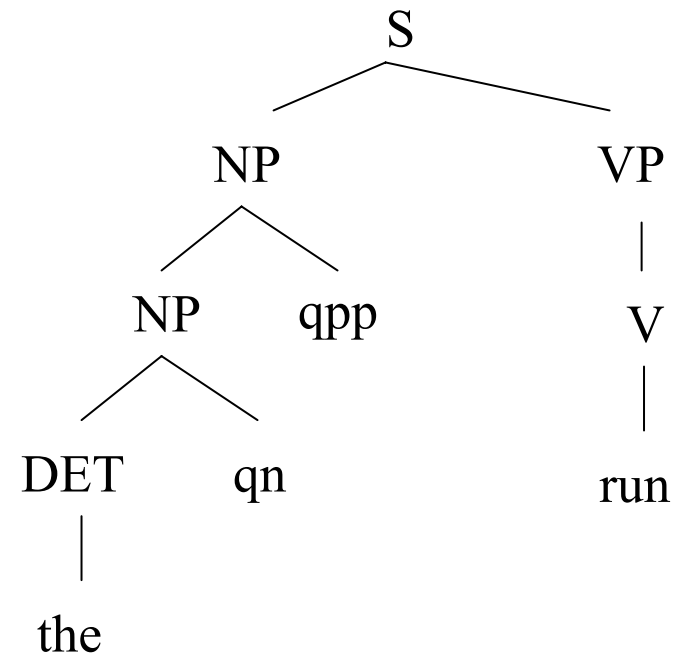
# Regular Tree Grammar (RTG)

- Example:

- $q \rightarrow S(qnp, VP(V(run)))$  [p=1.0]
- $qnp \rightarrow NP(qdet, qn)$  [p=0.7]
- $qnp \rightarrow NP(qnp, qpp)$  [p=0.3]
- $qpp \rightarrow PP(qprep, qnp)$  [p=1.0]
- $qdet \rightarrow DET(the)$  [p=1.0]
- $qprep \rightarrow PREP(of)$  [p=1.0]
- $qn \rightarrow N(sons)$  [p=0.5]
- $qn \rightarrow N(daughters)$  [p=0.5]

- Defines a set of *trees*

## Generative Process:



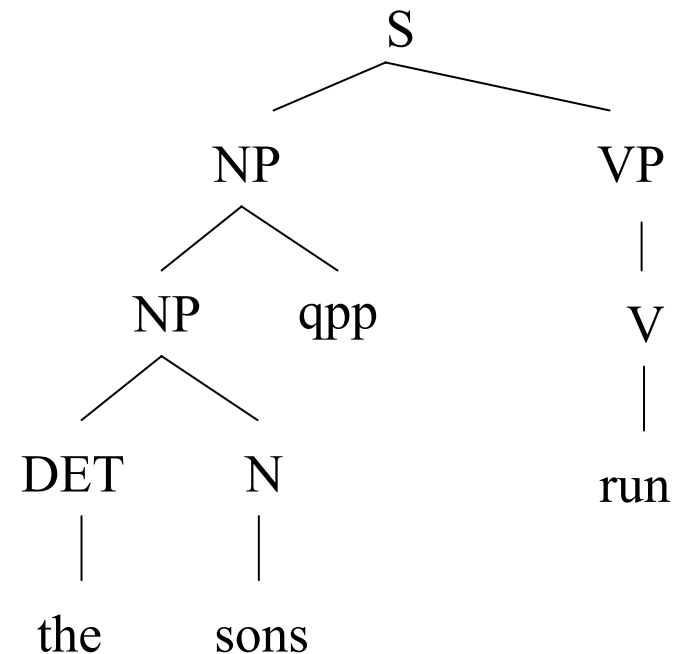
# Regular Tree Grammar (RTG)

- Example:

- $q \rightarrow S(qnp, VP(V(run)))$  [p=1.0]
- $qnp \rightarrow NP(qdet, qn)$  [p=0.7]
- $qnp \rightarrow NP(qnp, qpp)$  [p=0.3]
- $qpp \rightarrow PP(qprep, qnp)$  [p=1.0]
- $qdet \rightarrow DET(the)$  [p=1.0]
- $qprep \rightarrow PREP(of)$  [p=1.0]
- $qn \rightarrow N(sons)$  [p=0.5]
- $qn \rightarrow N(daughters)$  [p=0.5]

- Defines a set of *trees*

## Generative Process:



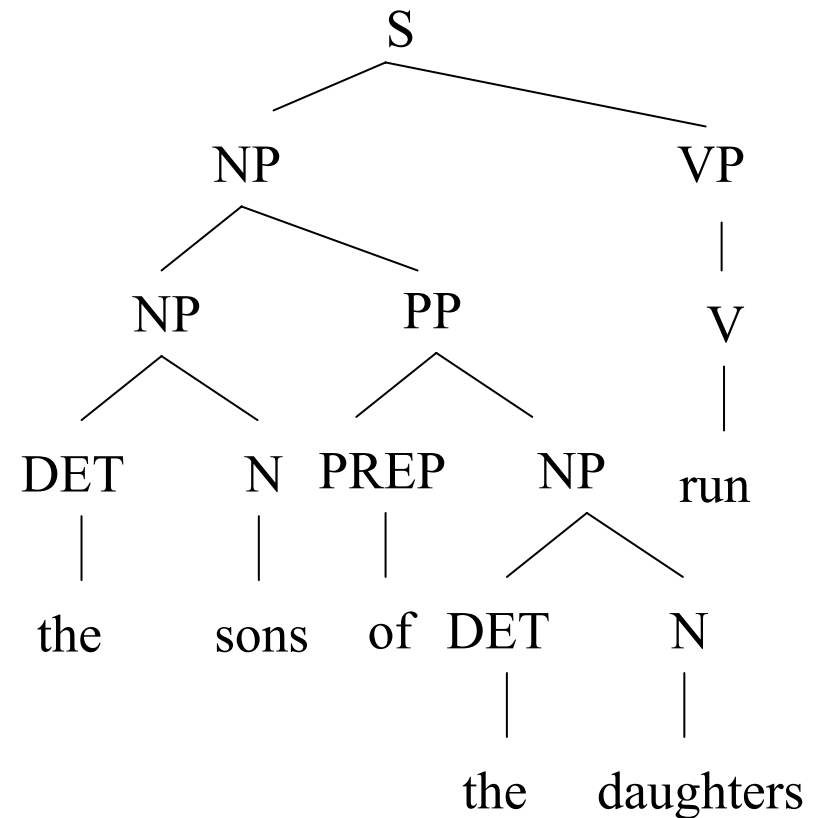
# Regular Tree Grammar (RTG)

- Example:

- $q \rightarrow S(qnp, VP(V(run)))$  [p=1.0]
- $qnp \rightarrow NP(qdet, qn)$  [p=0.7]
- $qnp \rightarrow NP(qnp, qpp)$  [p=0.3]
- $qpp \rightarrow PP(qprep, qnp)$  [p=1.0]
- $qdet \rightarrow DET(the)$  [p=1.0]
- $qprep \rightarrow PREP(of)$  [p=1.0]
- $qn \rightarrow N(sons)$  [p=0.5]
- $qn \rightarrow N(daughters)$  [p=0.5]

- Defines a set of *trees*

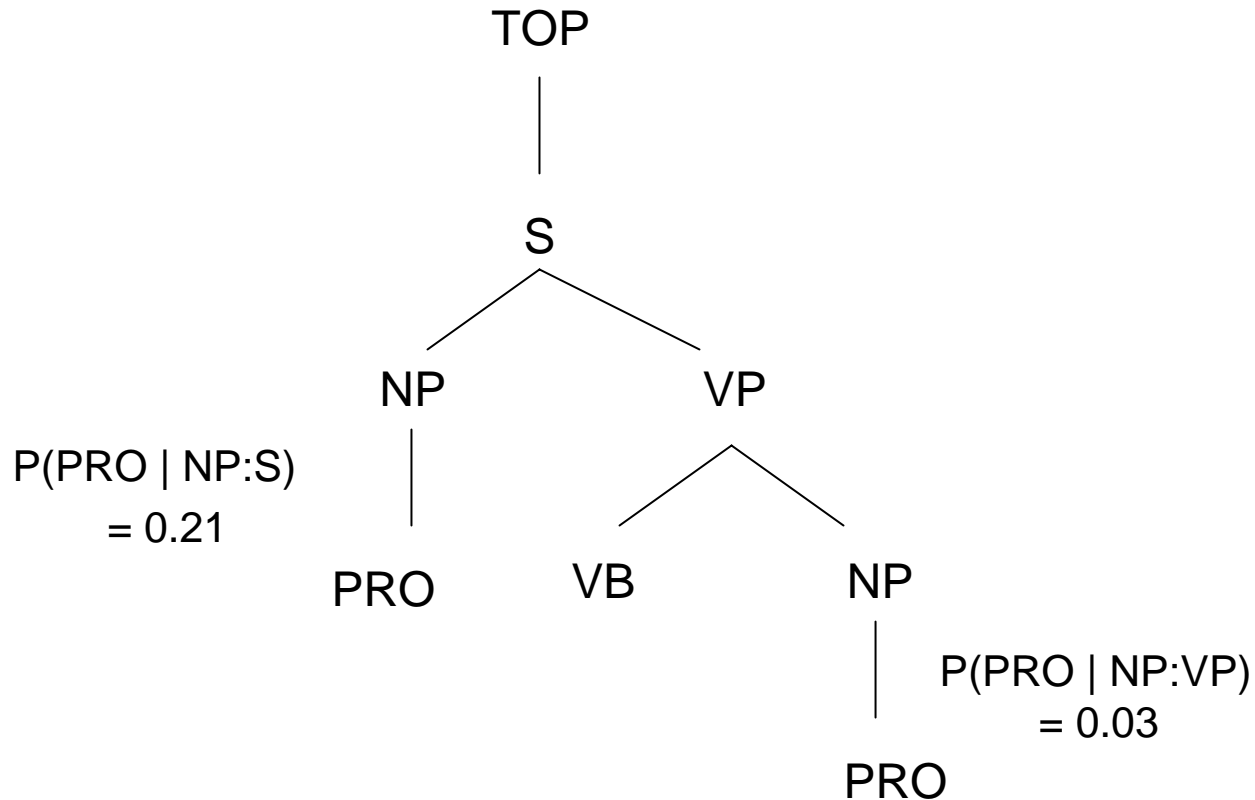
## Generative Process:



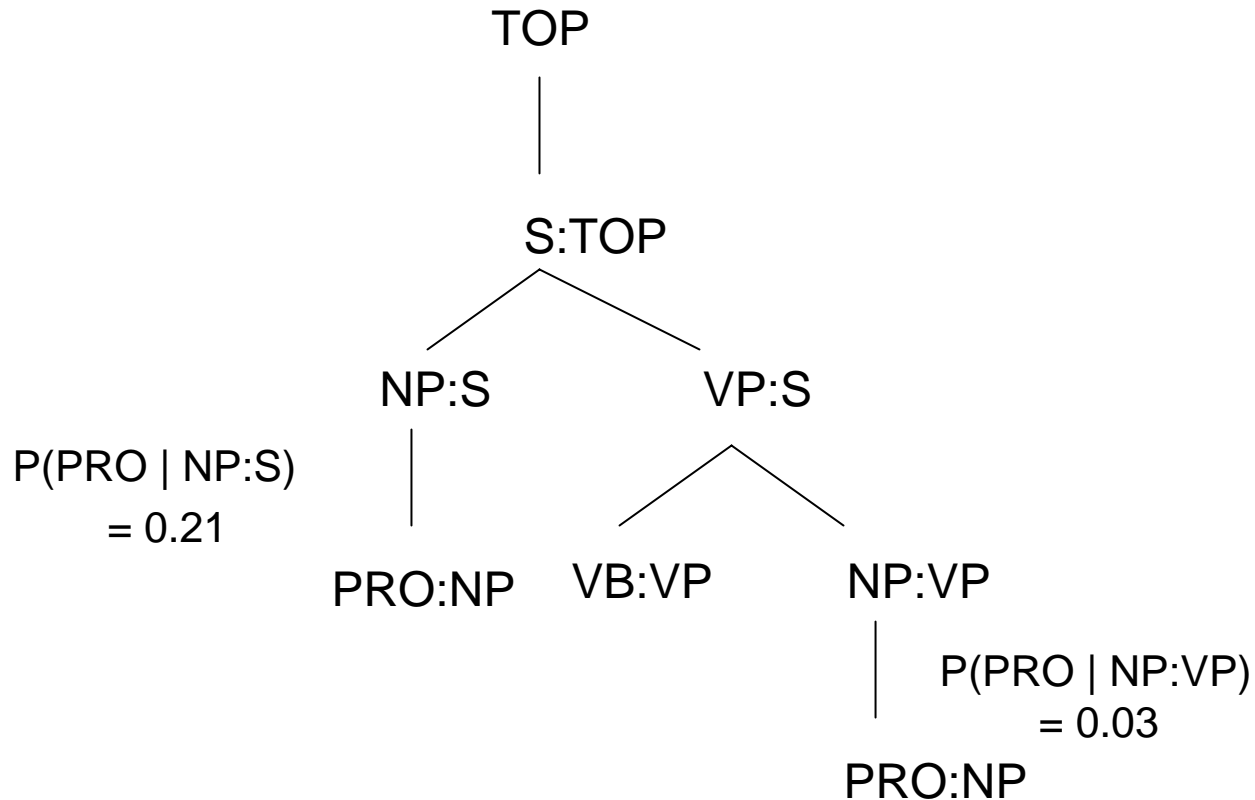
$$P(t) = 1.0 \times 0.3 \times 0.7 \times 0.5 \times 0.7 \times 0.5$$



# Relation of RTG to Johnson [98] Model



# Relation of RTG to Johnson [98] Model



# Relation of RTG to Johnson [98] Model

RTG:

$qstart \rightarrow S(q.np:s, q.vp:s)$

$q.np:s \rightarrow NP(q.pro:np)$

$q.np:vp \rightarrow NP(q.pro:np) \quad p=0.03$

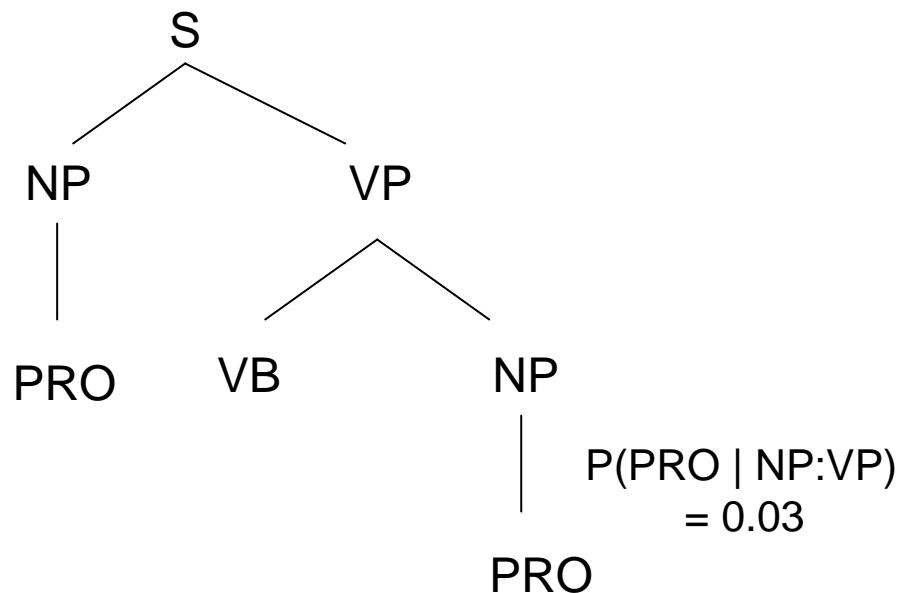
$q.pro:np \rightarrow PRO(q.pro)$

$q.pro:np \rightarrow he$

$q.pro:np \rightarrow she$

$q.pro:np \rightarrow him$

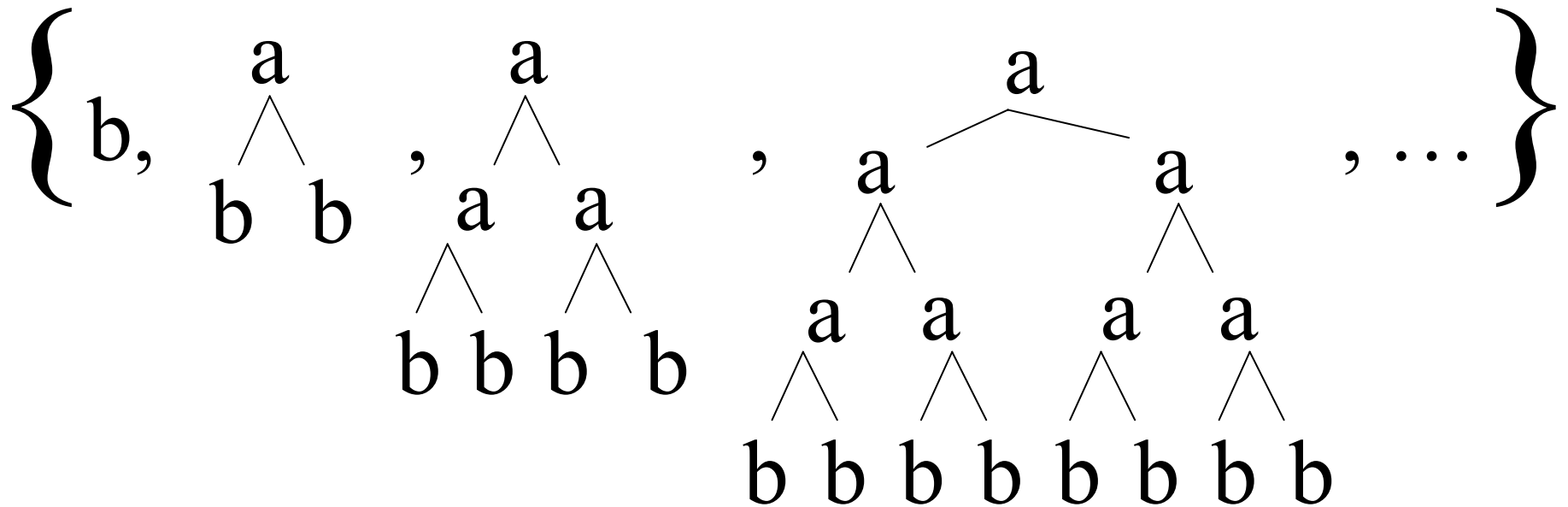
$q.pro:np \rightarrow her$



# Relation of RTG to Collins Model

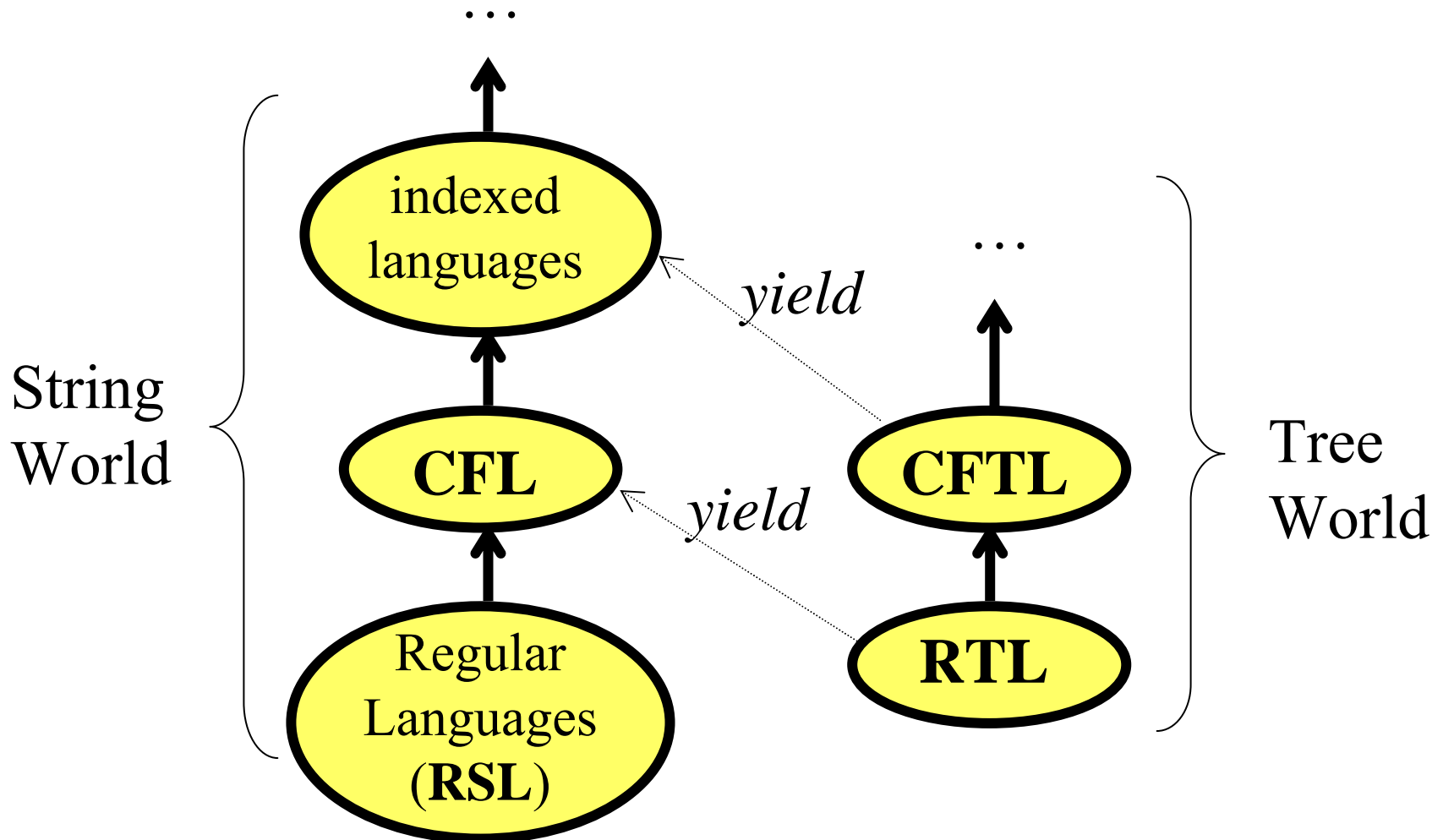
- Collins and Charniak assign  $P(t)$  to every tree
- Can weighted RTG (wRTG) implement Collins'  $P(t)$  for language modeling?
  - Just like a WFSA can implement smoothed n-gram language model?
- Roughly yes.
  - States encode relevant context that is passed up and down the tree.
- Two technical problems:
  - “Markovized” grammar (“Extended” RTG?)
  - Collins' model requires keep head-word information, and if we “back off” to a state that forgets head-word, we can't get it back.

