A Generative Model for Punctuation in Dependency Parsing

Xiang Lisa Li*, Dingquan Wang*, and Jason Eisner

*** Equal Contribution**

JOHNS HOPKINS UNIVERSITY

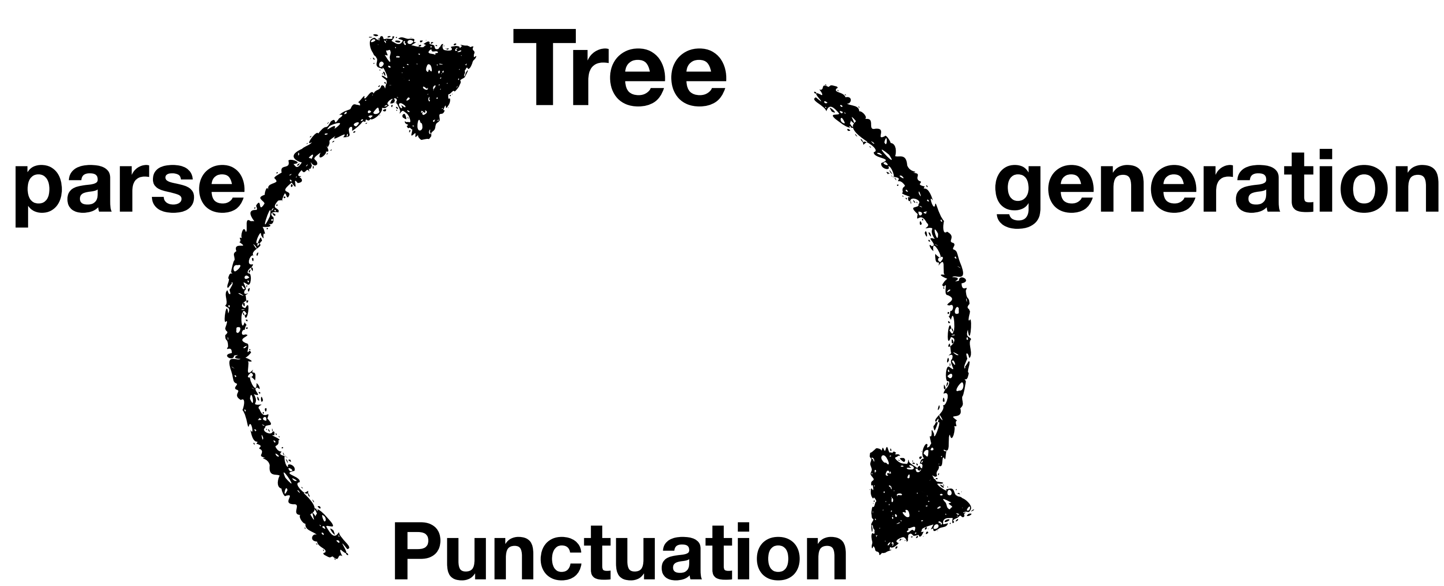
CENTER FOR LANGUAGE AND SPEECH PROCESSING