# A Tree Language Accepted by No RTG



Note also that the *yield language* is  $\{b^{2^n} : n \geq 1\}$ , which is not context-free.

# Language Classes in String World and Tree World



# More on RTG and CFG

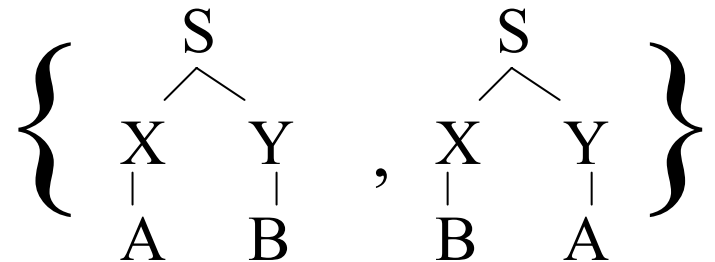
- For every CFG, there is an RTG that directly generates its derivation trees
  - **TRUE** (just convert the notation)
- For every RTG, there is a CFG that generates the same trees as derivations

– **FALSE**

$q \rightarrow S(X(A), Y(B))$

$q \rightarrow S(X(B), Y(A))$

accepts



# Picture So Far

	String Languages	Tree Languages
Grammars	Regular grammar (RG)	Regular tree grammar (RTG)
Acceptors	Finite-state acceptor (FSA)	?



# FSA Analog: Top-Down Tree Acceptor

[Rabin, 1969; Doner, 1970]

RTG:

$q \rightarrow S(X(A), Y(B))$

$q \rightarrow S(X(B), Y(A))$

TDTA:

$q S \rightarrow q2 q3$

$q3 X \rightarrow q3$

$q S \rightarrow q3 q2$

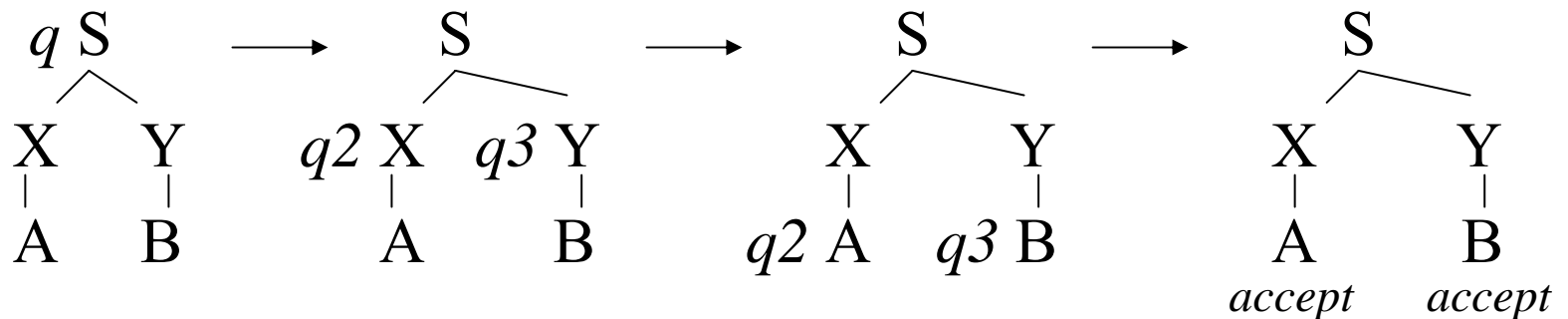
$q3 Y \rightarrow q3$

$q2 X \rightarrow q2$

$q2 A \rightarrow \text{accept}$

$q2 Y \rightarrow q2$

$q3 B \rightarrow \text{accept}$



For any RTG, there is a TDTA that accepts the same trees.  
 For any TDTA, there is an RTG that accepts the same trees.

# FSA Analog: Bottom-Up Tree Acceptor

[Thatcher, 1967]

- A similar story...

# Properties of Language Classes

	String World		Tree World
	<b>RSL</b> (FSA, rexp)	<b>CFL</b> (PDA, CFG)	<b>RTL</b> (TDTA, RTG)
Closed under union	YES	YES	YES
Closed under intersection	YES	<b>NO</b>	YES
Closed under complement	YES	<b>NO</b>	YES
Membership testing	$O(n)$	$O(n^3)$	$O(n)$
Emptiness decidable?	YES	YES	YES
Equality decidable?	YES	<b>NO</b>	YES

(references: Hopcroft & Ullman 79, Gécseg & Steinby 84)

# Tree Adjoining Grammar (TAG)

- Example:

- S(NP↓, VP(V(hit), NP↓))
- NP(D↓, N(boy))
- NP(D↓, N(dragon))
- NP(John)
- D(the)
- VP(VP\*, ADV(today))
- ADJ(ADV(very), ADJ\*)
- N(ADJ(funny), N\*)

- Defines a set of *trees*

## Generative Process:

S

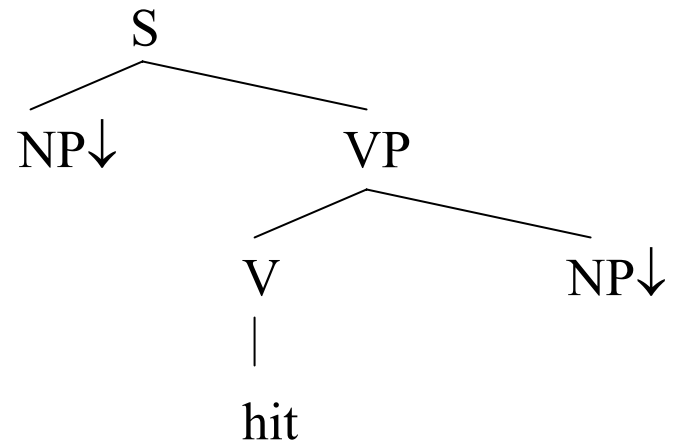
# Tree Adjoining Grammar (TAG)

- Example:

- $S(NP\downarrow, VP(V(hit), NP\downarrow))$
- $NP(D\downarrow, N(boy))$
- $NP(D\downarrow, N(dragon))$
- $NP(John)$
- $D(the)$
- $VP(VP^*, ADV(today))$
- $ADJ(ADV(very), ADJ^*)$
- $N(ADJ(funny), N^*)$

- Defines a set of *trees*

## Generative Process:

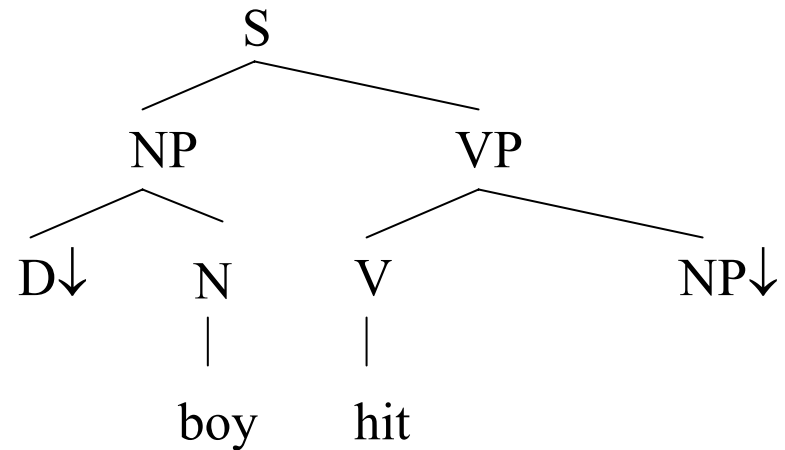


# Tree Adjoining Grammar (TAG)

- Example:

- S(NP↓, VP(V(hit), NP↓))
- NP(D↓, N(boy))
- NP(D↓, N(dragon))
- NP(John)
- D(the)
- VP(VP\*, ADV(today))
- ADJ(ADV(very), ADJ\*)
- N(ADJ(funny), N\*)

## Generative Process:



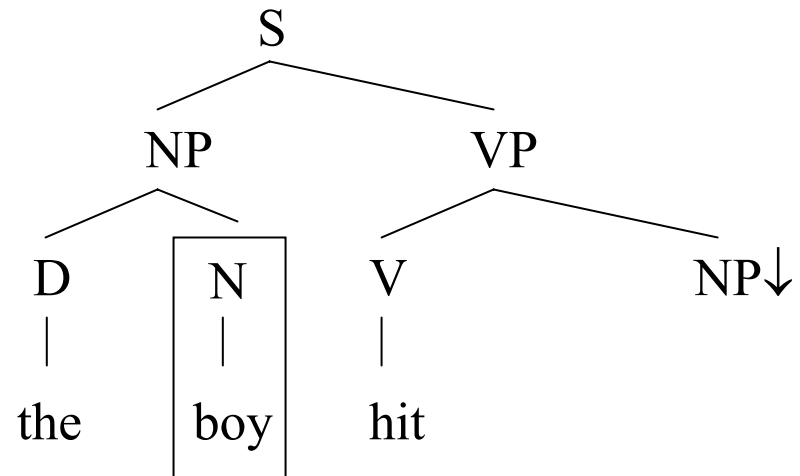
- Defines a set of *trees*

# Tree Adjoining Grammar (TAG)

- Example:

- $S(NP\downarrow, VP(V(hit), NP\downarrow))$
- $NP(D\downarrow, N(boy))$
- $NP(D\downarrow, N(dragon))$
- $NP(John)$
- $D(the)$
- $VP(VP^*, ADV(today))$
- $ADJ(ADV(very), ADJ^*)$
- $N(ADJ(funny), N^*)$

## Generative Process:



So far, just like RTG/TSG.

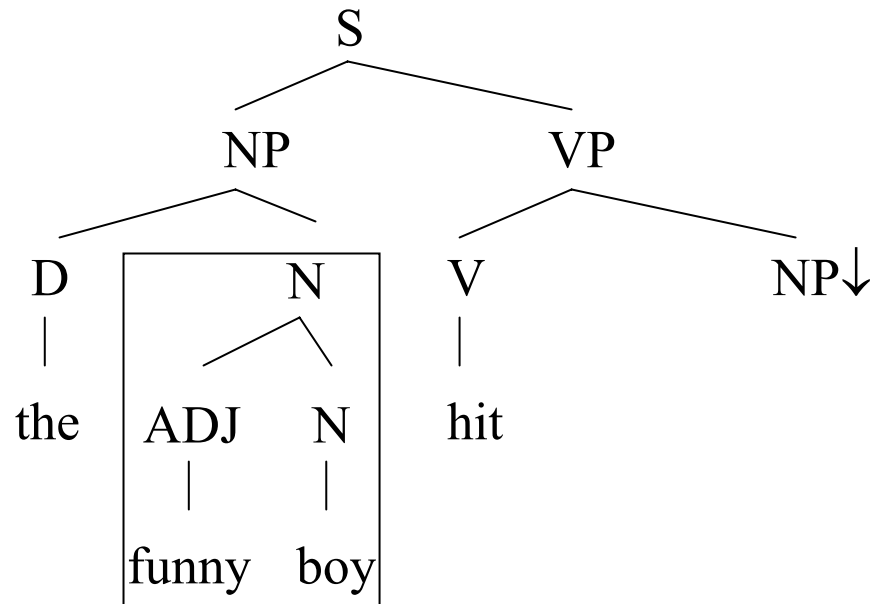
- Defines a set of *trees*

# Tree Adjoining Grammar (TAG)

- Example:

- $S(NP\downarrow, VP(V(hit), NP\downarrow))$
- $NP(D\downarrow, N(boy))$
- $NP(D\downarrow, N(dragon))$
- $NP(John)$
- $D(the)$
- $VP(VP^*, ADV(today))$
- $ADJ(ADV(very), ADJ^*)$
- $N(ADJ(funny), N^*)$

## Generative Process:



- Defines a set of *trees*

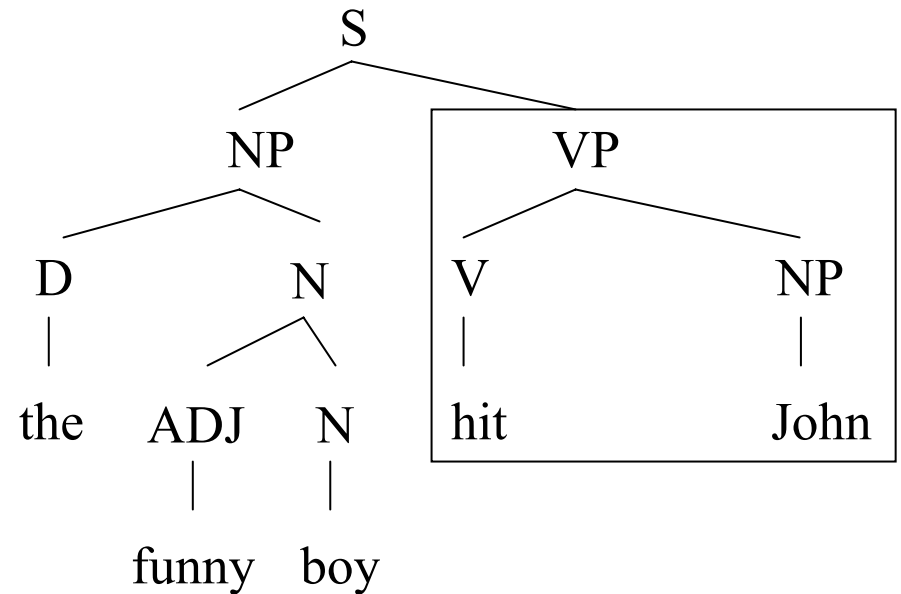


# Tree Adjoining Grammar (TAG)

- Example:

- $S(NP\downarrow, VP(V(hit), NP\downarrow))$
- $NP(D\downarrow, N(boy))$
- $NP(D\downarrow, N(dragon))$
- $NP(John)$
- $D(the)$
- $VP(VP^*, ADV(today))$
- $ADJ(ADV(very), ADJ^*)$
- $N(ADJ(funny), N^*)$

## Generative Process:



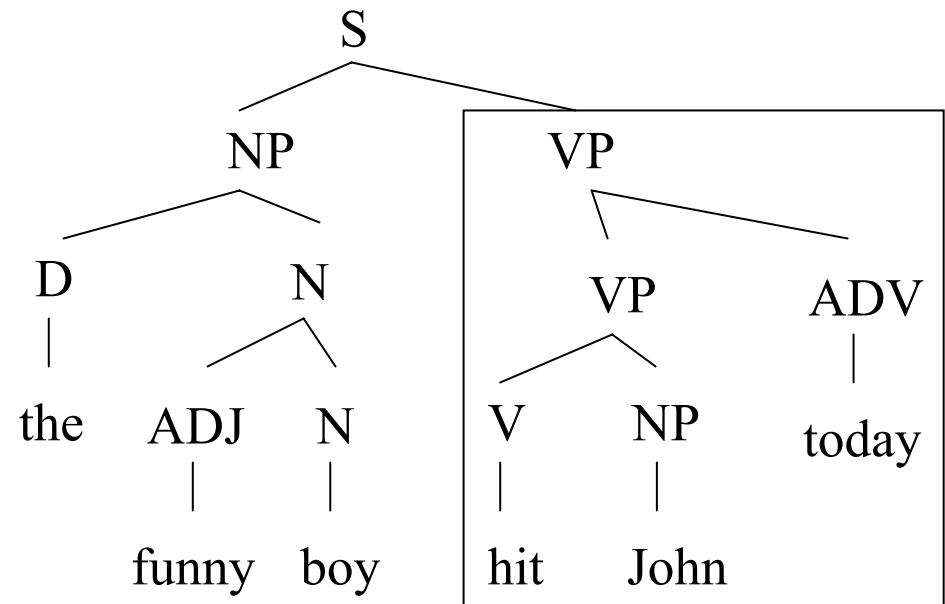
- Defines a set of *trees*

# Tree Adjoining Grammar (TAG)

- Example:

- $S(NP\downarrow, VP(V(hit), NP\downarrow))$
- $NP(D\downarrow, N(boy))$
- $NP(D\downarrow, N(dragon))$
- $NP(John)$
- $D(the)$
- $VP(VP^*, ADV(today))$
- $ADJ(ADV(very), ADJ^*)$
- $N(ADJ(funny), N^*)$

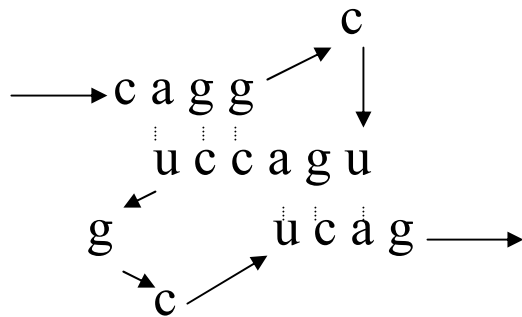
## Generative Process:



- Defines a set of *trees*

# Tree Adjoining Grammar (TAG)

- Can generate tree sets that RTG cannot
- Elements far apart in the tree can depend on each other.
  - Mary attacked.
  - Mary had attacked.
  - Mary had attacked yesterday. [now Mary & attacked are far apart]
- Has recently been used to model RNA structure
  - Observed string: ... c a g g c u g a c c u g c u c a g ...
  - Most likely folding structure according to grammar:



[Uemura, Hasegawa, Kobayashi, Yokomori, 1999]  
[Rivas and Eddy, 2000]

# Picture So Far

## String Language

- FSA compactly represents a possibly-infinite set of strings
- Probabilistic FSA assigns a  $P(s)$  to every string
- Can ask: Is  $s$  grammatical & sensible?

## Tree Language

- RTG compactly represents a possibly-infinite set of trees
- Probabilistic RTG assigns a  $P(t)$  to every tree
- Can ask: Is  $t$  grammatical & sensible?

# Picture So Far

## String Language

- FSA compactly represents a possibly-infinite set of strings
- Probabilistic FSA assigns a  $P(s)$  to every string
- Can ask: Is  $s$  grammatical & sensible?

## String Transducer

- FST compactly represents a possibly-infinite set of *string pairs*
- Probabilistic FST assign  $P(s_2 \mid s_1)$  to every *string pair*
- Can ask: What's the best transformation of string  $s_1$ ?

## Tree Language

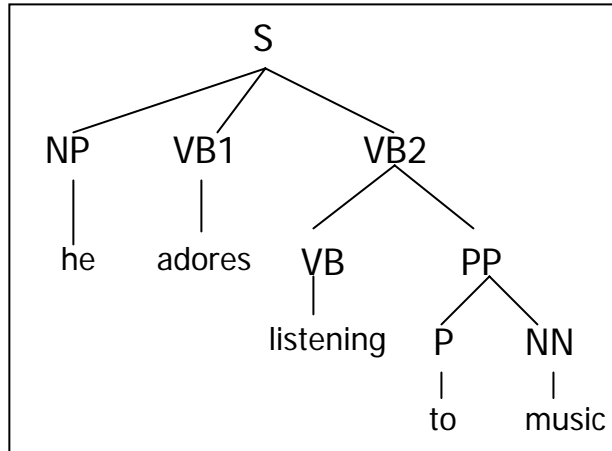
- RTG compactly represents a possibly-infinite set of trees
- Probabilistic RTG assigns a  $P(t)$  to every tree
- Can ask: Is  $t$  grammatical & sensible?

## Tree Transducer

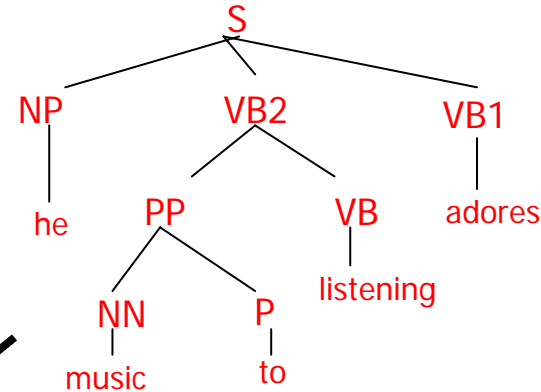
- Tree transducers compactly represent a possibly-infinite set of *tree pairs*
- Probabilistic tree transducers assign  $P(t_2 \mid t_1)$  to every *tree pair*
- Can ask: What's the best transformation of tree  $t_1$ ?