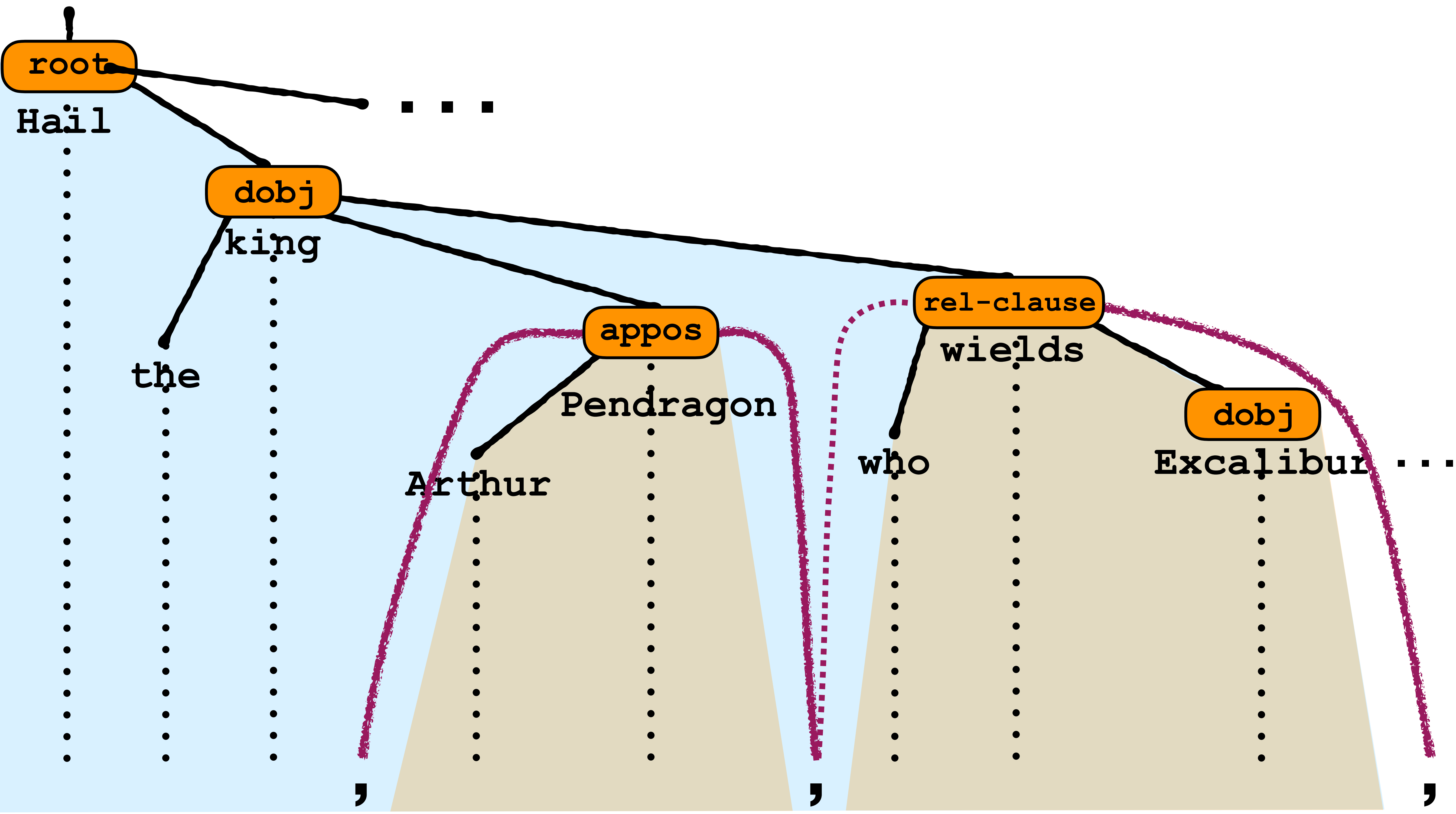
NLP has neglected punctuation

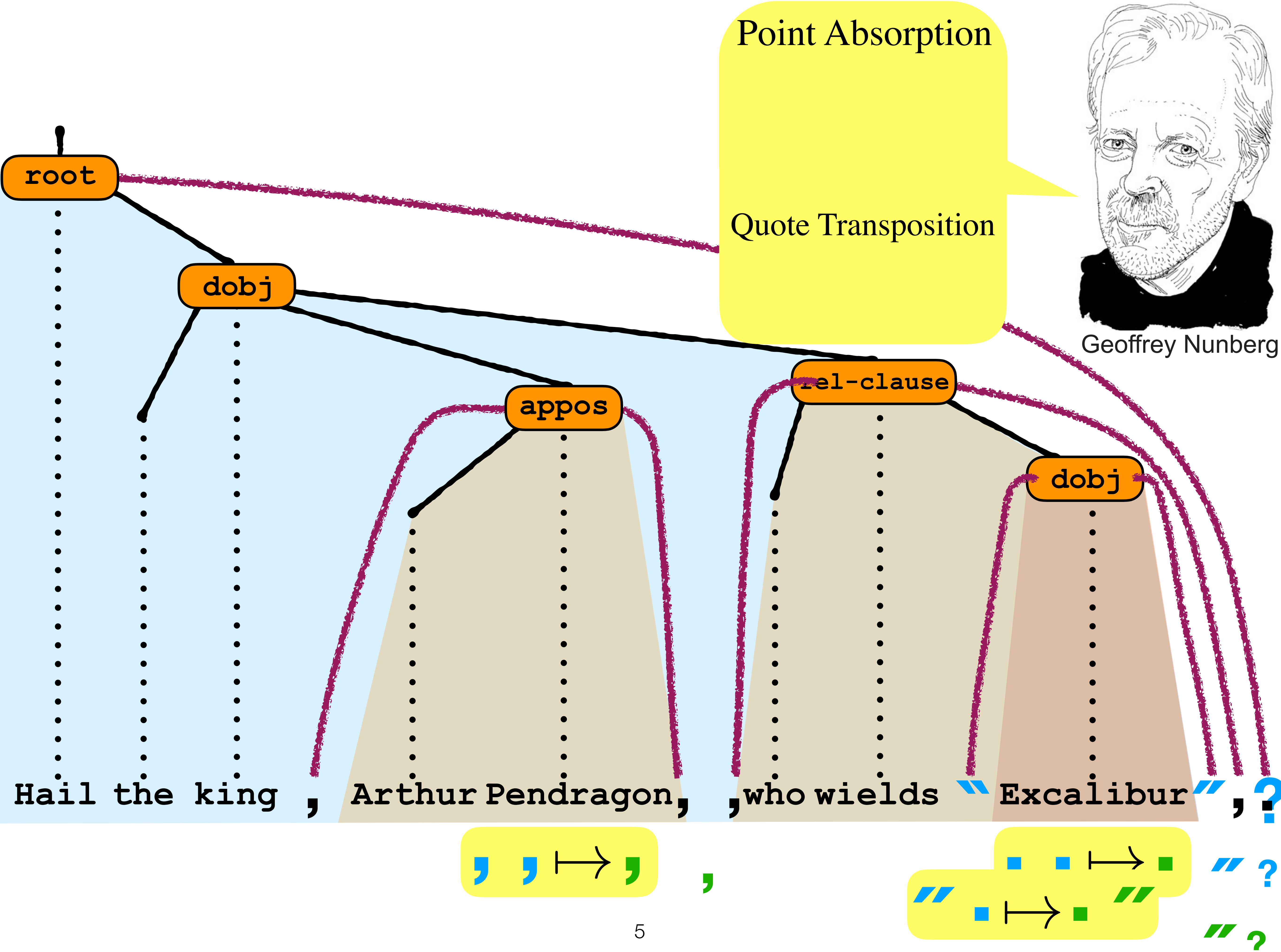
- Treebanks treat punctuation marks as ordinary tokens, but ARE THEY?
- *The Linguistics of Punctuation*, Nunberg (1990)

Punctuation is useful

Punctuation marks are correlated with prosody and the <u>syntactic tree structure</u>.



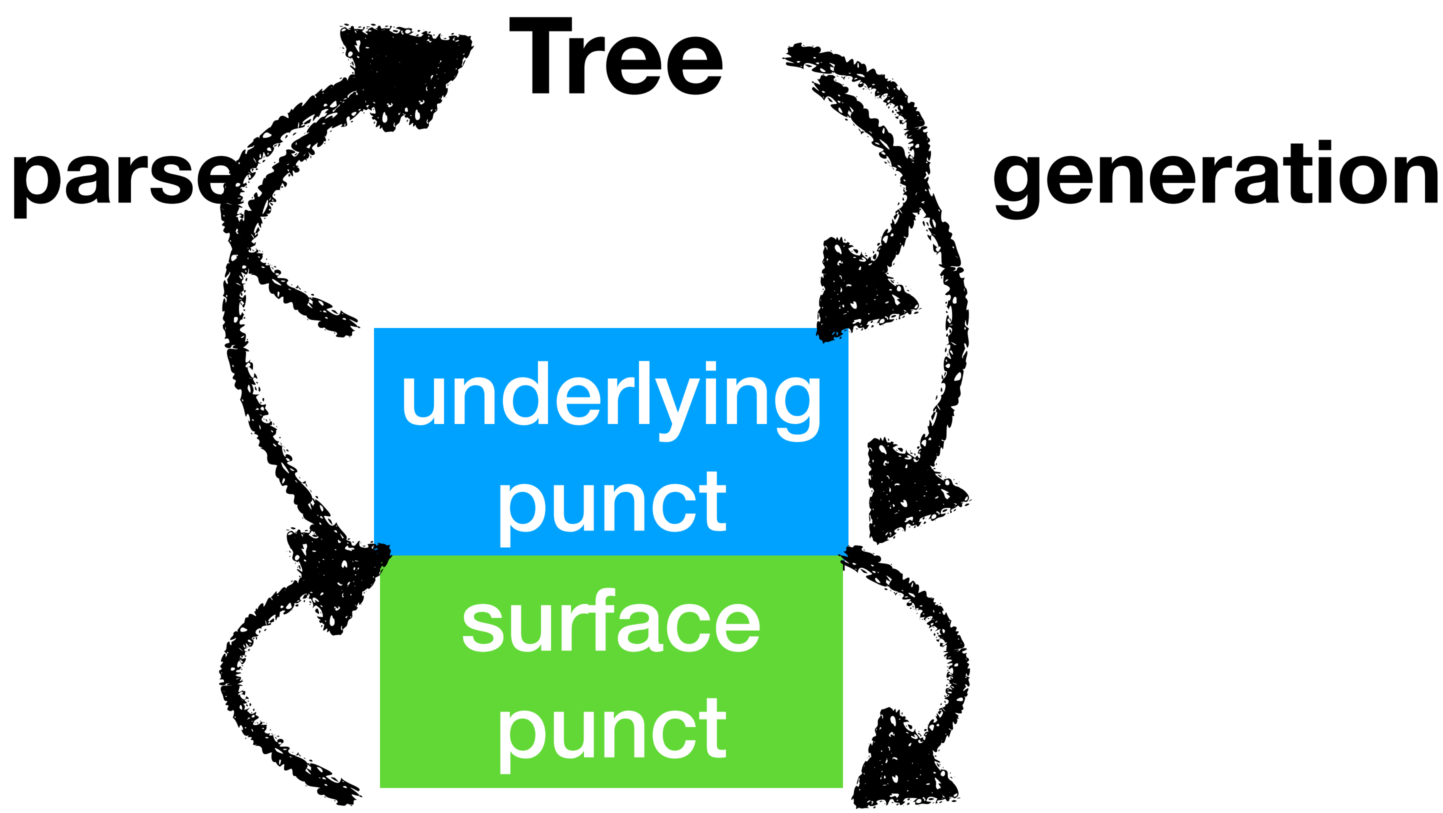

root
Hail
...
dobj
king
the
appos
Pendragon
Arthur
,
,
rel-clause
wields
who
dobj
Excalibur ...
,



Point Absorption
Quote Transposition
Geoffrey Nunberg
root
dobj
appos
rel-clause
dobj
Hail the king , Arthur Pendragon, , who wields “ Excalibur ” , ?
, , ↦ ,
. . ↦ .
” . ↦ . ”
” ?
” ?