# Example 1: Machine Translation (Yamada & Knight 2001)

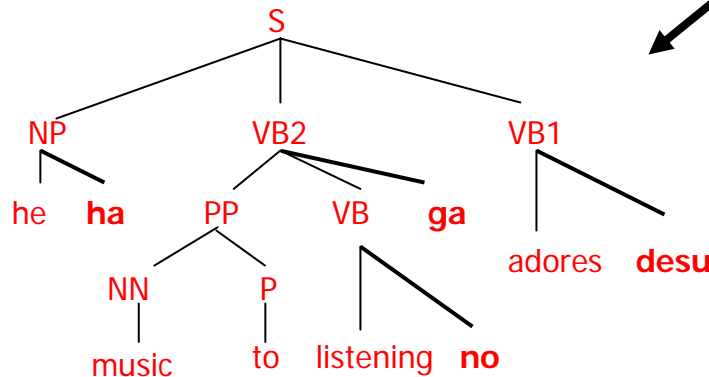
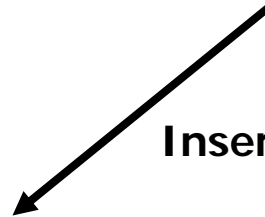
Parse Tree(E)



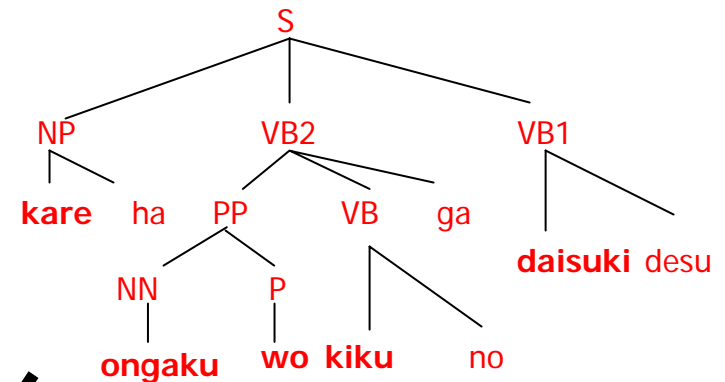
Reorder



Insert



Translate



Take Leaves

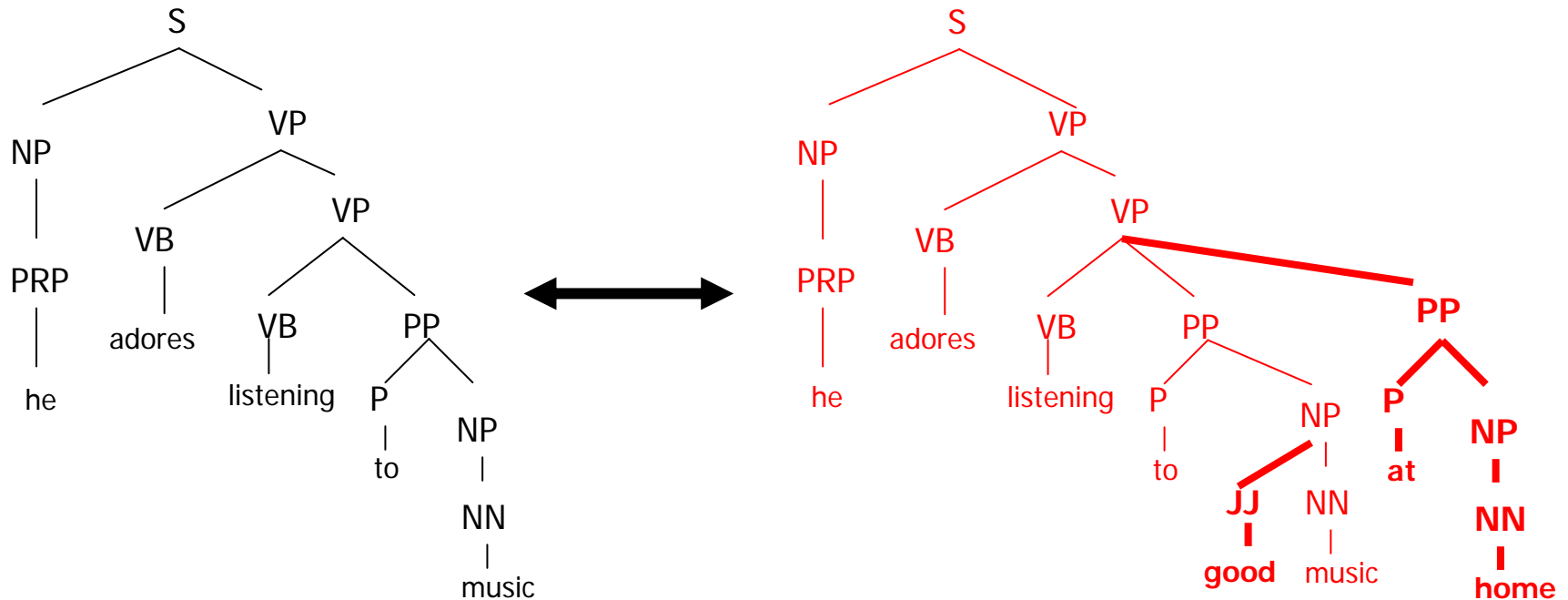


Sentence(J)

**Kare ha ongaku wo kiku no ga daisuki desu**

# Example 2: Sentence Compression

Knight & Marcu (2000)



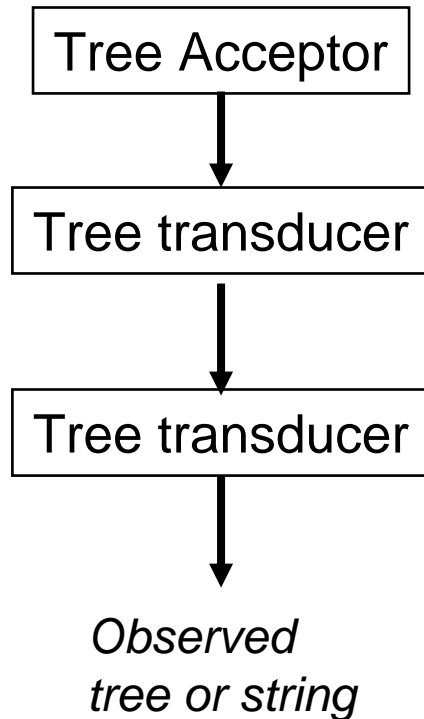
# What We'd Like to Do

- Device T for compactly representing possibly-infinite set of tree pairs (possibly with weights)
- Want to support operations like:
  - **Forward application:** Send input tree X through T, get all possible output trees, represented as an RTG.
  - **Backward application:** What input trees, when sent through T, would output tree X?
  - **Composition:** Given a sequence of T1 and T2, can we build a transducer T3 that does the work of both?
  - **Equality testing:** Do T1 and T2 contain exactly the same tree pairs?
  - **Training:** How to set weights given a corpus?



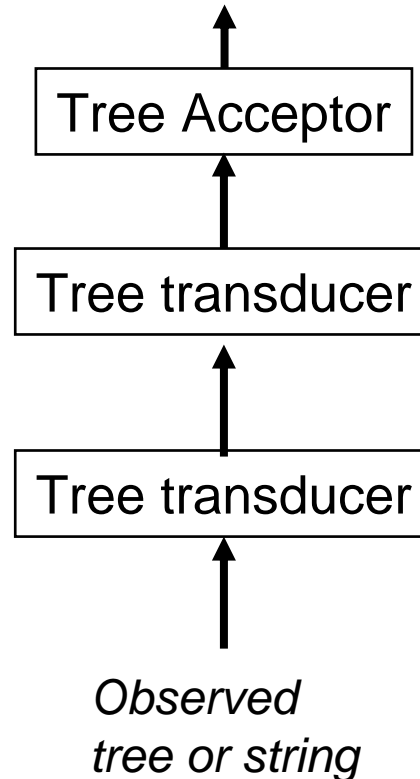
# Transducer Cascades

## modeling direction:



## decoding direction:

*tree acceptor containing all (scored) answers*



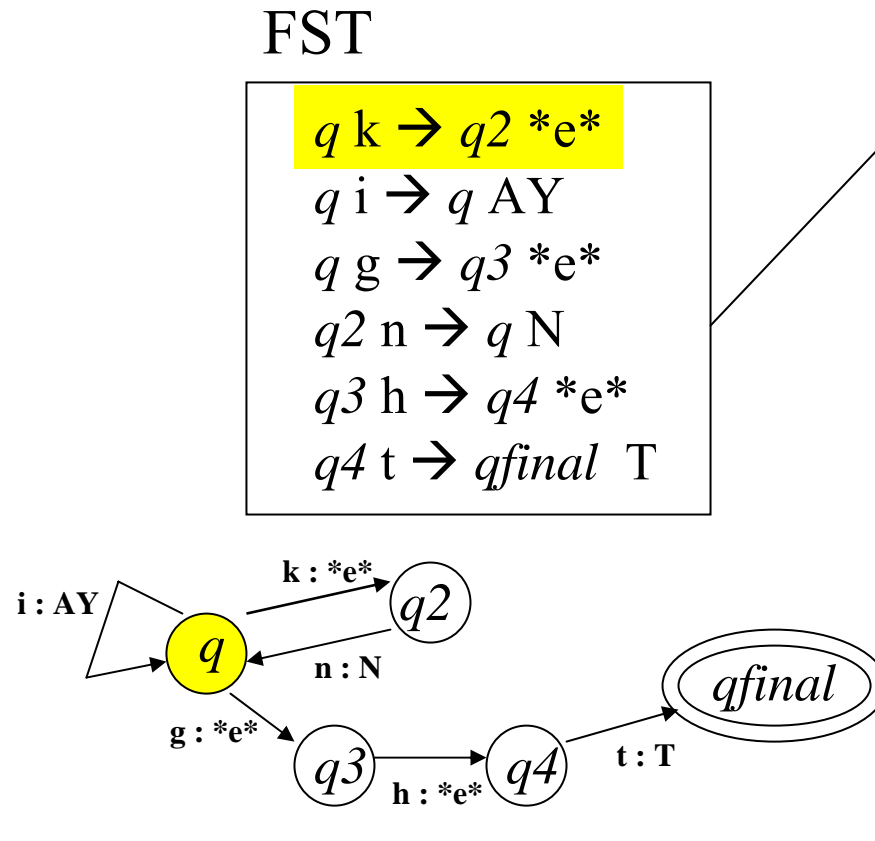
# Finite-State Transducer (FST)

Original input:

k  
|  
n  
|  
i  
|  
g  
|  
h  
|  
t

Transformation:

$q$  k  
|  
n  
|  
i  
|  
g  
|  
h  
|  
t



# Finite-State (String) Transducer

Original input:

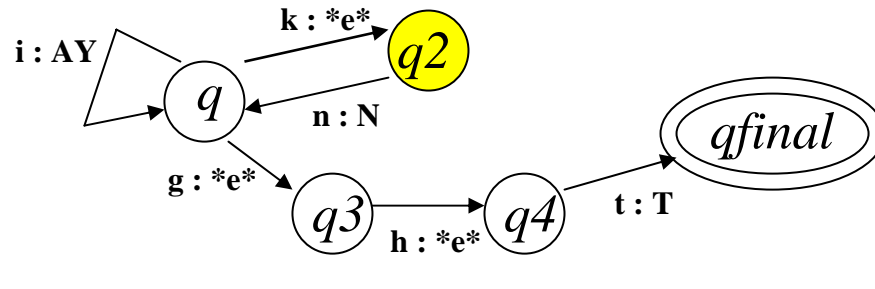
k  
|  
n  
|  
i  
|  
g  
|  
h  
|  
t

Transformation:

q2 n  
|  
i  
|  
g  
|  
h  
|  
t

FST

$q \ k \rightarrow q2 \ *e^*$   
 $q \ i \rightarrow q \ AY$   
 $q \ g \rightarrow q3 \ *e^*$   
 $q2 \ n \rightarrow q \ N$   
 $q3 \ h \rightarrow q4 \ *e^*$   
 $q4 \ t \rightarrow q_{final} \ T$



# Finite-State (String) Transducer

Original input:

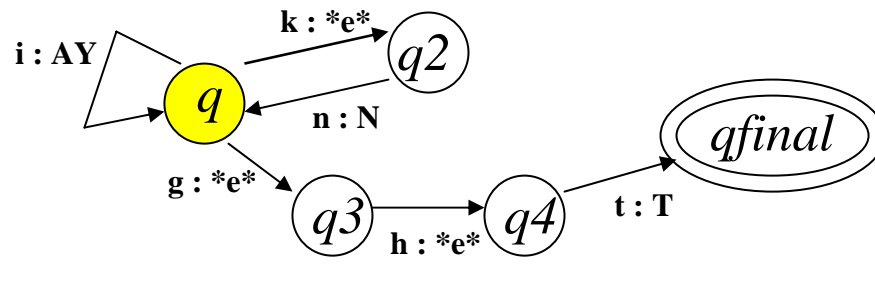
k  
|  
n  
|  
i  
|  
g  
|  
h  
|  
t

Transformation:

**N**  
|  
i  
|  
g  
|  
h  
|  
t

FST

$q \ k \rightarrow q2 \ *e^*$   
 $q \ i \rightarrow q \ AY$   
 $q \ g \rightarrow q3 \ *e^*$   
 $q2 \ n \rightarrow q \ N$   
 $q3 \ h \rightarrow q4 \ *e^*$   
 $q4 \ t \rightarrow q_{final} \ T$



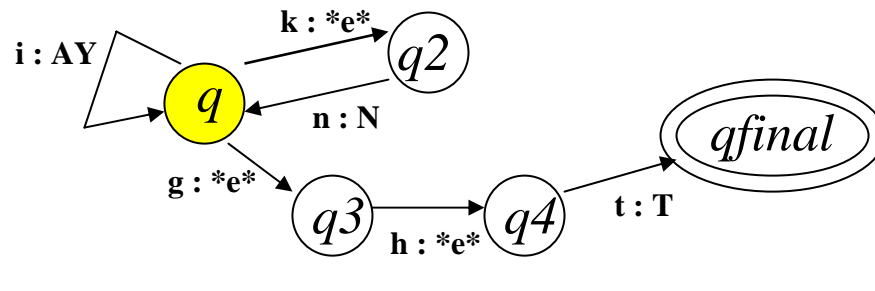
# Finite-State (String) Transducer

Original input:

k  
|  
n  
|  
i  
|  
g  
|  
h  
|  
t

FST

$q \ k \rightarrow q2 \ *e^*$   
 $q \ i \rightarrow q \ AY$   
 $q \ g \rightarrow q3 \ *e^*$   
 $q2 \ n \rightarrow q \ N$   
 $q3 \ h \rightarrow q4 \ *e^*$   
 $q4 \ t \rightarrow q_{final} \ T$



Transformation:

**N**  
|  
**AY**  
|  
 $q \ g$   
|  
h  
|  
t

# Finite-State (String) Transducer

Original input:

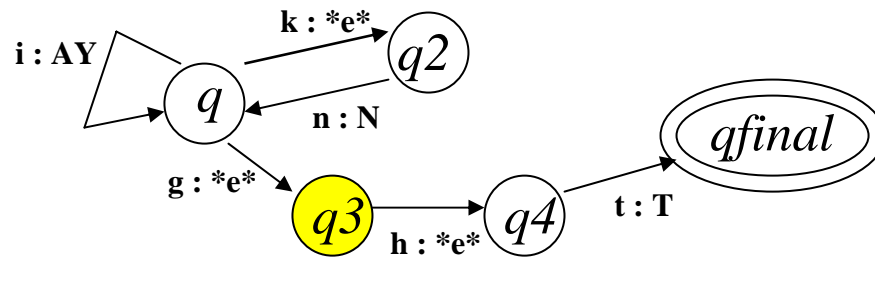
k  
|  
n  
|  
i  
|  
g  
|  
h  
|  
t

Transformation:

**N**  
|  
**AY**  
|  
q3 h  
|  
t

FST

$q \ k \rightarrow q2 \ *e^*$   
 $q \ i \rightarrow q \ AY$   
 $q \ g \rightarrow q3 \ *e^*$   
 $q2 \ n \rightarrow q \ N$   
 **$q3 \ h \rightarrow q4 \ *e^*$**   
 $q4 \ t \rightarrow q_{final} \ T$

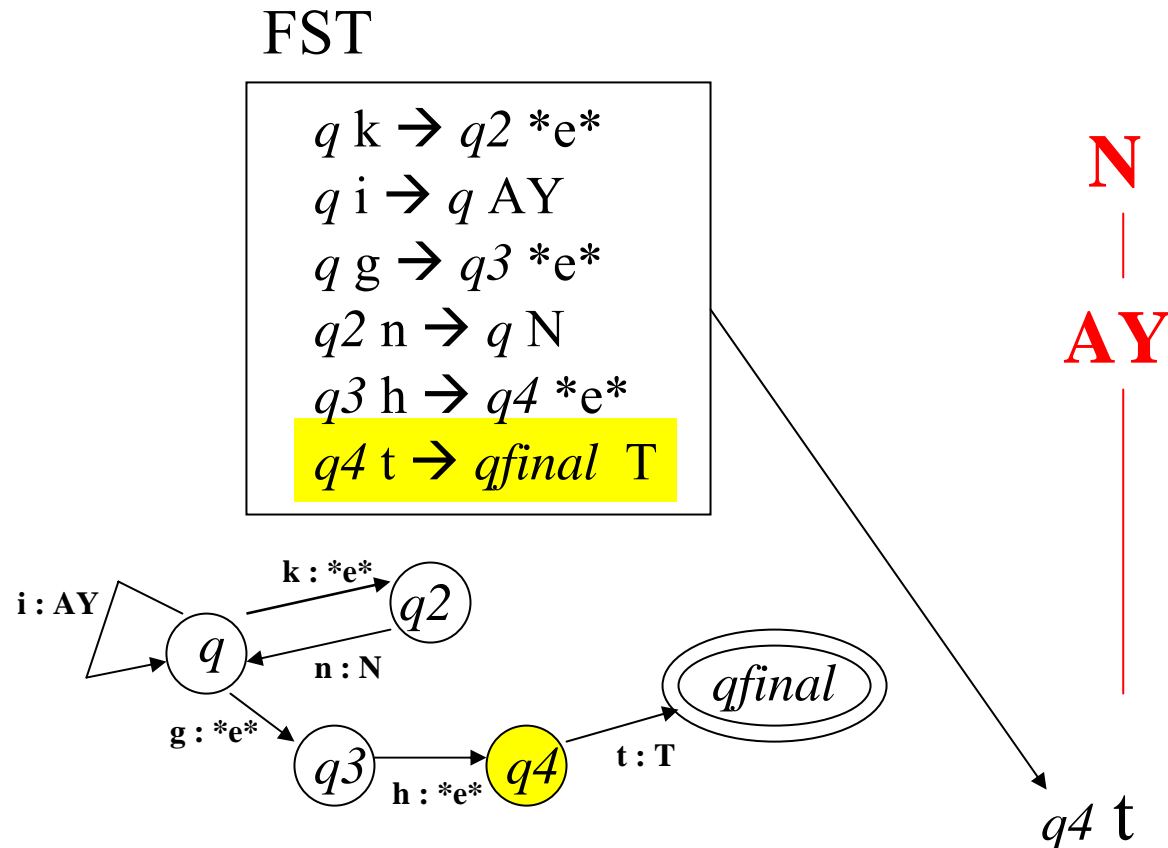


# Finite-State (String) Transducer

Original input:

k  
|  
n  
|  
i  
|  
g  
|  
h  
|  
t

Transformation:



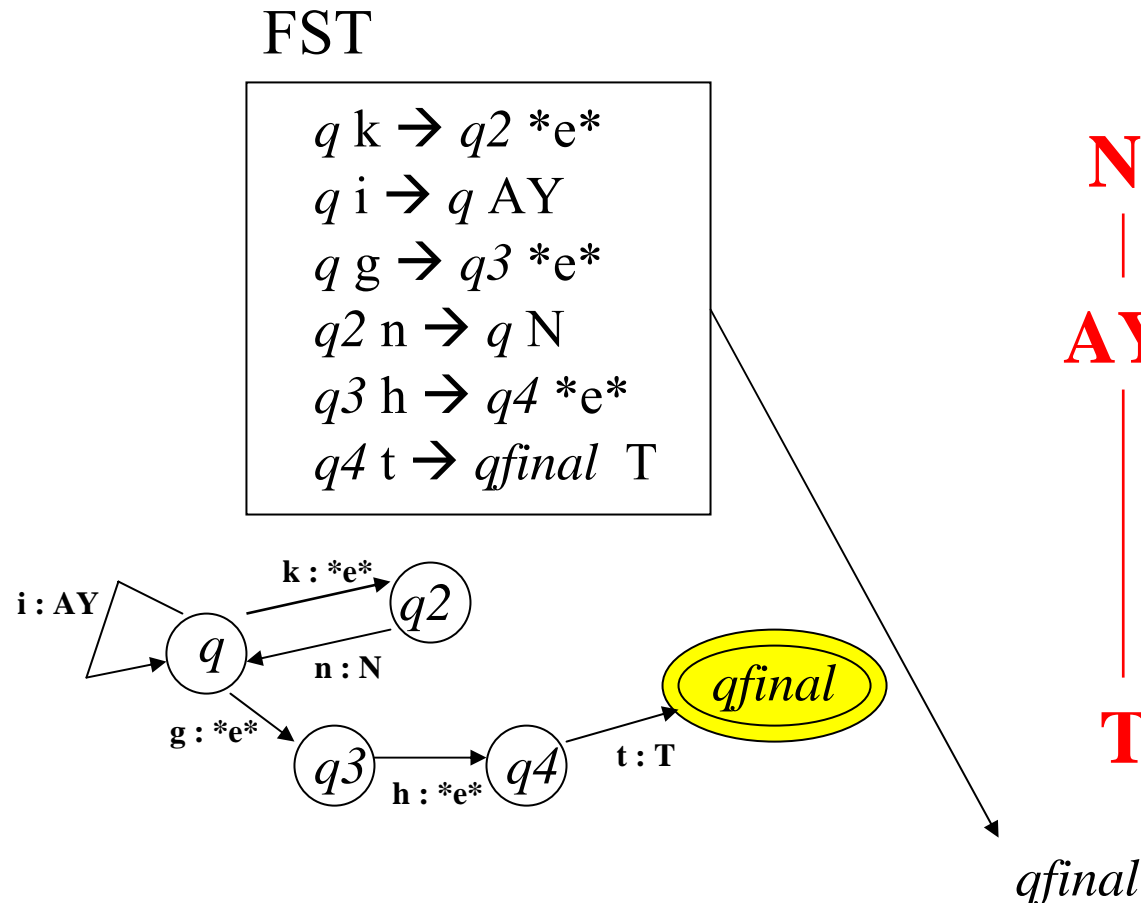
# Finite-State (String) Transducer

Original input:

k  
|  
n  
|  
i  
|  
g  
|  
h  
|  
t

Transformation:

**N**  
|  
**AY**  
|  
**T**





# Finite-State (String) Transducer

Original input:

k  
|  
n  
|  
i  
|  
g  
|  
h  
|  
t

FST

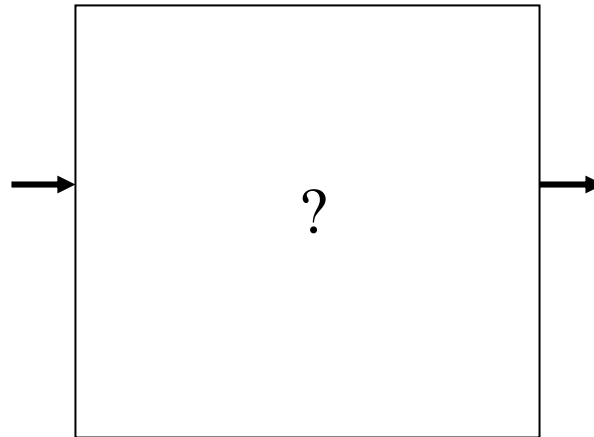
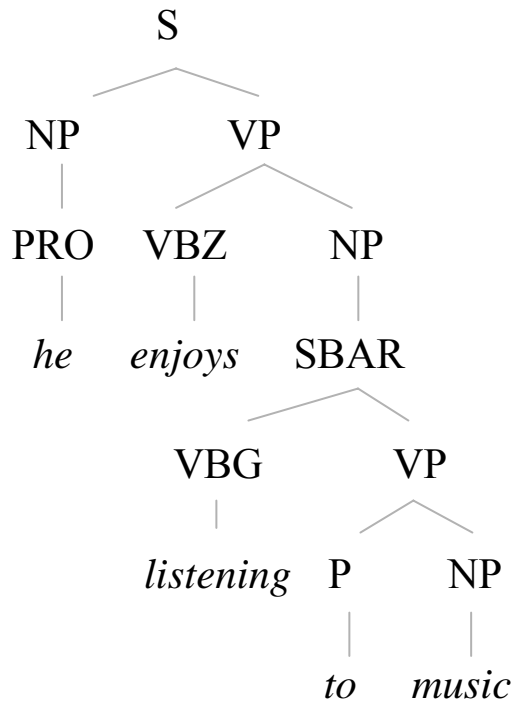
$q\ k \rightarrow q2\ *e*$   
 $q\ i \rightarrow q\ AY$   
 $q\ g \rightarrow q3\ *e*$   
 $q2\ n \rightarrow q\ N$   
 $q3\ h \rightarrow q4\ *e*$   
 $q4\ t \rightarrow q_{final}\ T$

Final output:

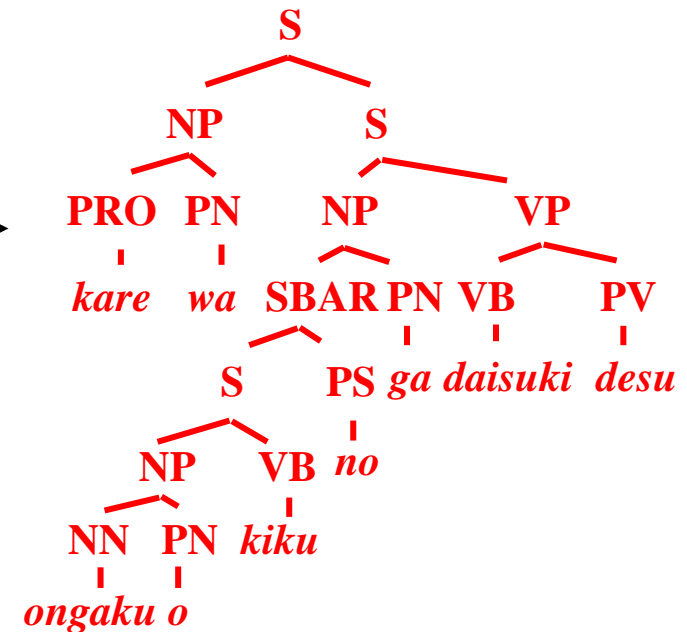
**N**  
|  
**AY**  
|  
**T**

# Trees

Original input:

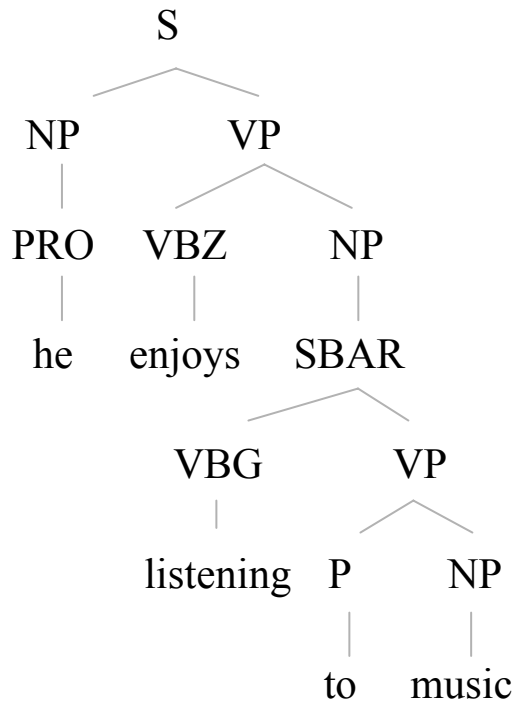


Target output:

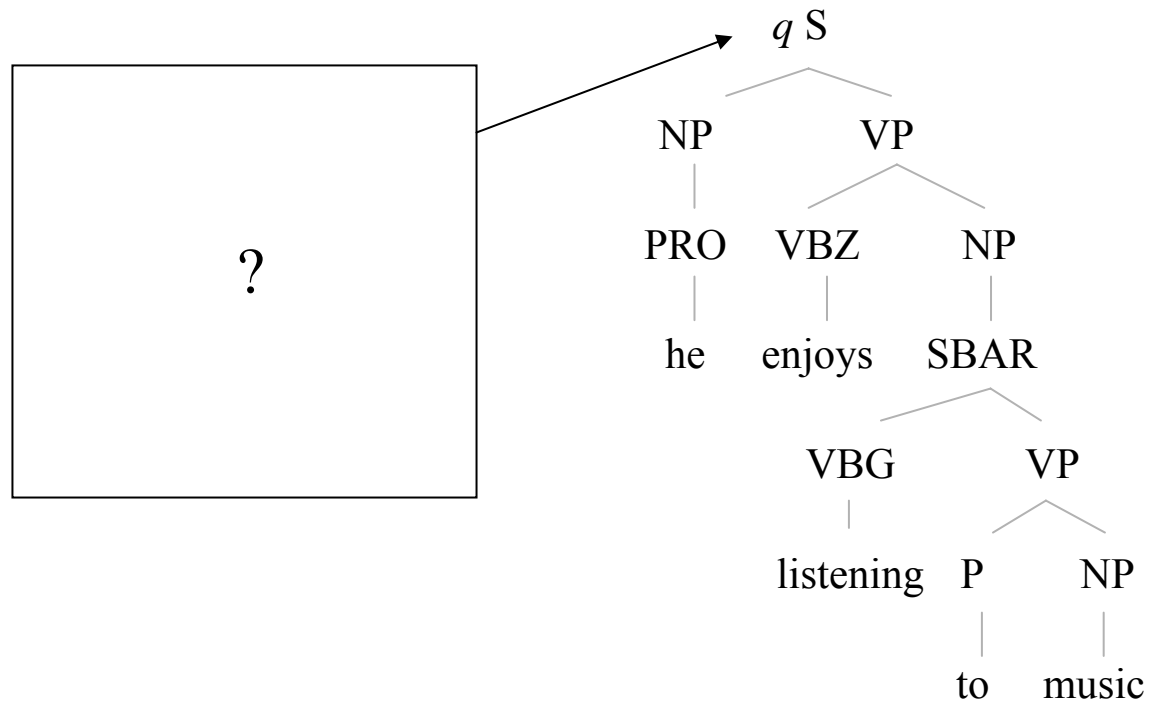


# Tree Transformation

Original input:

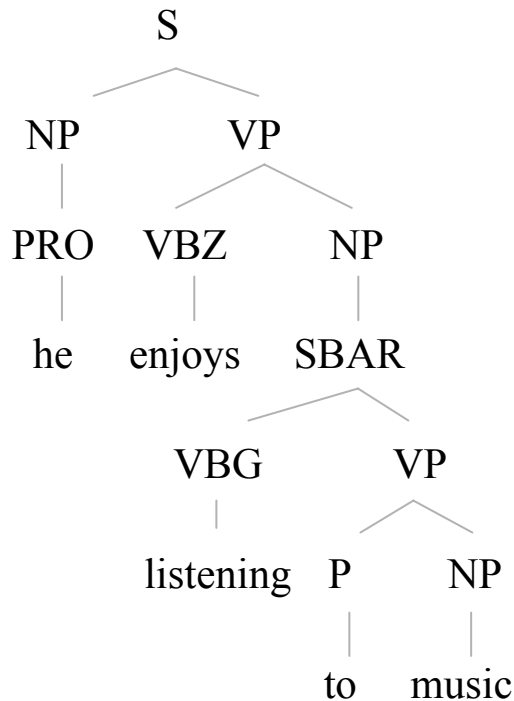


Transformation:

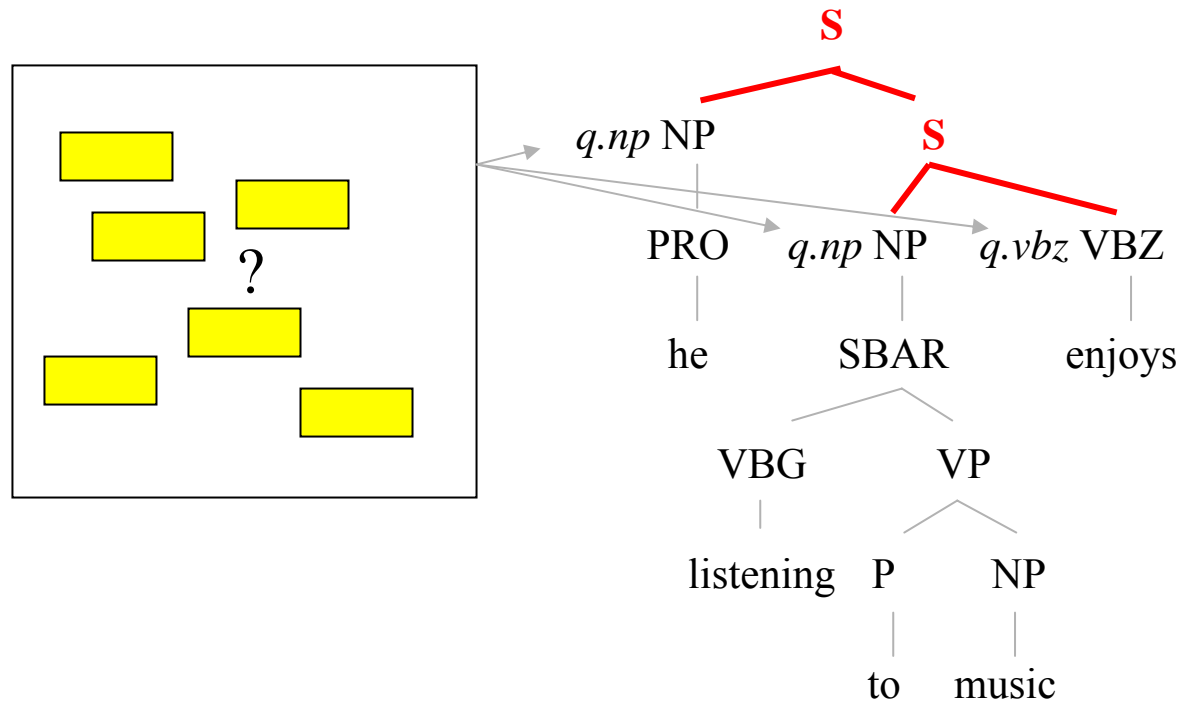


# Tree Transformation

Original input:

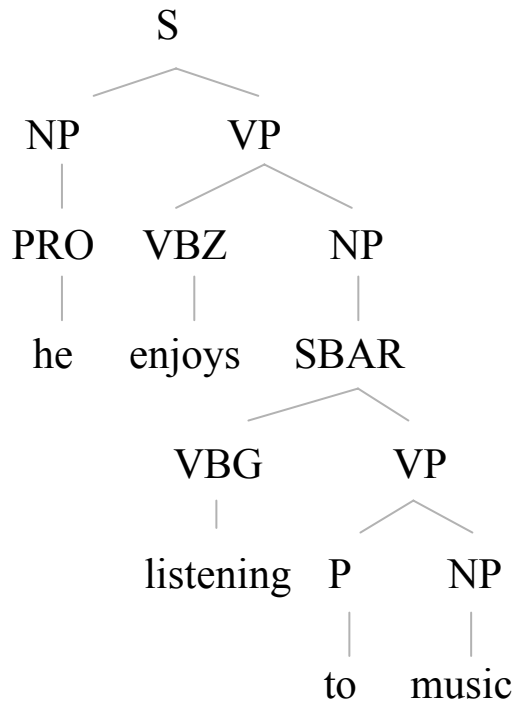


Transformation:

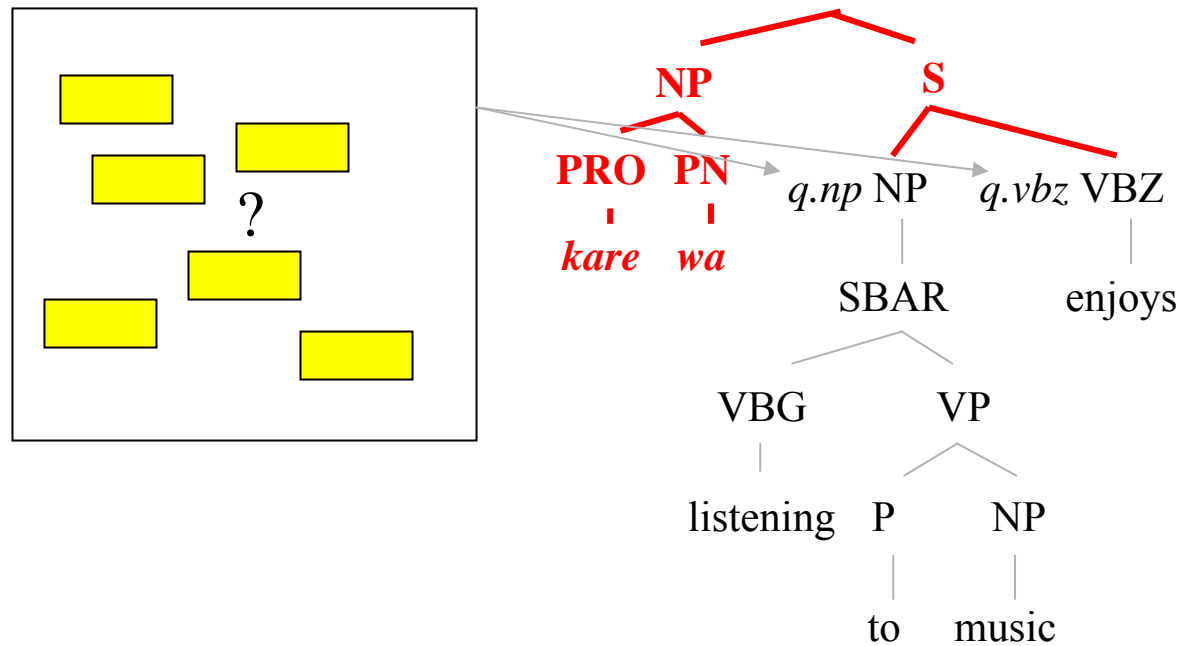


# Trees

Original input:

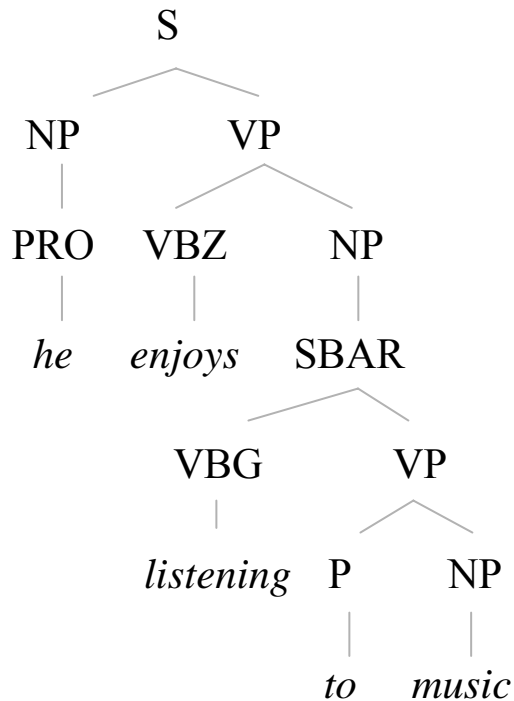


Transformation:

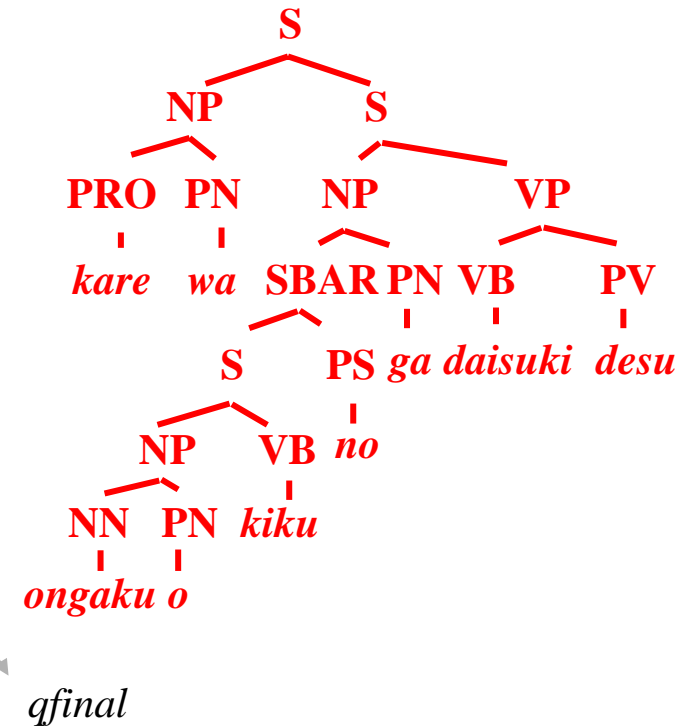


# Trees

Original input:



Final output:

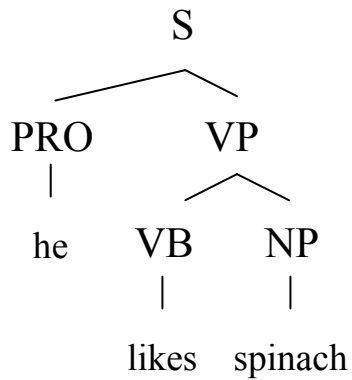


# Top-Down Tree Transducer

- Introduced by Rounds (1970) & Thatcher (1970)
  - “Recent developments in the theory of automata have pointed to an extension of the domain of definition of automata from strings to trees ... parts of mathematical linguistics can be formalized easily in a tree-automaton setting ... Our results should clarify the nature of syntax-directed translations and transformational grammars ...” (Mappings on Grammars and Trees, *Math. Systems Theory* 4(3), Rounds 1970)
- “FST for trees”
- Large theory literature
  - e.g., Gécseg & Steinby (1984), Comon et al (1997)
- Many classes of transducers
  - Top-down: **R** (“root-to-frontier”), RL, RLN...
  - Bottom-up: F, FL, FLN...