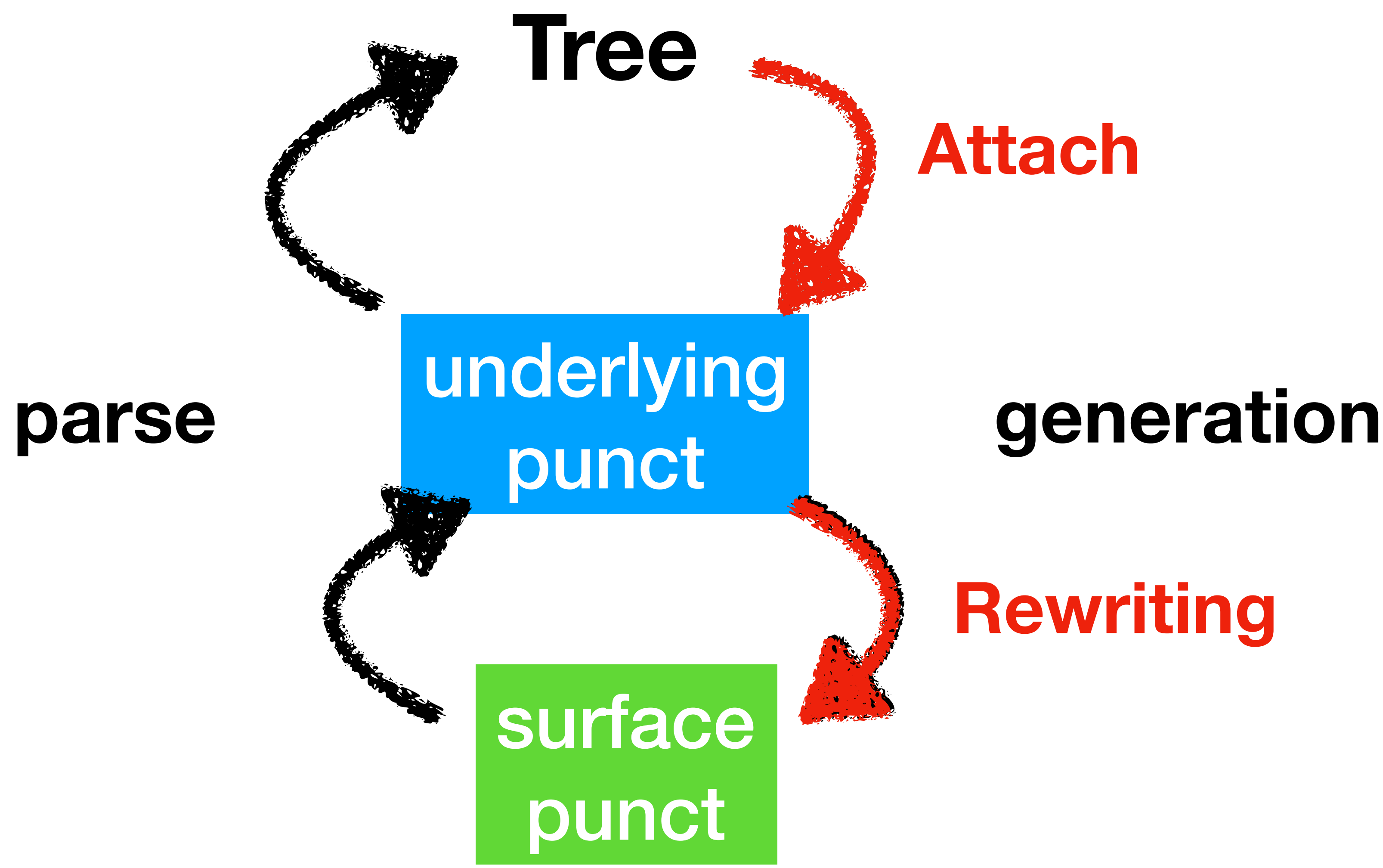
Summary…

- Punctuation marks are not words.
  - Just as prosody is not words.
- They do not belong in the tree.
- Only <u>underlying</u> punctuation marks are in the tree, where they surround certain phrases.



The surface punctuation has a non-obvious correlation with the tree.

The underlying punctuation has a more direct correlation with the tree.