# Top-Down Tree Transducer

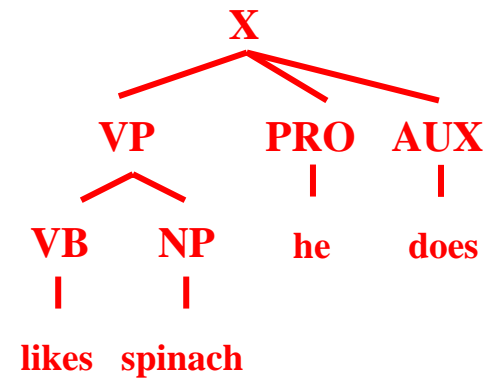
Original  
Input:



R transducer

?

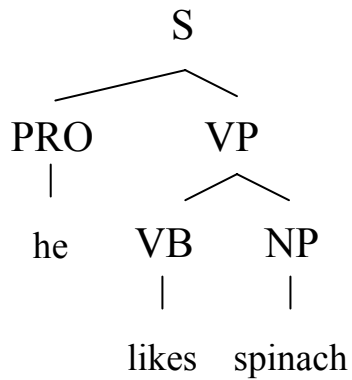
Target  
Output:



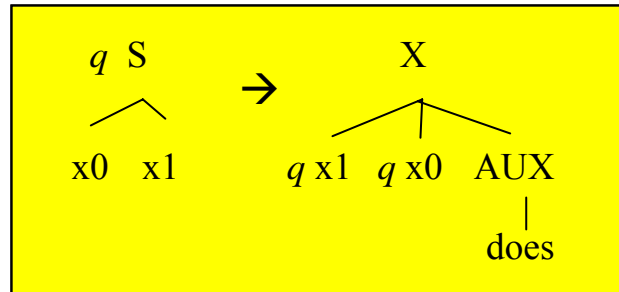


# Top-Down Tree Transducer

Original  
Input:



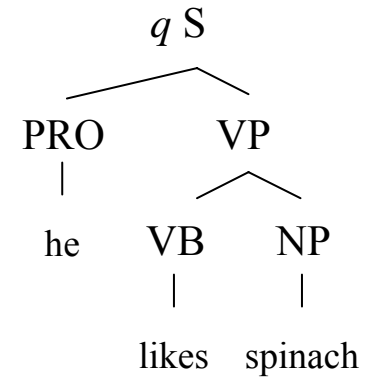
R transducer



Or in computer-speak:

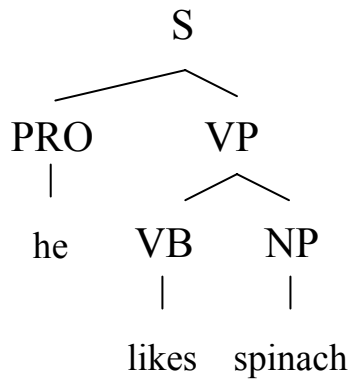
$q S(x0, x1) \rightarrow$   
 $X(q x1, q x0, AUX(does))$

Transformation:

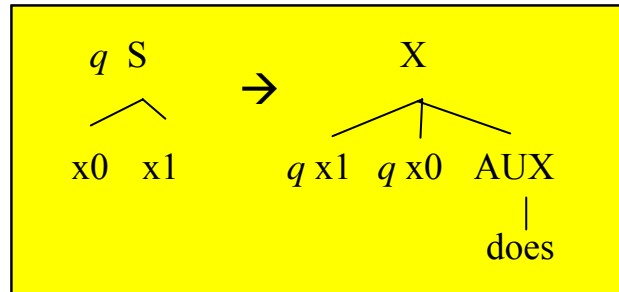


# Top-Down Tree Transducer

Original  
Input:



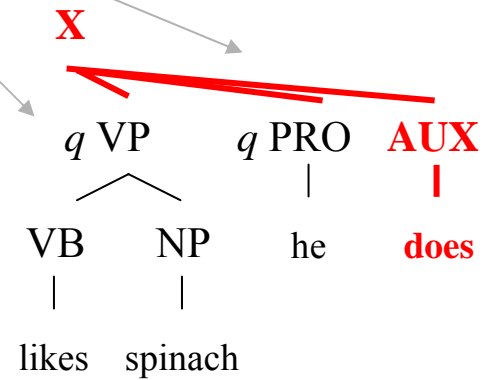
R transducer



Or in computer-speak:


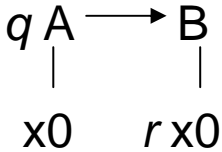
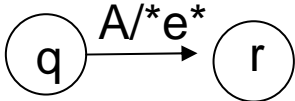
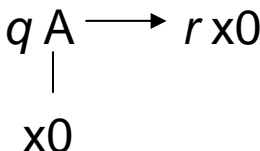
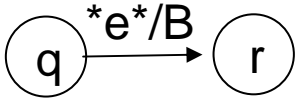
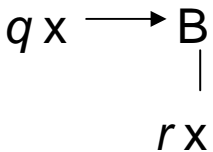
$q S(x0, x1) \rightarrow$   
 $X(q x1, q x0, AUX(does))$

Transformation:



# Tree Transducers and FSTs

- Tree transducers generalize FST (strings are “monadic trees”)

FST transition	Equivalent tree transducer rule
	
	
	

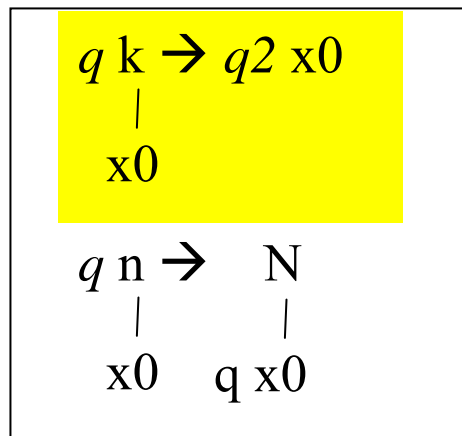
# Tree Transducer Simulating FST

Original input:

k  
|  
n  
|  
i  
|  
g  
|  
h  
|  
t

Transformation:

R Transducer



$q\ k$   
|  
n  
|  
i  
|  
g  
|  
h  
|  
t

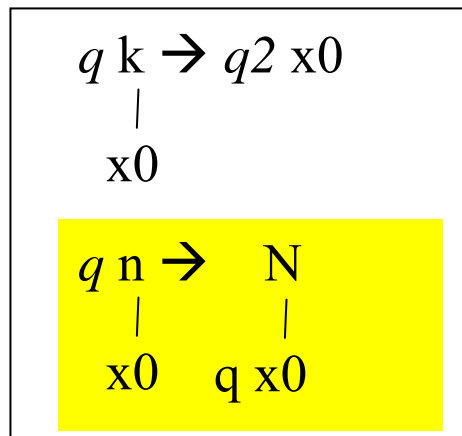
# Tree Transducer Simulating FST

Original input:

k  
|  
n  
|  
i  
|  
g  
|  
h  
|  
t

Transformation:

R Transducer



$q2$  n  
|  
i  
|  
g  
|  
h  
|  
t

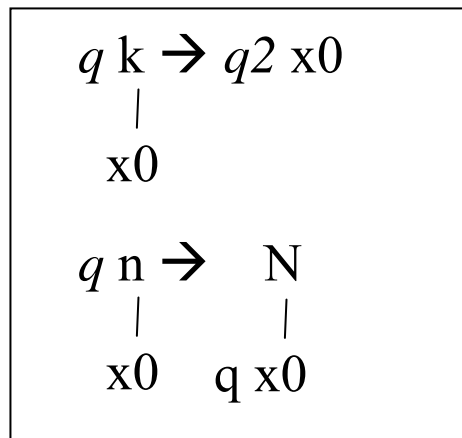
# Tree Transducer Simulating FST

Original input:

k  
|  
n  
|  
i  
|  
g  
|  
h  
|  
t

Transformation:

R Transducer

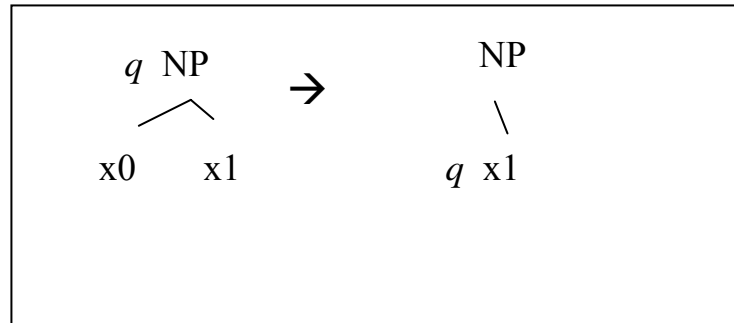
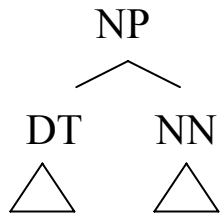


(and so on ...)

**N**  
|  
i  
|  
g  
|  
h  
|  
t

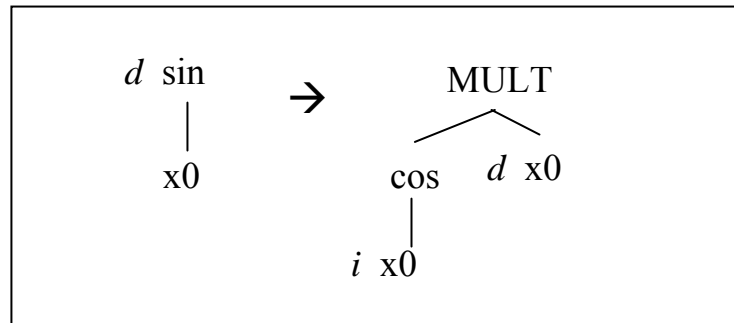
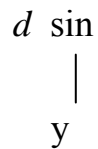
# R can Copy and/or Delete

R transducer



**NP**  
**/**  
**NN**  
△

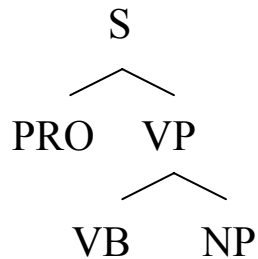
R transducer



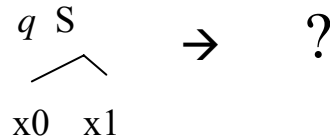
**MULT**  
**cos** **d y**  
|  
**i y**

# Complex Re-Ordering

Original  
Input:



R transducer



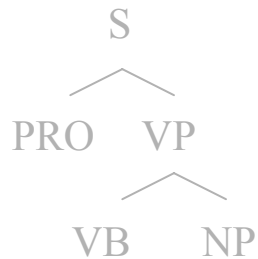
Target  
Output:





# Complex Re-Ordering

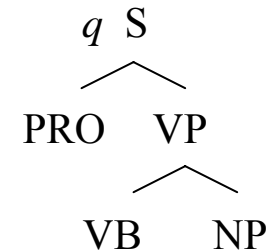
Original  
Input:



Transformation:

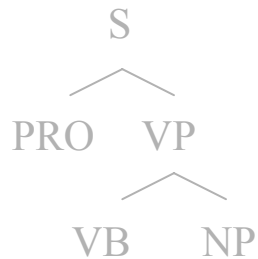
R transducer

$  \begin{array}{c}  q \ S \\  \swarrow \searrow \\  x0 \ x1  \end{array}  \rightarrow  \begin{array}{c}  S \\  \swarrow \quad \downarrow \quad \searrow \\  qlift \ x1 \quad q \ x0 \quad qright \ x1  \end{array}  $
$  \begin{array}{c}  qlift \ VP \\  \swarrow \searrow \\  x0 \ x1  \end{array}  \rightarrow q \ x0  $
$  \begin{array}{c}  qright \ VP \\  \swarrow \searrow \\  x0 \ x1  \end{array}  \rightarrow q \ x1  $
$q \ PRO \rightarrow PRO$
$q \ VB \rightarrow VB$
$q \ NP \rightarrow NP$



# Complex Re-Ordering

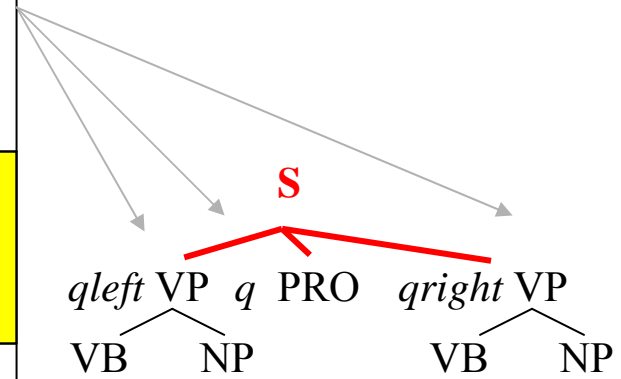
Original  
Input:



Transformation:

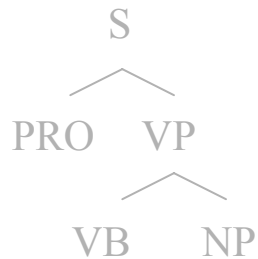
R transducer

$\begin{array}{c} q \ S \\ \swarrow \searrow \\ x0 \ x1 \end{array} \rightarrow \begin{array}{c} S \\ \swarrow \downarrow \searrow \\ qlift \ x1 \quad q \ x0 \quad qright \ x1 \end{array}$
$\begin{array}{c} qlift \ VP \\ \swarrow \searrow \\ x0 \ x1 \end{array} \rightarrow q \ x0$
$\begin{array}{c} qright \ VP \\ \swarrow \searrow \\ x0 \ x1 \end{array} \rightarrow q \ x1$
$q \ PRO \rightarrow PRO$
$q \ VB \rightarrow VB$
$q \ NP \rightarrow NP$



# Complex Re-Ordering

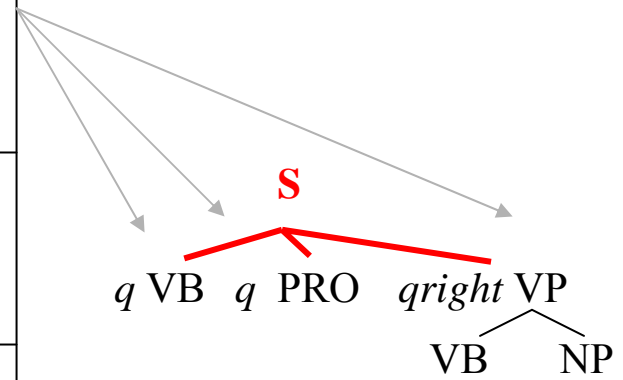
Original  
Input:



Transformation:

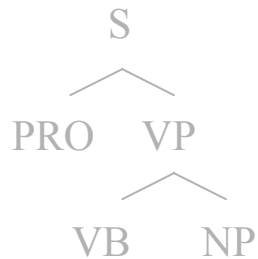
R transducer

$  \begin{array}{c}  q \ S \\  \swarrow \searrow \\  x0 \ x1  \end{array}  \rightarrow  \begin{array}{c}  S \\  \swarrow \quad \downarrow \quad \searrow \\  qlift \ x1 \quad q \ x0 \quad qright \ x1  \end{array}  $
$  \begin{array}{c}  qlift \ VP \\  \swarrow \searrow \\  x0 \ x1  \end{array}  \rightarrow q \ x0  $
$  \begin{array}{c}  qright \ VP \\  \swarrow \searrow \\  x0 \ x1  \end{array}  \rightarrow q \ x1  $
$q \ PRO \rightarrow PRO$
$q \ VB \rightarrow VB$
$q \ NP \rightarrow NP$



# Complex Re-Ordering

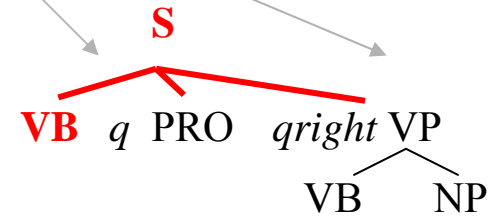
Original  
Input:



Transformation:

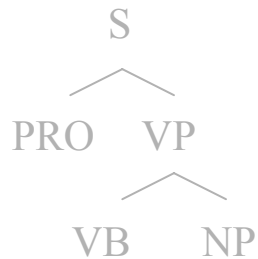
R transducer

$  \begin{array}{c}  q \ S \\  \swarrow \searrow \\  x0 \ x1  \end{array}  \rightarrow  \begin{array}{c}  S \\  \swarrow \downarrow \searrow \\  qlift \ x1 \ \ q \ x0 \ \ qright \ x1  \end{array}  $
$  \begin{array}{c}  qlift \ VP \\  \swarrow \searrow \\  x0 \ x1  \end{array}  \rightarrow q \ x0  $
$  \begin{array}{c}  qright \ VP \\  \swarrow \searrow \\  x0 \ x1  \end{array}  \rightarrow q \ x1  $
$q \ PRO \rightarrow PRO$
$q \ VB \rightarrow VB$
$q \ NP \rightarrow NP$



# Complex Re-Ordering

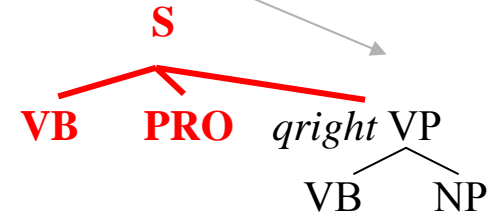
Original  
Input:



Transformation:

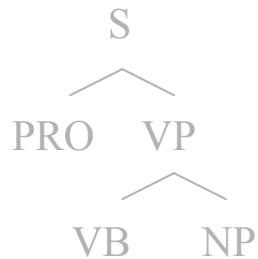
R transducer

$  \begin{array}{c}  q \ S \\  \swarrow \searrow \\  x0 \ x1  \end{array}  \rightarrow  \begin{array}{c}  S \\  \swarrow \downarrow \searrow \\  qlift \ x1 \quad q \ x0 \quad qright \ x1  \end{array}  $
$  \begin{array}{c}  qlift \ VP \\  \swarrow \searrow \\  x0 \ x1  \end{array}  \rightarrow q \ x0  $
$  \begin{array}{c}  qright \ VP \\  \swarrow \searrow \\  x0 \ x1  \end{array}  \rightarrow q \ x1  $
$q \ PRO \rightarrow PRO$
$q \ VB \rightarrow VB$
$q \ NP \rightarrow NP$



# Complex Re-Ordering

Original  
Input:



Transformation:

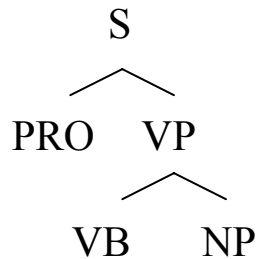
R transducer

$\begin{array}{c} q \ S \\ \swarrow \searrow \\ x0 \ x1 \end{array} \rightarrow \begin{array}{c} S \\ \swarrow \quad \downarrow \quad \searrow \\ qlift \ x1 \quad q \ x0 \quad qright \ x1 \end{array}$
$\begin{array}{c} qlift \ VP \\ \swarrow \searrow \\ x0 \ x1 \end{array} \rightarrow q \ x0$
$\begin{array}{c} qright \ VP \\ \swarrow \searrow \\ x0 \ x1 \end{array} \rightarrow q \ x1$
$q \ PRO \rightarrow PRO$
$q \ VB \rightarrow VB$
$q \ NP \rightarrow NP$



# Complex Re-Ordering

Original  
Input:



R transducer

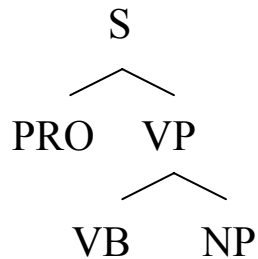
$  \begin{array}{c}  q \ S \\  \swarrow \searrow \\  x0 \ x1  \end{array}  \rightarrow  \begin{array}{c}  S \\  \swarrow \downarrow \searrow \\  qlift \ x1 \quad q \ x0 \quad qright \ x1  \end{array}  $
$  \begin{array}{c}  qlift \ VP \\  \swarrow \searrow \\  x0 \ x1  \end{array}  \rightarrow q \ x0  $
$  \begin{array}{c}  qright \ VP \\  \swarrow \searrow \\  x0 \ x1  \end{array}  \rightarrow q \ x1  $
$q \ PRO \rightarrow PRO$
$q \ VB \rightarrow VB$
$q \ NP \rightarrow NP$

Final  
Output:



# Extended Left-Hand Side: xR

Original  
Input:



xR transducer

$q$ PRO	$\rightarrow$	PRO
$q$ VB	$\rightarrow$	VB
$q$ NP	$\rightarrow$	NP

Final  
Output:

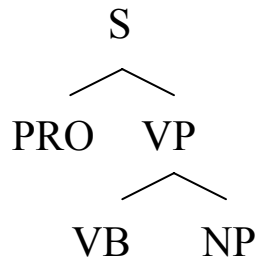


Mentioned already in Section 4, Rounds 1970.  
Not defined or used in proofs.

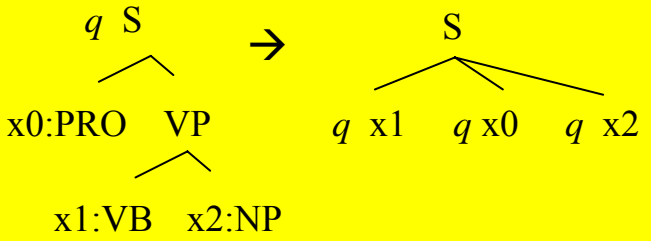


# Extended Left-Hand Side: xR

Original  
Input:



xR transducer

		
<i>q</i> PRO	→	PRO
<i>q</i> VB	→	VB
<i>q</i> NP	→	NP

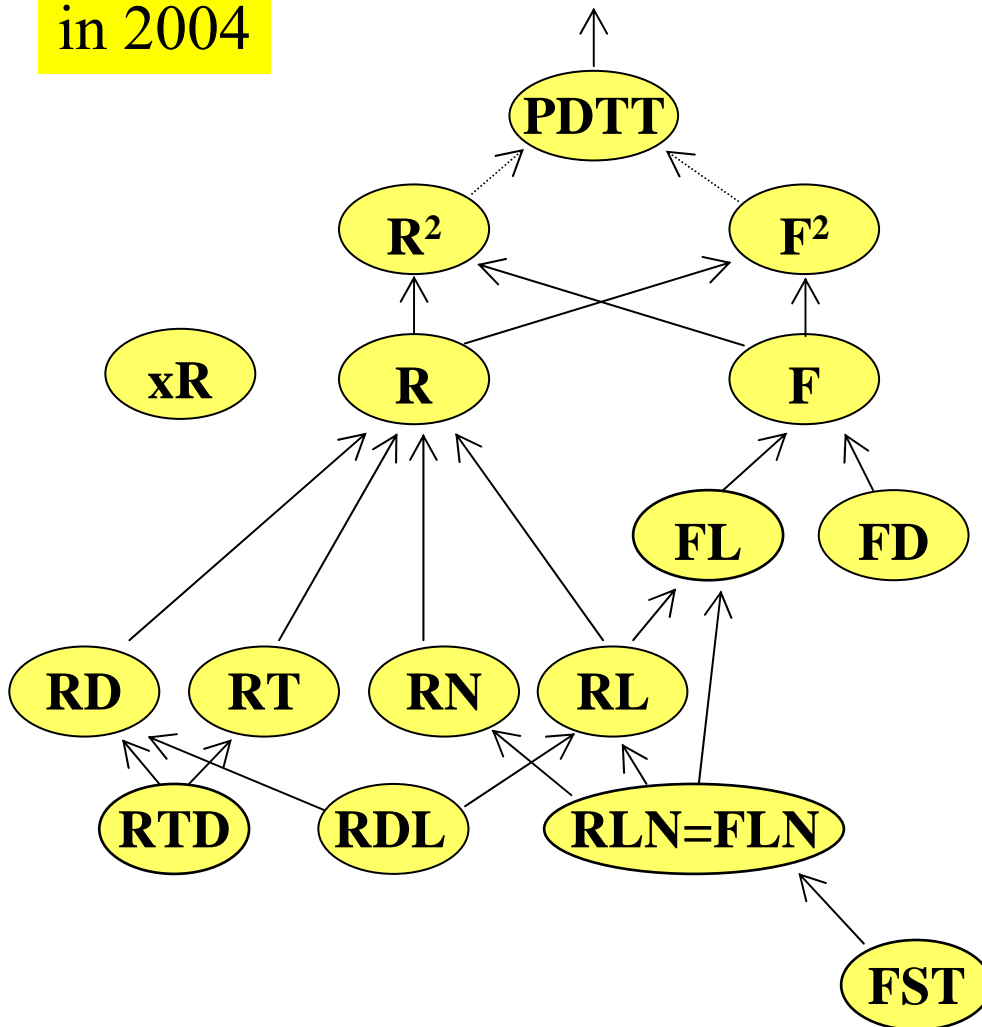
Final  
Output:



NOTE: This “extended” is not the same kind of “extended” as Thatcher’s [1967] Extended Context Free Grammar (ECFG)! Is there such a thing as an ER transducer that works over its children one by one??

# Tree Transducer Classes

Picture  
in 2004



Abbreviations:

**R** top-down tree transducer

**F** bottom-up tree transducer

**L** linear (non-copying)

**N** non-deleting

**D** deterministic

**T** total (non-rejecting)

**x** extended left-hand side

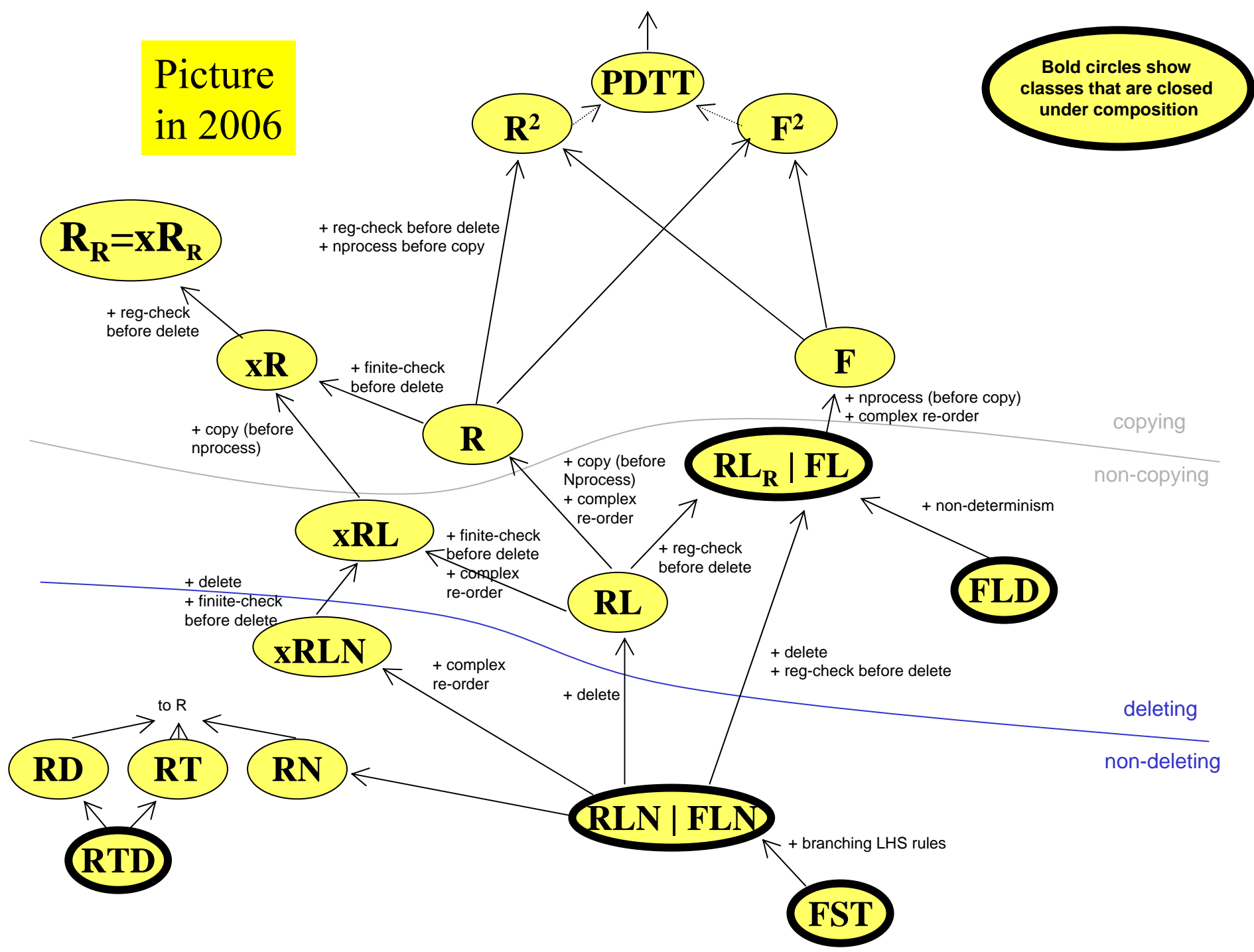
**PDDT** push-down transducer

# Open Question for NLP

- Is there a tree automata framework that does what NLP needs?
  - has power enough to capture needed NLP grammar and execute needed NLP transformations
    - $S(\text{PRO}, \text{VP}(\text{V}, \text{NP})) \rightarrow S(\text{V PRO NP})$
  - generalizes well-understood string automata framework/tools
    - “if we have Tiburon, we should be able to throw away Carmel”
  - transducers are closed under composition
    - for modular system construction
  - transducers preserve regularity
    - if you put a tree through, an RTG should come out
  - admits efficient algorithms for forward application, k-best paths, EM training
- Very important -- starting to get some answers!

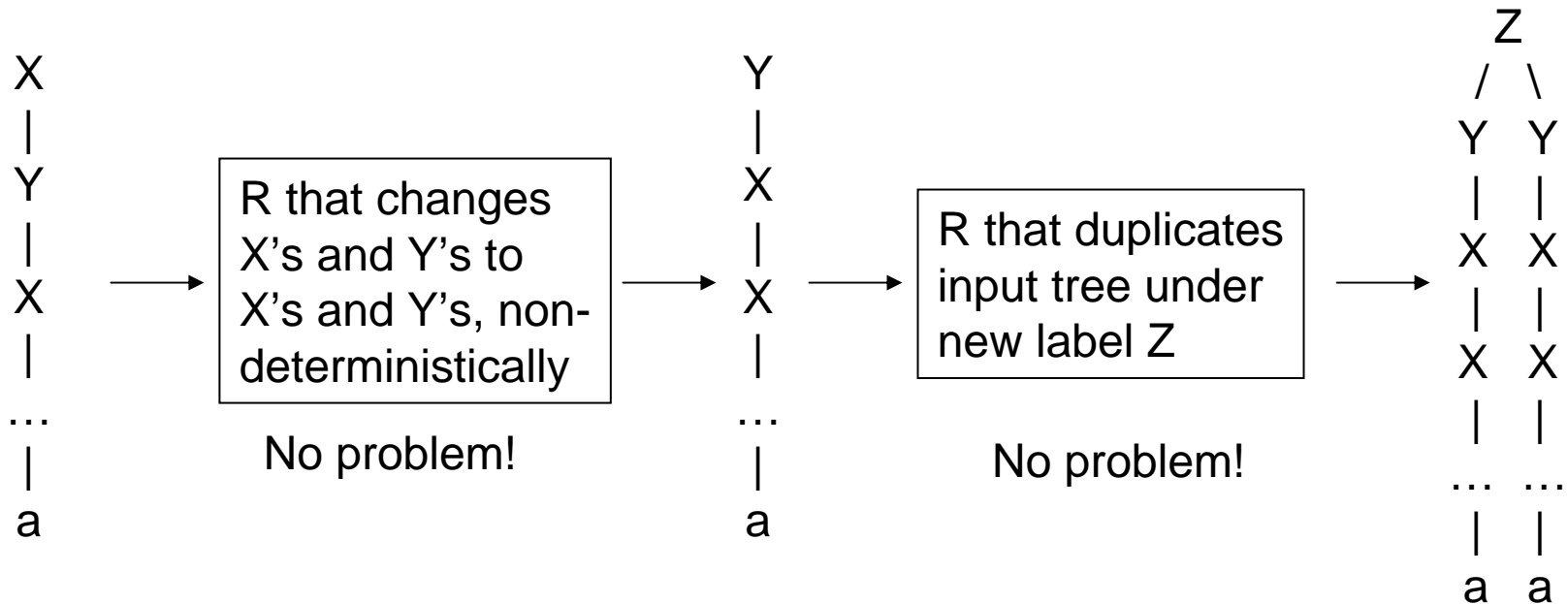
Picture  
in 2006

Bold circles show  
classes that are closed  
under composition



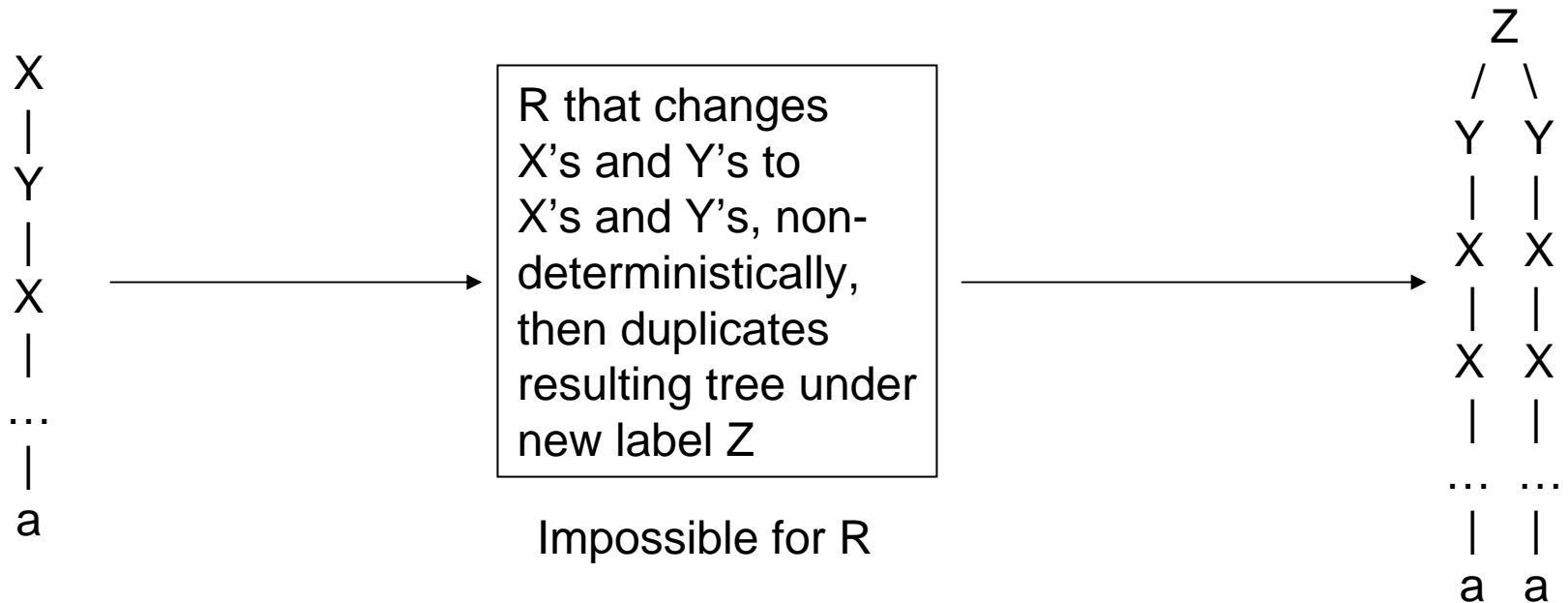
# Composability of R

- Deterministic R-transducers are composable.
- Non-deterministic R transducers are not!



# Composability of R

- Deterministic R-transducers are composable.
- Non-deterministic R transducers are not!



# Properties of Transducer Classes

String World

Tree World

	<b>FST</b>	<b>R</b>	<b>RL</b>	<b>RLN=FLN</b>	<b>F</b>	<b>FL</b>
Closed under composition	YES	NO	NO	YES	NO	YES
Closed under intersection	NO	NO	NO	NO	NO	NO
Image of tree (or finite forest) is:	RSL	RTL	RTL	RTL	<i>Not RTL</i>	RTL
Image of RTL is:	RSL	<i>Not RTL</i>	RTL	RTL	<i>Not RTL</i>	RTL
Inverse image of tree is:	RSL	RTL	RTL	RTL	RTL	RTL
Inverse image of RTL is:	RSL	RTL	RTL	RTL	RTL	RTL
Efficiently trainable	YES	YES	YES	YES		

(references: Hopcroft & Ullman 79, Gécseg & Steinby 84, Baum & Welch 71, Graehl & Knight 04)

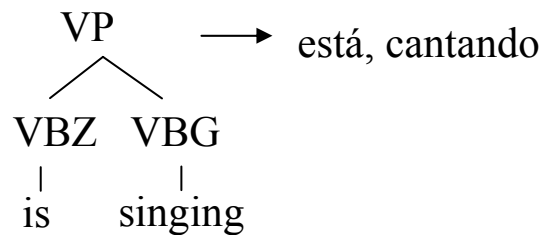
# Are Tree Transducers Expressive Enough?

- MT model (Yamada & Knight, 2001) can be cast as a tree transducer.
  - Details in [Graehl & Knight, 2004]
- Good for elucidating the model, as in:
  - Knight & Al-Onaizan, 1998 – IBM Model 3
  - Kumar & Byrne, 2003 – Phrase-based SMT
- With tree transducer framework, these models can be extended with additional transitions.

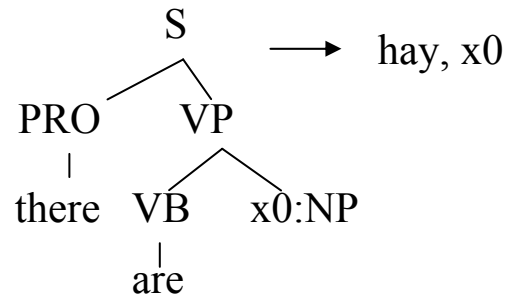


# Lots of Extensions Possible

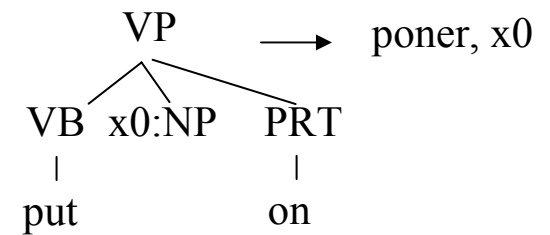
## Phrasal Translation



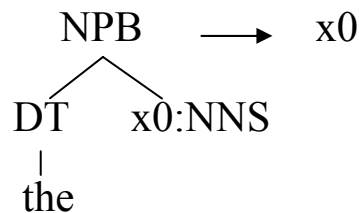
## Non-constituent Phrases



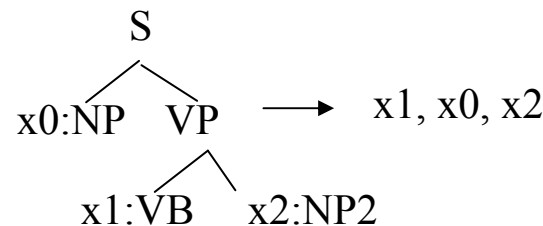
## Non-contiguous Phrases



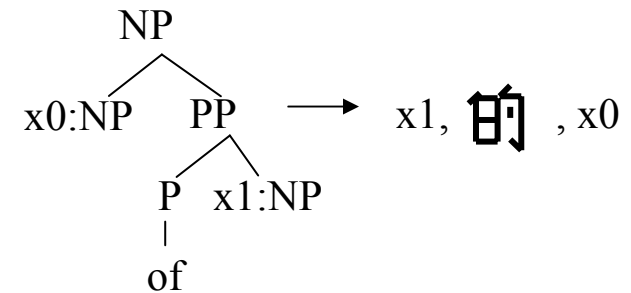
## Context-Sensitive Word Insertion



## Multilevel Re-Ordering



## Lexicalized Re-Ordering

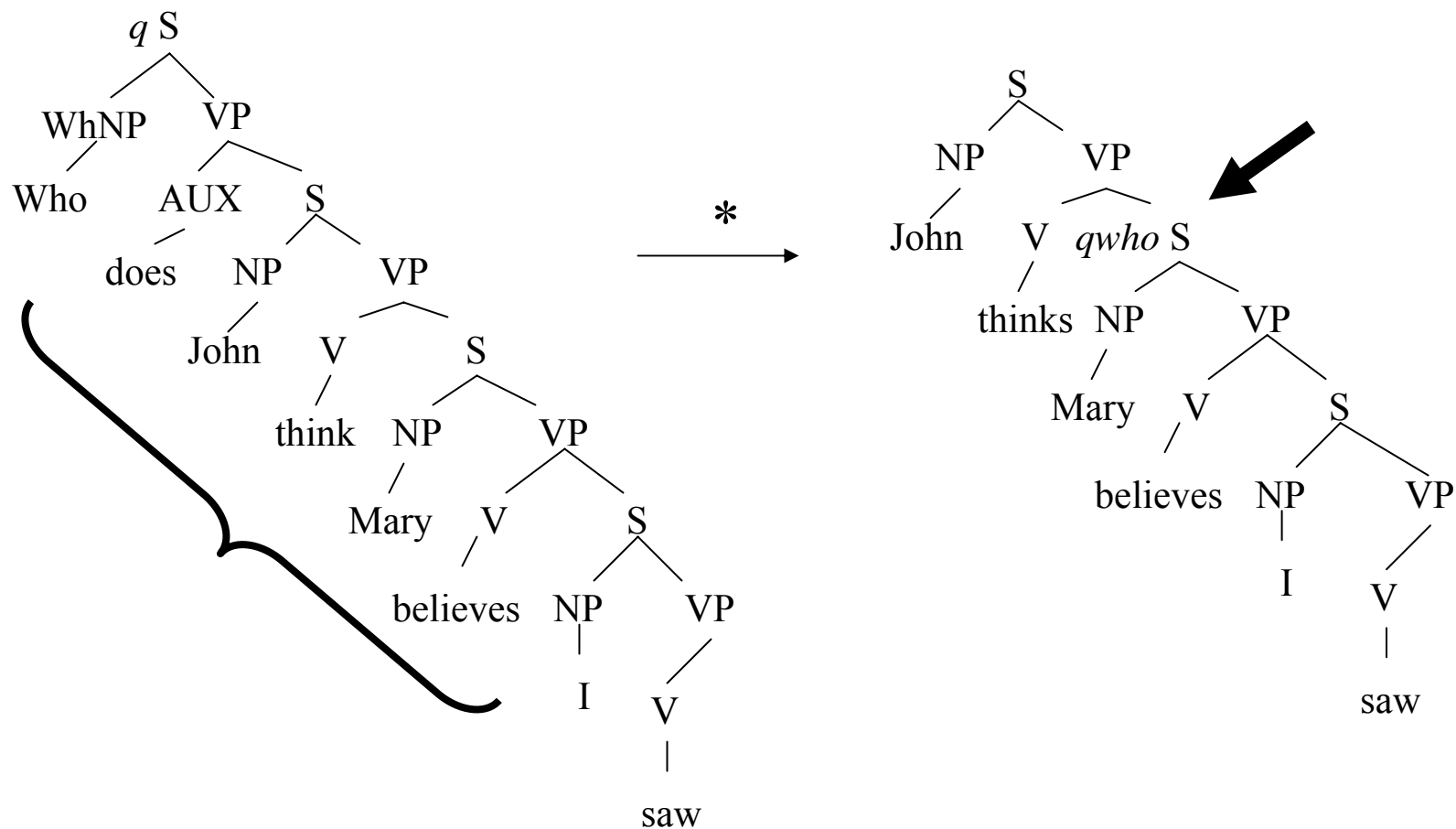


These rules can be learned from parallel corpora  
Galley, Hopkins, Knight, Marcu [2004]