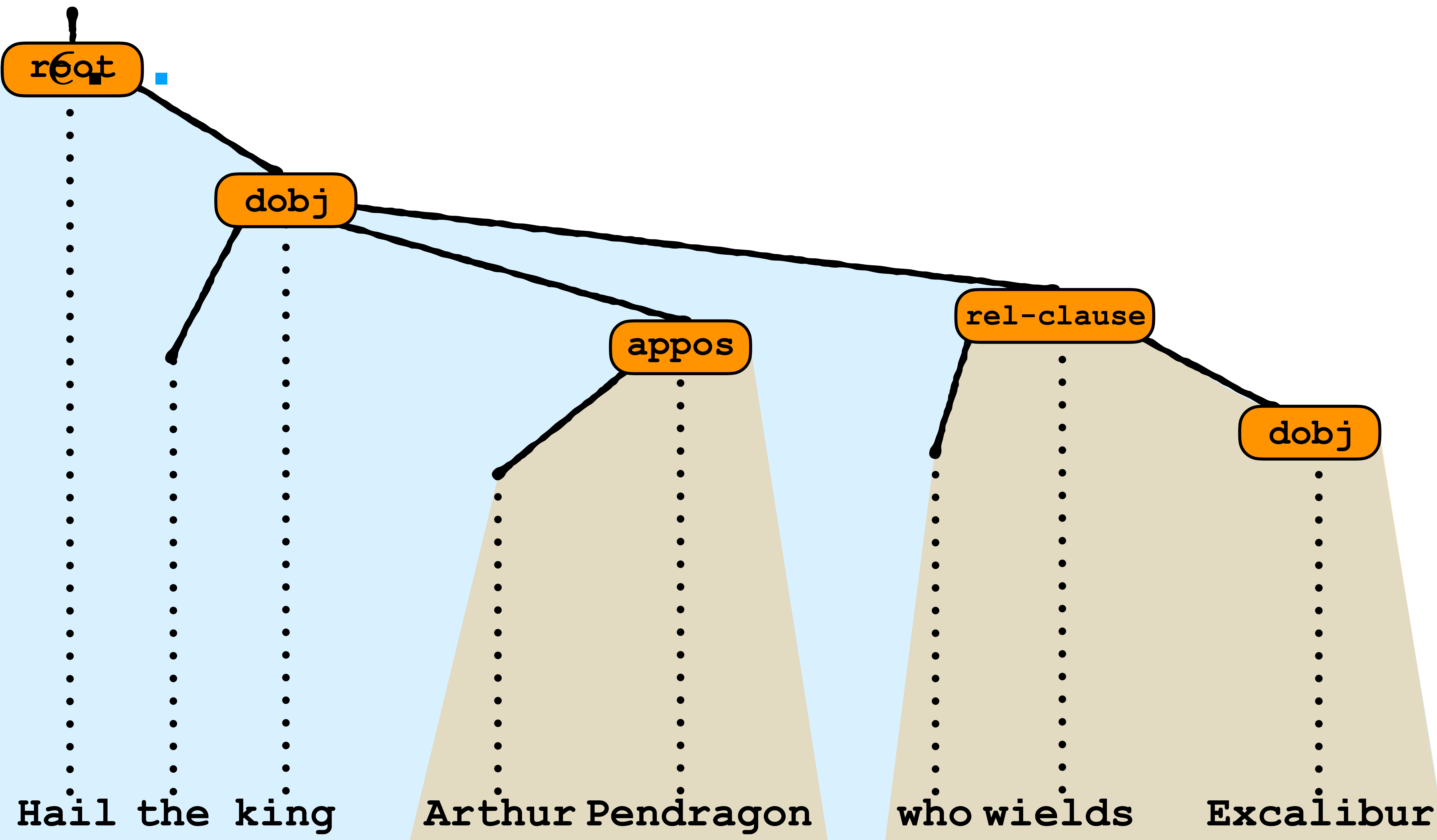


Let’s exploit the underlying punctuation under a generative model !

P(punctuated tree | unpunctuated tree)

= P(ϵ ▪ |root, unpunctuated tree)

Attach

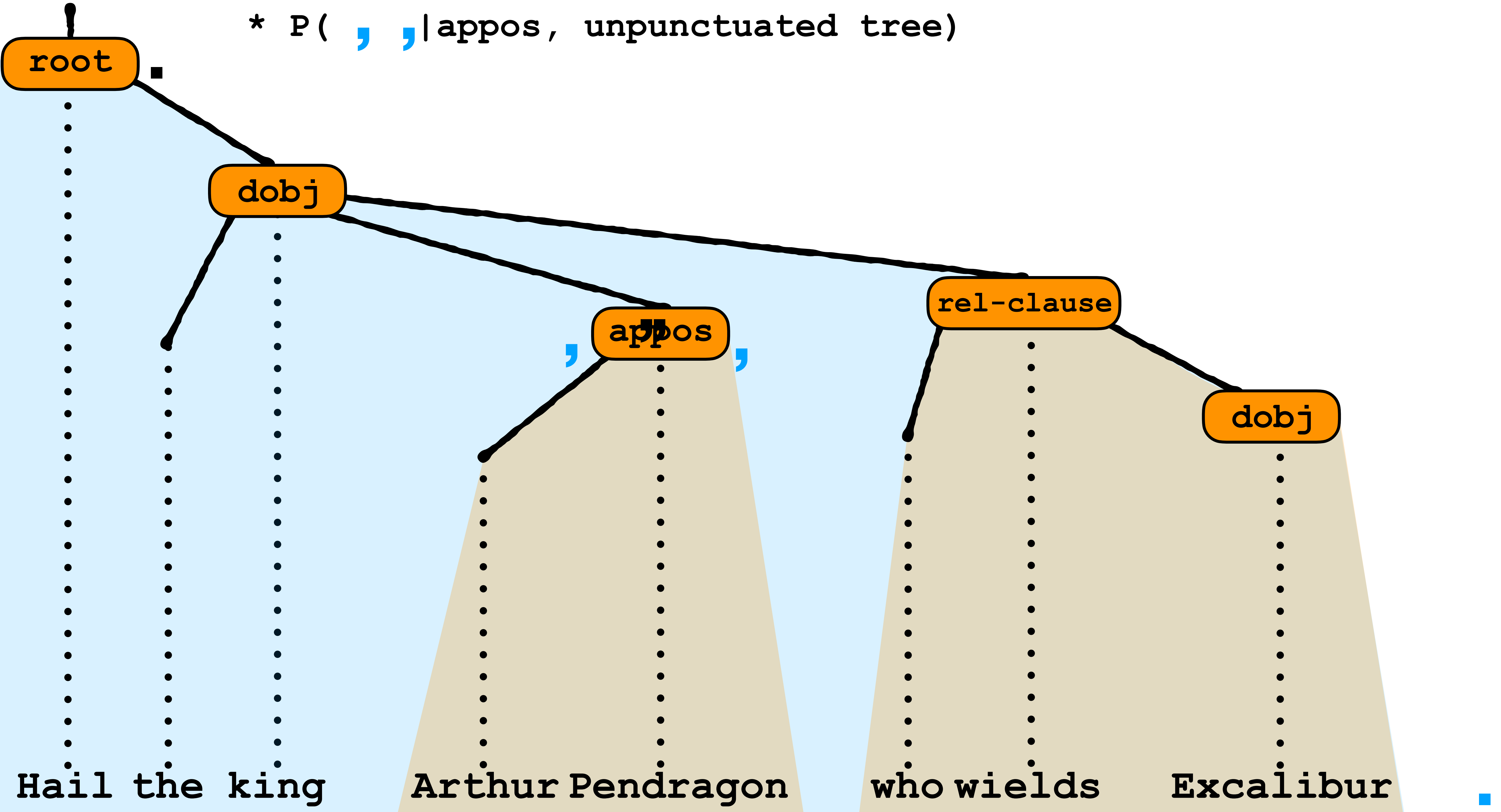


P(punctuated tree | unpunctuated tree)