# Limitations of R Model

## Long-distance Re-Ordering

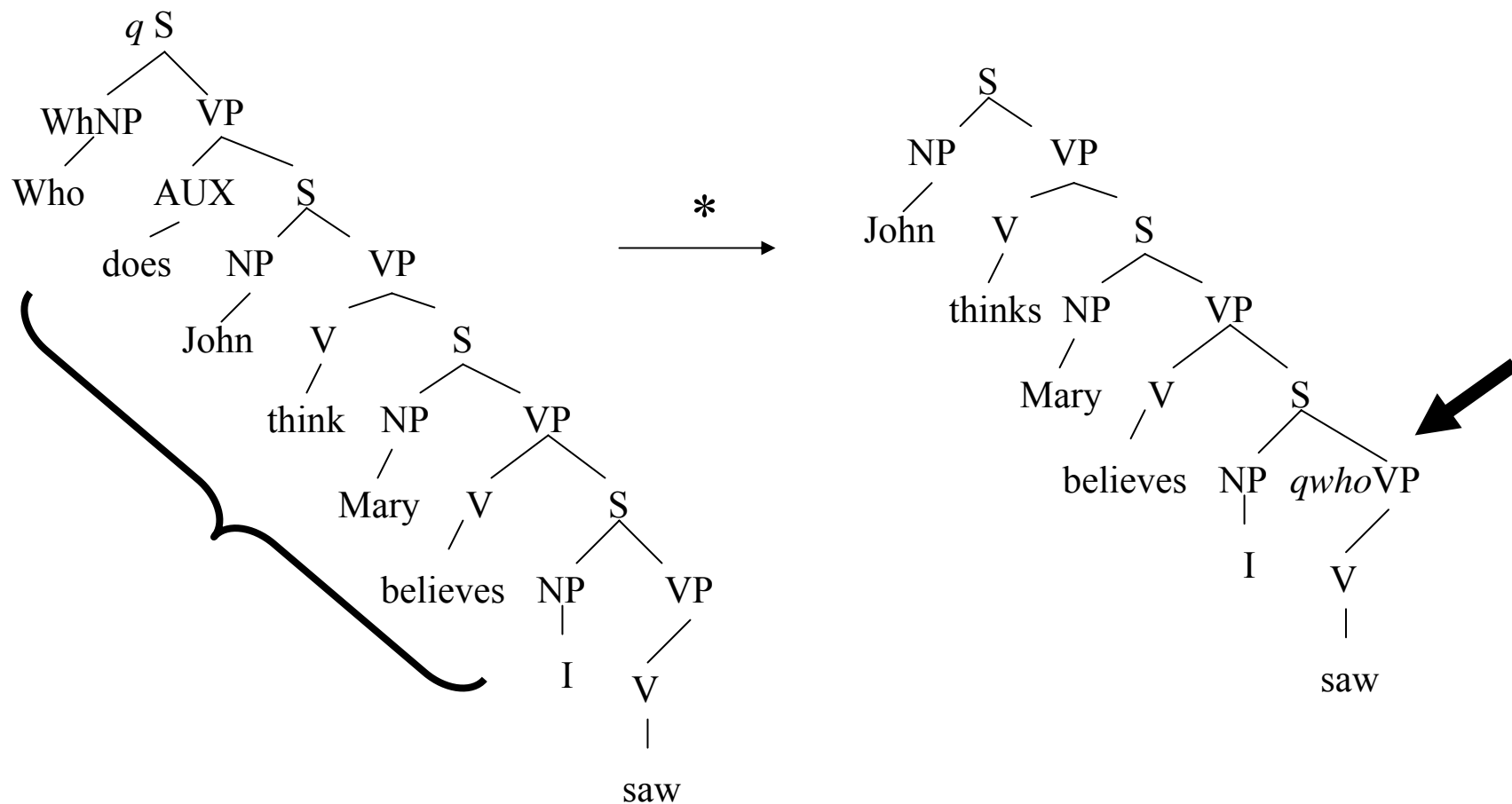
Who does John think Mary believes I saw? → John thinks Mary believes I saw *who*?



# Limitations of R Model

## Long-distance Re-Ordering

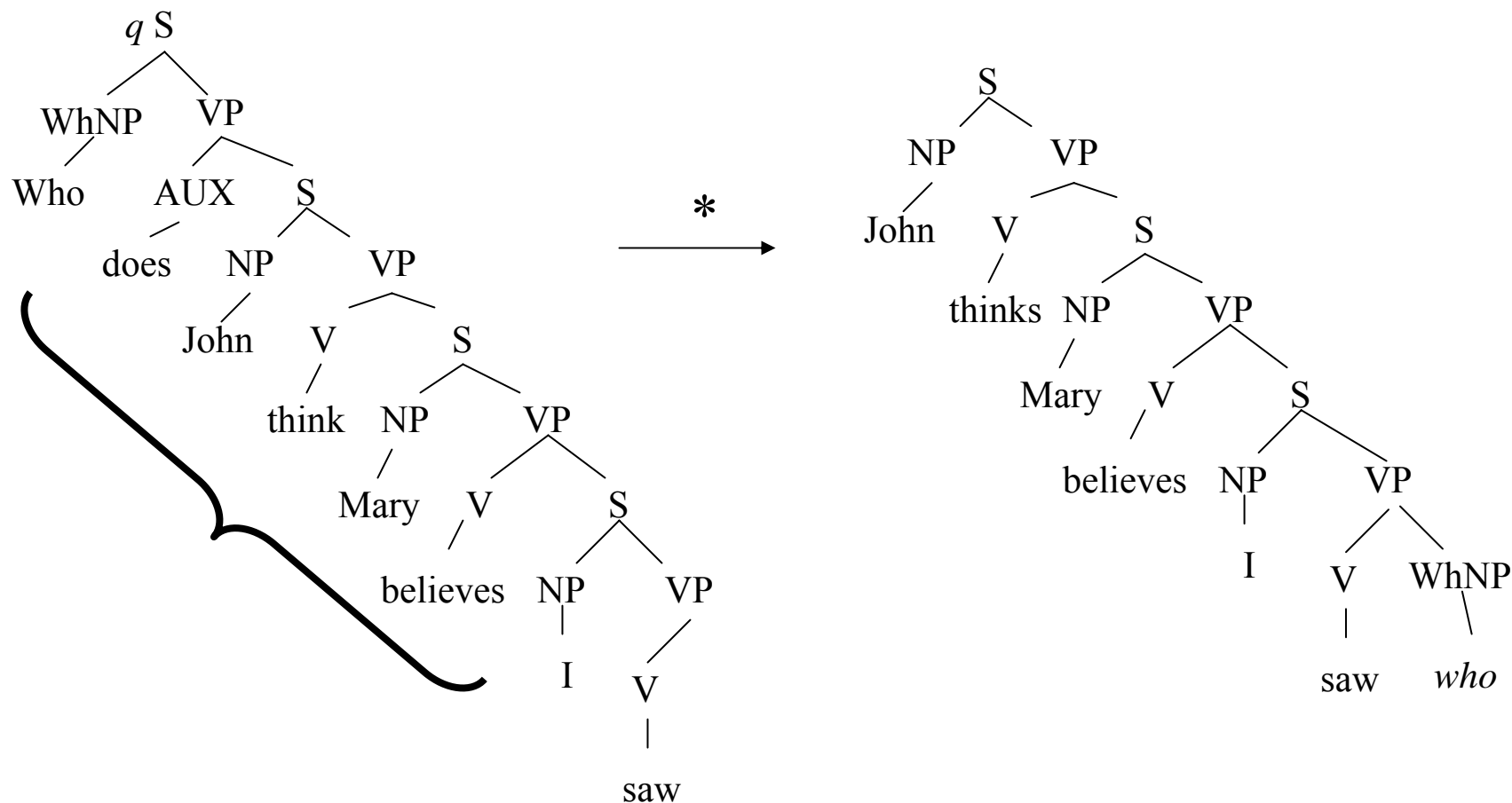
Who does John think Mary believes I saw? → John thinks Mary believes I saw *who*?



# Limitations of R Model

## Long-distance Re-Ordering

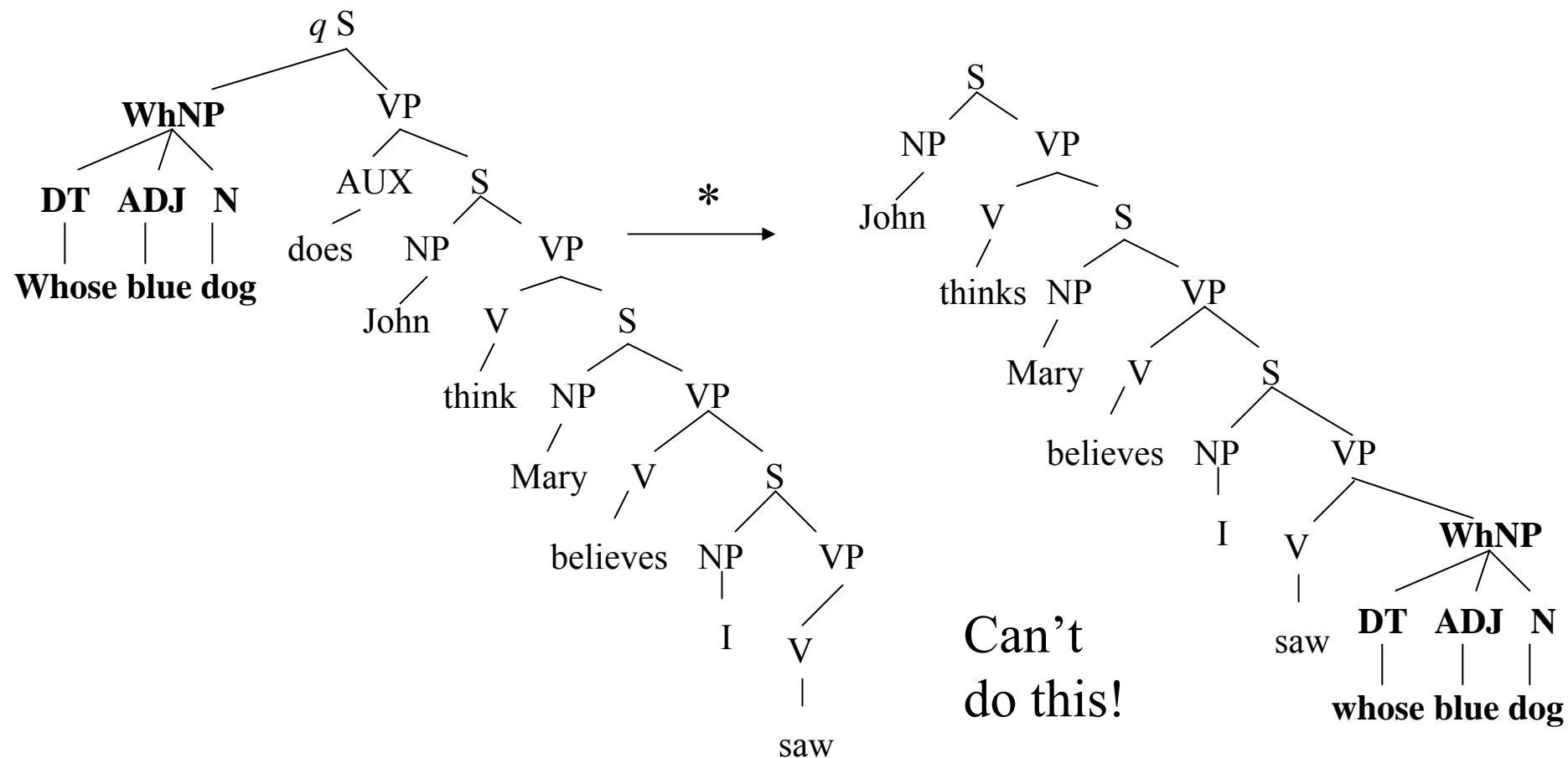
Who does John think Mary believes I saw? → John thinks Mary believes I saw *who*?



# Limitations of R Model

## Long-distance Re-Ordering

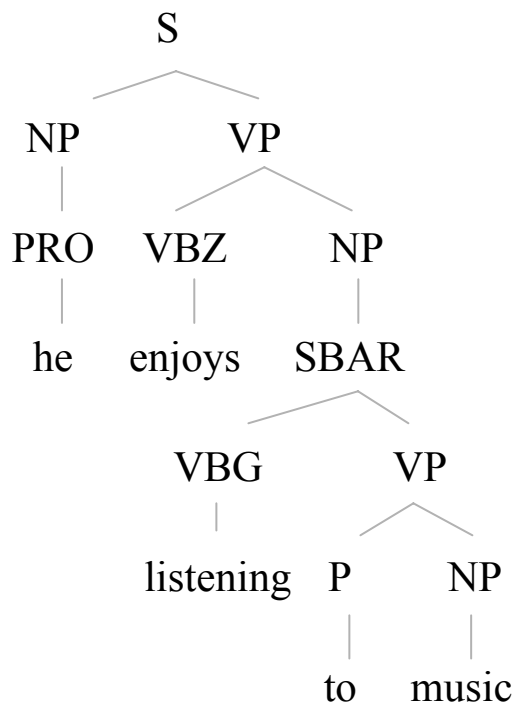
**Whose blue dog** does John think Mary believes I saw? → John thinks Mary believes I saw **whose blue dog**?



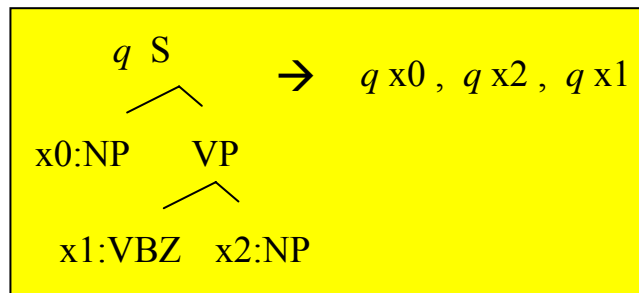
# Tree-to-String: xRS

Syntax-directed translation for compilers (Aho & Ullman, 1971)

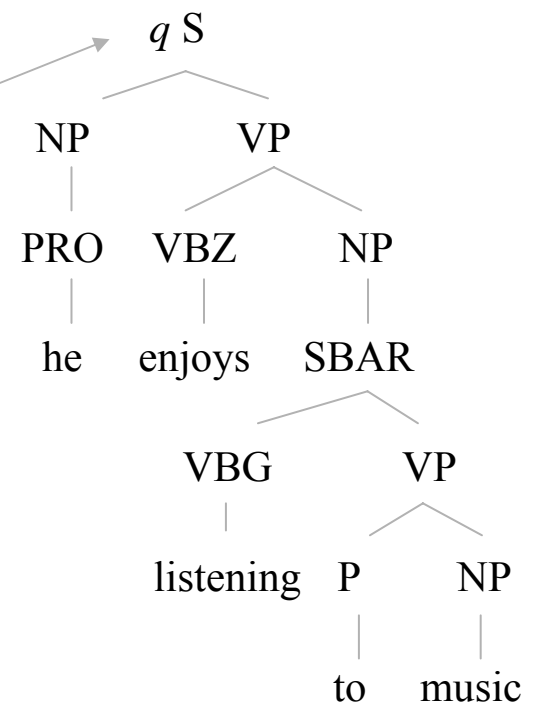
Original input:



xRS transducer



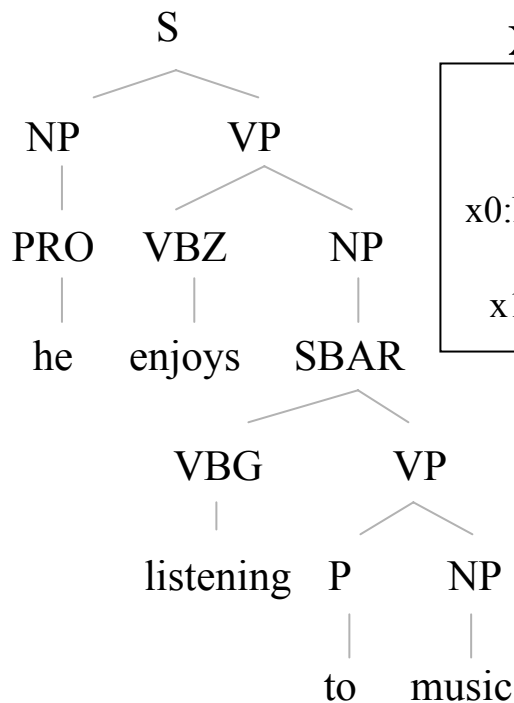
Transformation:



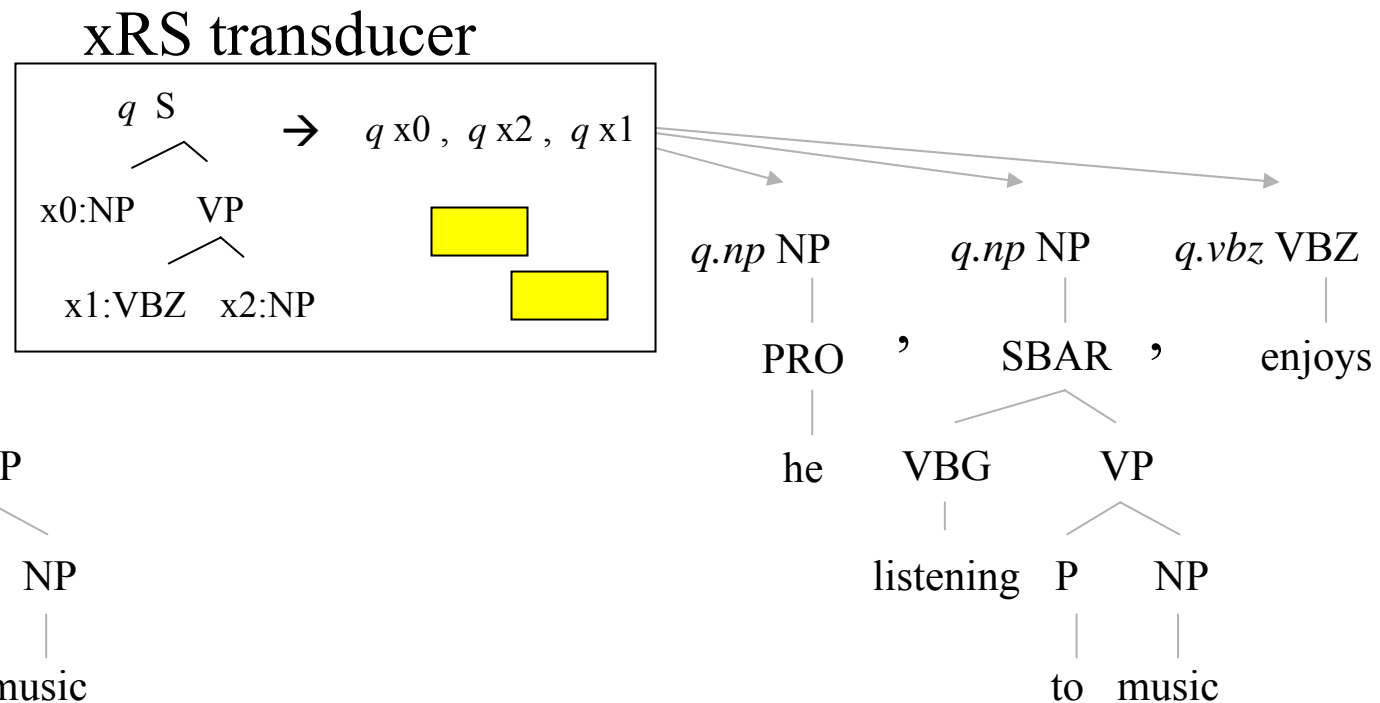
# Tree-to-String: xRS

Syntax-directed translation for compilers (Aho & Ullman, 1971)

Original input:



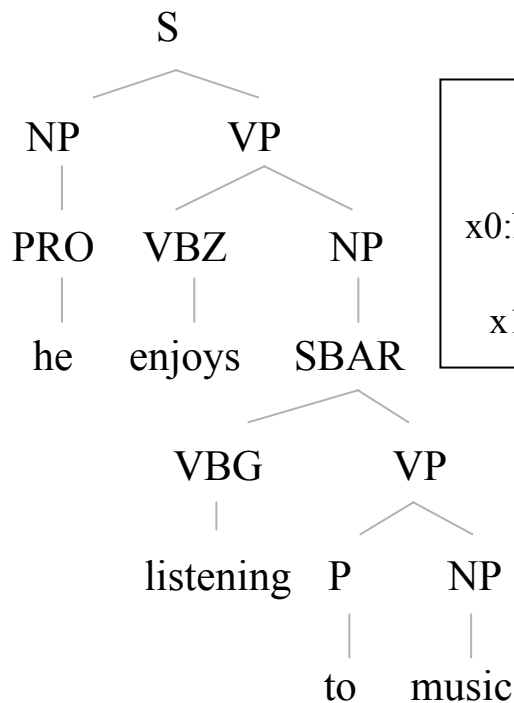
Transformation:



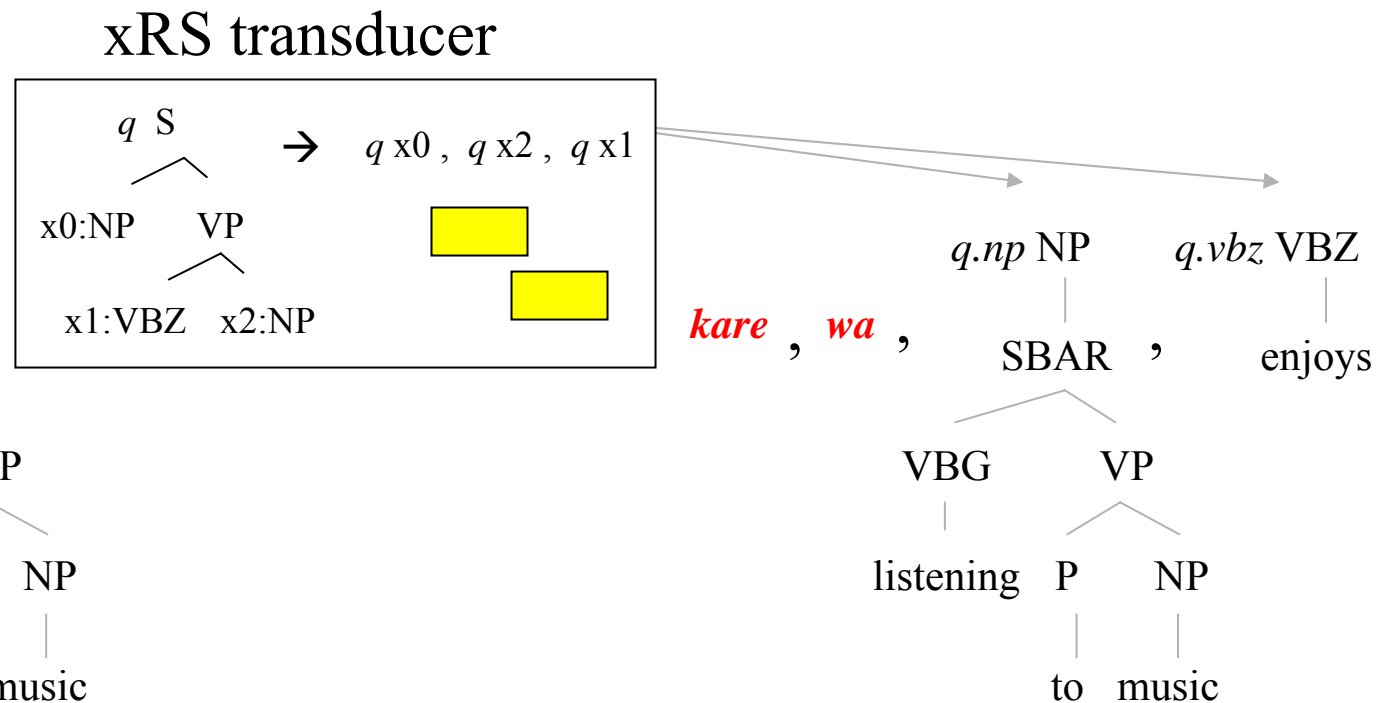
# Tree-to-String: xRS

Syntax-directed translation for compilers (Aho & Ullman, 1971)

Original input:



Transformation:

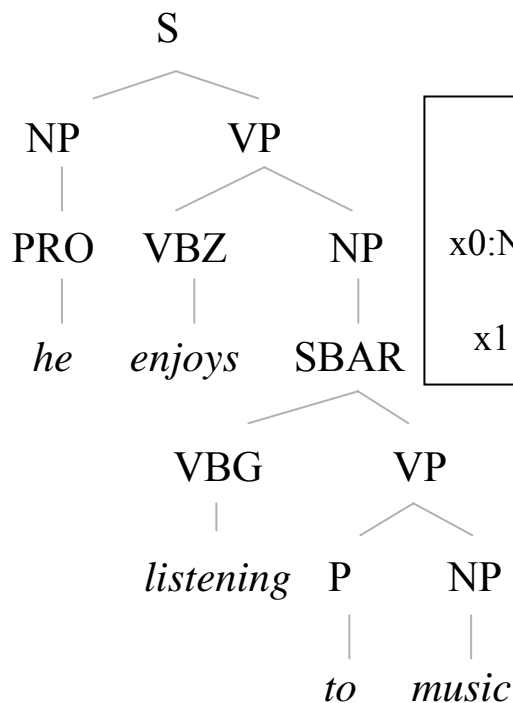




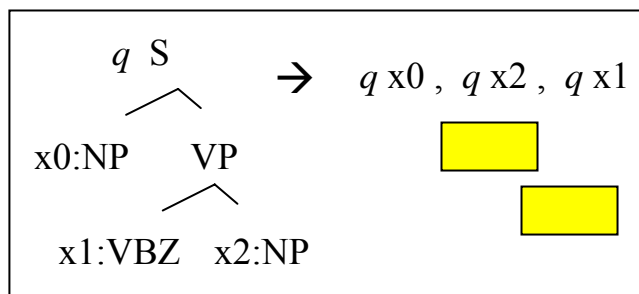
# Tree-to-String: xRS

Syntax-directed translation for compilers (Aho & Ullman, 1971)

Original input:



xRS transducer



Final output:

*kare, wa, ongaku, o, kiku, no, ga, daisuki, desu*

Foundation of ISI Syntax-Based  
Machine Translation System  
(100m rules learned from corpora)

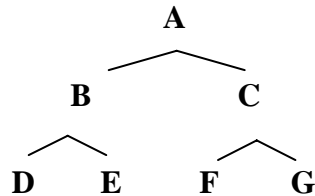
# The Training Problem

## [Graehl & Knight, 2004]

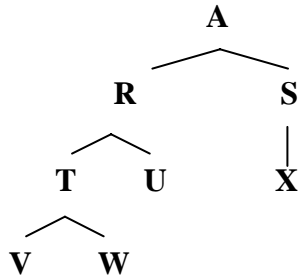
- Given:
  - an xR transducer with a set of non-deterministic rules ( $r_1 \dots r_m$ )
  - a training corpus of **input-tree/output-tree** pairs
- Produce:
  - conditional probabilities  $p_1 \dots p_m$  that maximize
$$\mathbf{P}(\text{output trees} \mid \text{input trees})$$
$$= \Pi_{i,o} P(o \mid i) \quad \text{input/output trees } \langle i, o \rangle$$
$$= \Pi_{i,o} \Sigma_d P(d, o \mid i) \quad \text{derivations } d \text{ mapping } i \text{ to } o$$
$$= \Pi_{i,o} \Sigma_d \Pi_r p_r \quad \text{rules } r \text{ in derivation } d$$

# Derivations in R

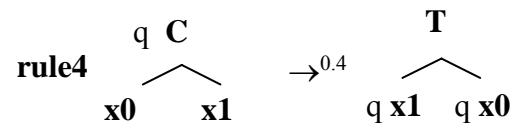
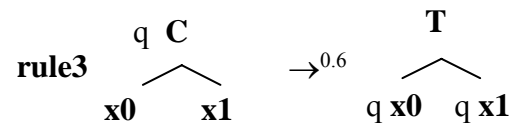
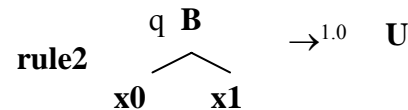
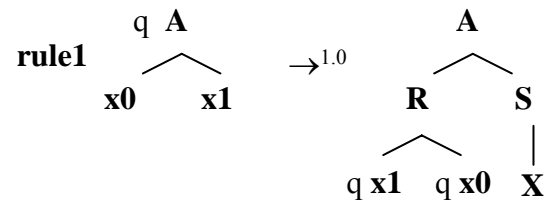
*Input tree:*



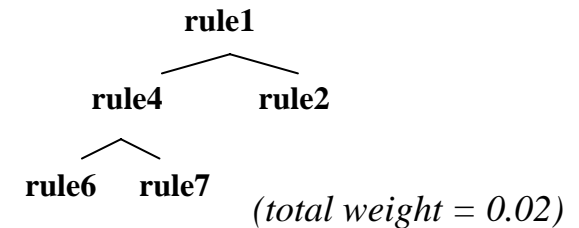
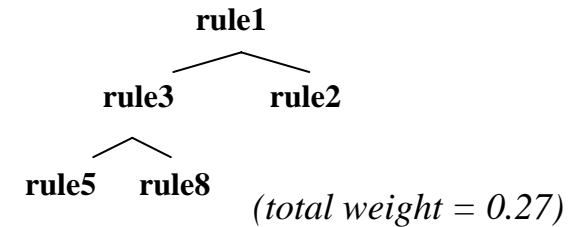
*Output tree:*



*R Transducer rules:*



*Derivations:*



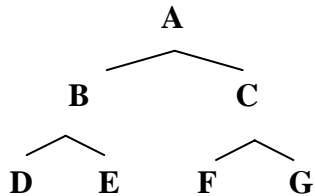
*Packed Derivations:*

```

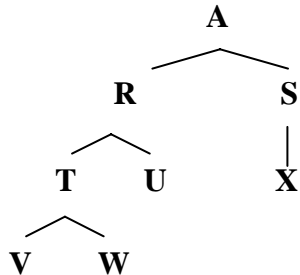
qstart   →1.0 rule1(q.1.12, q.2.11)
q.1.12   →1.0 rule2
q.2.11   →0.6 rule3(q.21.111, q.22.112)
q.2.11   →0.4 rule4(q.21.112, q.22.111)
q.21.111 →0.9 rule5
q.22.112 →0.5 rule8
q.21.112 →0.1 rule6
q.22.111 →0.5 rule7
  
```

# Derivations in R

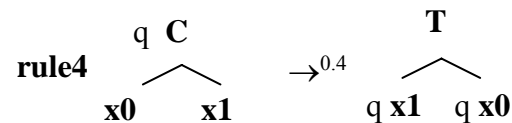
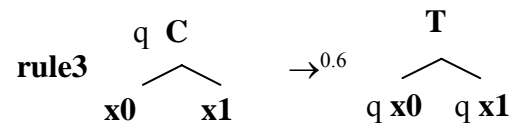
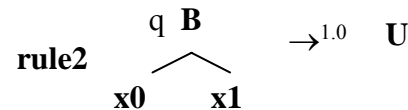
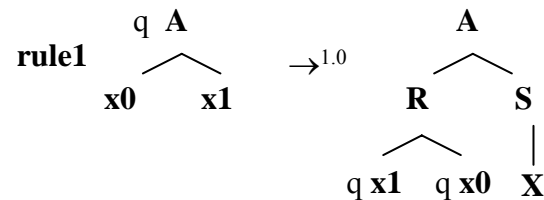
*Input tree:*



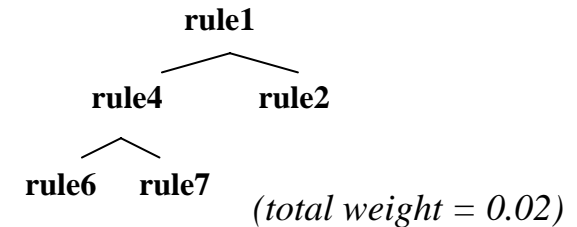
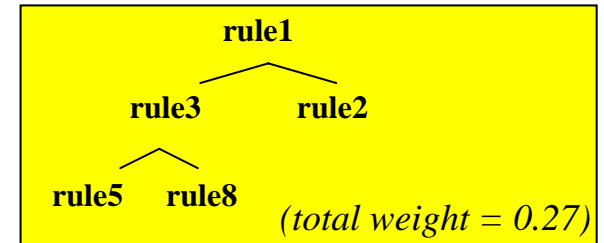
*Output tree:*



*R Transducer rules:*



*Derivations:*



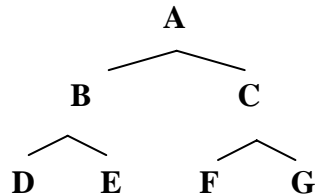
*Packed Derivations:*

```

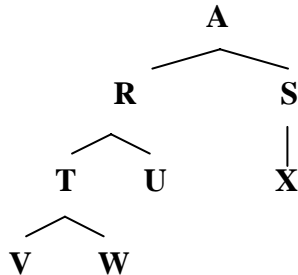
qstart   →1.0 rule1(q.1.12, q.2.11)
q.1.12   →1.0 rule2
q.2.11   →0.6 rule3(q.21.111, q.22.112)
q.2.11   →0.4 rule4(q.21.112, q.22.111)
q.21.111 →0.9 rule5
q.22.112 →0.5 rule8
q.21.112 →0.1 rule6
q.22.111 →0.5 rule7
  
```

# Derivations in R

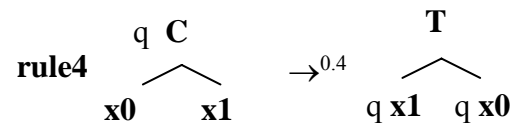
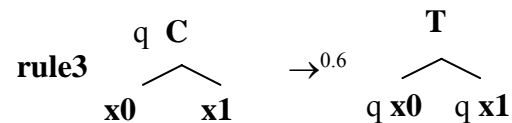
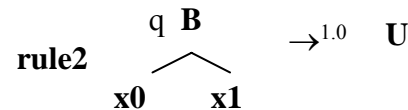
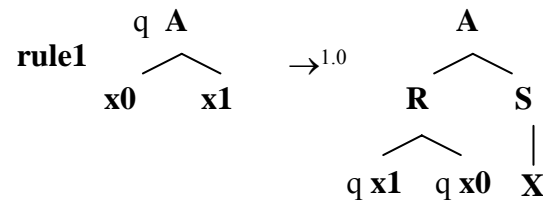
*Input tree:*



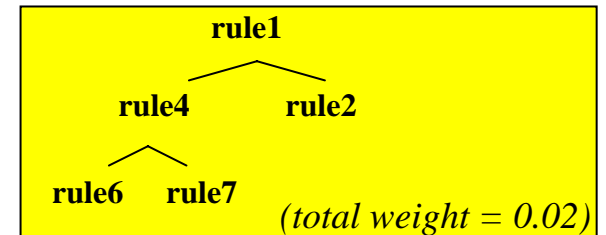
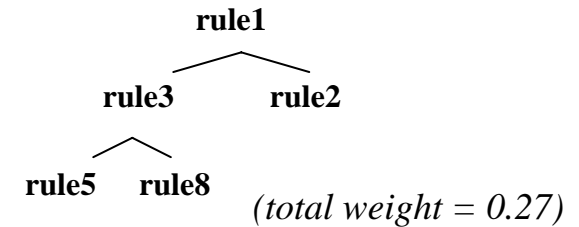
*Output tree:*



*R Transducer rules:*



*Derivations:*



*Packed Derivations:*

```

qstart  →1.0 rule1(q.1.12, q.2.11)
q.1.12  →1.0 rule2
q.2.11  →0.6 rule3(q.21.111, q.22.112)
q.2.11  →0.4 rule4(q.21.112, q.22.111)
q.21.111 →0.9 rule5
q.22.112 →0.5 rule8
q.21.112 →0.1 rule6
q.22.111 →0.5 rule7
  
```

# Naïve EM Algorithm for xR

Initialize uniform probabilities

Until tired do:

    Make zero rule count table

    For each training case  $\langle i, o \rangle$

        Compute  $P(o \mid i)$  by summing over derivations  $d$

        For each derivation  $d$  for  $\langle i, o \rangle$  */\* exponential! \*/*

            Compute  $P(d \mid i, o) = P(d, o \mid i) / P(o \mid i)$

            For each rule  $r$  used in  $d$

$\text{count}(r) += P(d \mid i, o)$

    Normalize counts to probabilities

# Packed Derivation Forest

*Build packed derivation forest* using dynamic programming  
(Yamada & Knight 2001, Eisner 2003)

Hit each triple **<q, i-node, o-node>** at most once:

For each **<q, i-node, o-node>**

For each rule **r** matching subtrees at i-node & o-node

Can **r** kick off a mapping from subtree at i-node to  
subtree at o-node?

Bottom-up, or top-down memo-ized

$O(q n^2 r)$  time

$O(q n^2 r)$  space (therefore max size of derivation forest)

# Efficient EM for xR

Initialize uniform probabilities

Until tired do:

    Make zero rule count table

    For each training case  $\langle i, o \rangle$

*Build packed derivation forest*     $O(q n^2 r)$  time/space

        Use inside-outside algorithm to collect rule counts

            Inside pass                             $O(q n^2 r)$  time/space

            Outside pass                           $O(q n^2 r)$  time/space

            Count collection pass                 $O(q n^2 r)$  time/space

    Normalize counts to probabilities (joint or conditional)

Per-example training complexity is  $O(n^2) \times$  “transducer constant” -- same as forward-backward training for string transducers (Baum & Welch, 1971).



# EM for Tree/String Transducer (xRS)

Build packed derivation forest using dynamic programming.

Hit each  $\langle \mathbf{q}, \mathbf{i\text{-}node}, \mathbf{o\text{-}span} \rangle$  at most once:

For each  $\langle q, i\text{-node}, o\text{-span} \rangle$

For each rule  $r$  (*with*  $|RHS| = k$ ) matching at  $i\text{-node}$

*For each break-up  $b$  of  $o\text{-span}$  into  $k$  pieces*

Can  $r$  kick off a mapping from subtree at  $i\text{-node}$  to substring at  $o\text{-span}$ , *via break-up  $b$* ?

$O(q n^4 r)$  time (if  $k = 2$ ).

$O(q n^3 r)$  space (therefore max size of derivation forest)

Followed by same inside-outside rule counting as for xR.

# Training: Related Work

- Generalizes MT training in Appendix of (Yamada/Knight, 2001)
  - xR training not tied to particular MT model, or to MT at all
- Generalizes forward-backward HMM training (Baum/Welch, 1971)
  - Write strings vertically, and use xR training
- Generalizes inside-outside PCFG training (Lari/Young, 1990)
  - Attach fixed input tree to each “output” string, and use xRS training
- Generalizes synchronous tree-substitution grammar training (Eisner, 2003)
  - xR and xRS allow copying of subtrees

# Best-Tree Search

- Given an input, what's the most likely output?
- FSA
  - Best path through FSA can be found with [Dijkstra, 1959]
  - Carmel lists k-best paths (incl. cycles), using [Eppstein 1999]
    - $O(n + m \log m + k \log k)$  running time
- Regular Tree Grammar (RTG)
  - Knuth [1977] solved the related *grammar problem*: efficiently extracts best tree from a forest
  - Can be viewed as hypergraph search
  - Huang & Chiang [2005] give an algorithm for k-best trees

# Alternative Formulation of Transducers

- Synchronous Tree Grammars
  - Synchronous Tree Substitution Grammar (STSG)  
[Shieber, 1992; Eisner 2003 for training]
$$\left\{ \begin{array}{l} q \rightarrow S(\text{CC}(\text{if}), S(qnp, qvp), \text{CC}(\text{then}), q) \\ q \rightarrow S(qnp, \text{ifthen}, q, qvp) \end{array} \right\}$$
  - Synchronous Tree Adjoining Grammar (STAG)  
[Shieber & Schabes, 1990]
- STSG same power as xRLN, up to a relabeling  
[Graehl, Hopkins, Knight, 2006]

# Why Worry About Formal Languages for Natural Language Processing?

- Is formal language a pretty dead field?
- They already figured out most things back in the 1960s...
- Lucky us!

# Why Worry About Formal Languages?

## Some Open Questions

- Is there an extended version of R-style transducers, with ECFG-like processing of children?
- Are there transducers that can move material across a longer distance in the tree? Are they trainable?
- Is there an automata version of TAG? What properties does it have?
- What about tree-based models that could train on string/string data? (SxRS...)
- What about “parent-less left-hand-side” rules like
  - $\text{NP}(x_0:\text{DT}, x_1:\text{JJ}, x_2:\text{NN}) \rightarrow x_0, x_1, x_2$
  - $\text{DT}(\text{the}) \rightarrow \text{el}$
  - $\text{JJ}(\text{big}) \text{NN}(\text{cheese}) \rightarrow \text{jefe}$       /\* parent-less LHS rule \*/
- Can “cloning” operation of [Gildea 03] be captured by a tree transducer?
- Is xR closed under composition if epsilon transitions are disallowed?
- Can we build tree transducer toolkits as powerful and ubiquitous as string transducer toolkits?

~~the end~~ beginning