= P(ϵ . |root, unpunctuated tree)

* P(, ,|appos, unpunctuated tree)

Attach



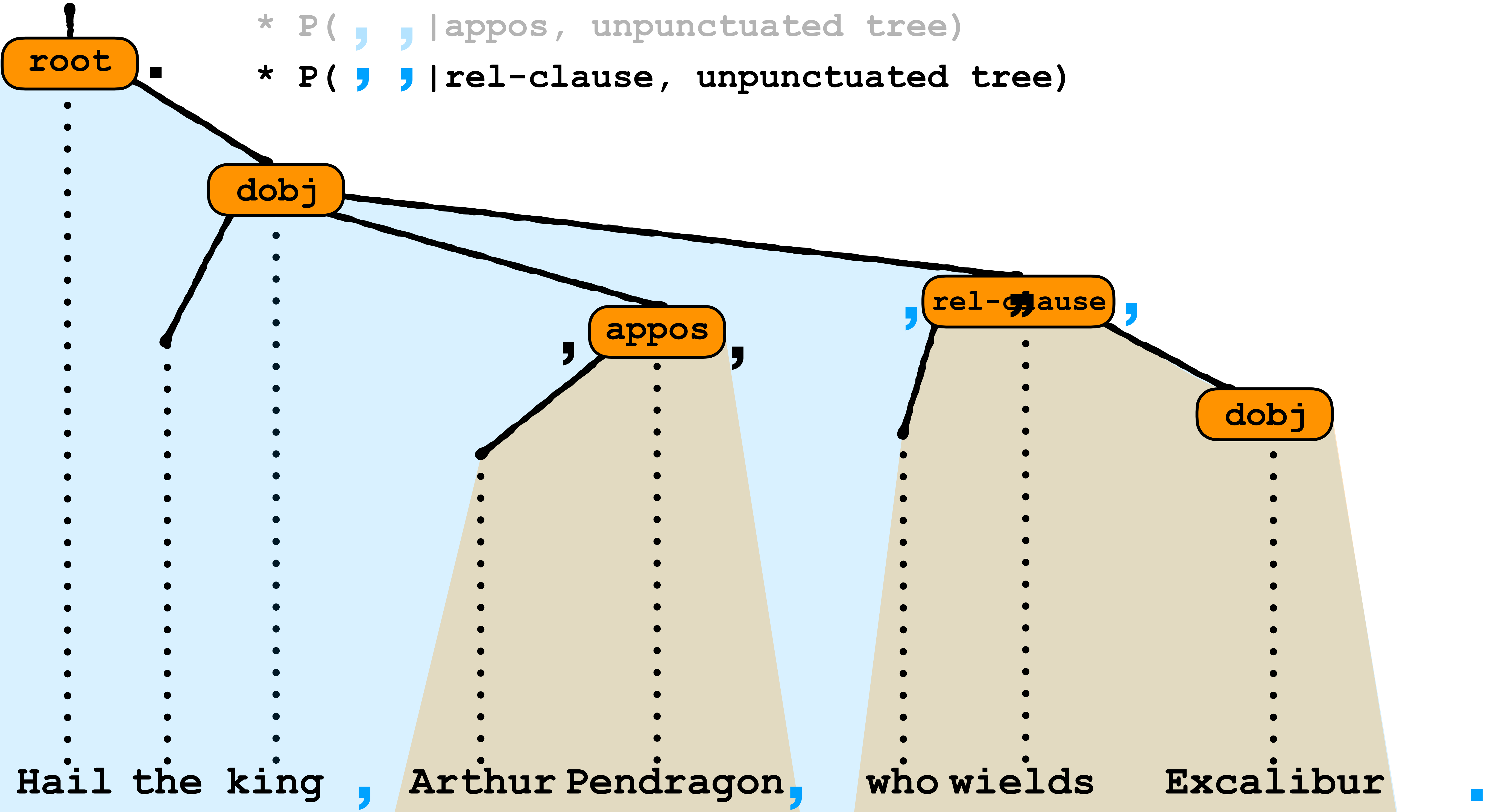
P(punctuated tree | unpunctuated tree)

= P(∈ . |root, unpunctuated tree)

   * P(, , |appos, unpunctuated tree)

   * P(, , |rel-clause, unpunctuated tree)

Attach



P(punctuated tree | unpunctuated tree)

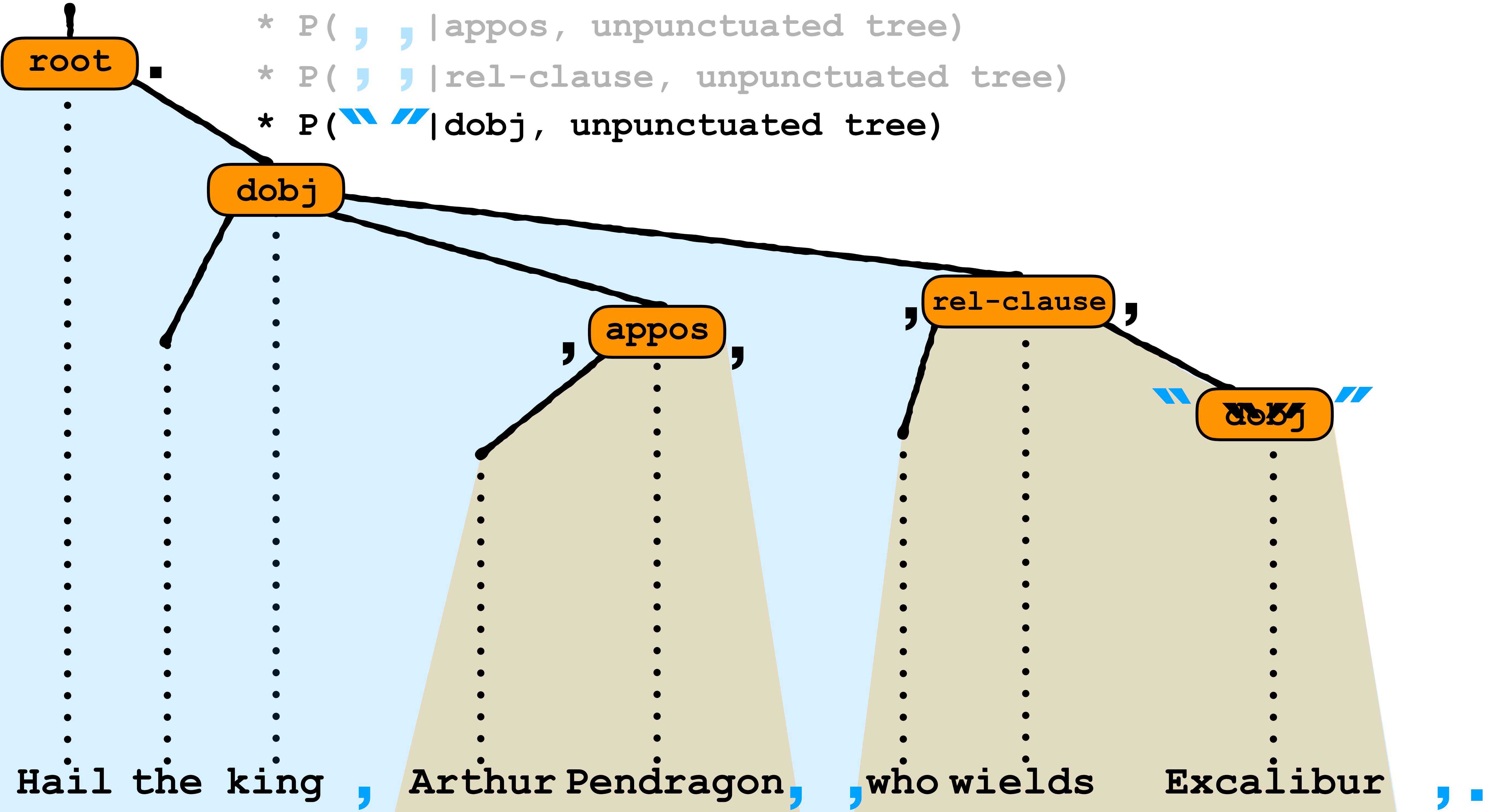
= P(ϵ . |root, unpunctuated tree)

* P(, , |appos, unpunctuated tree)

* P(, , |rel-clause, unpunctuated tree)

*** P(“ ”|dobj, unpunctuated tree)**

Attach



P(punctuated tree | unpunctuated tree)

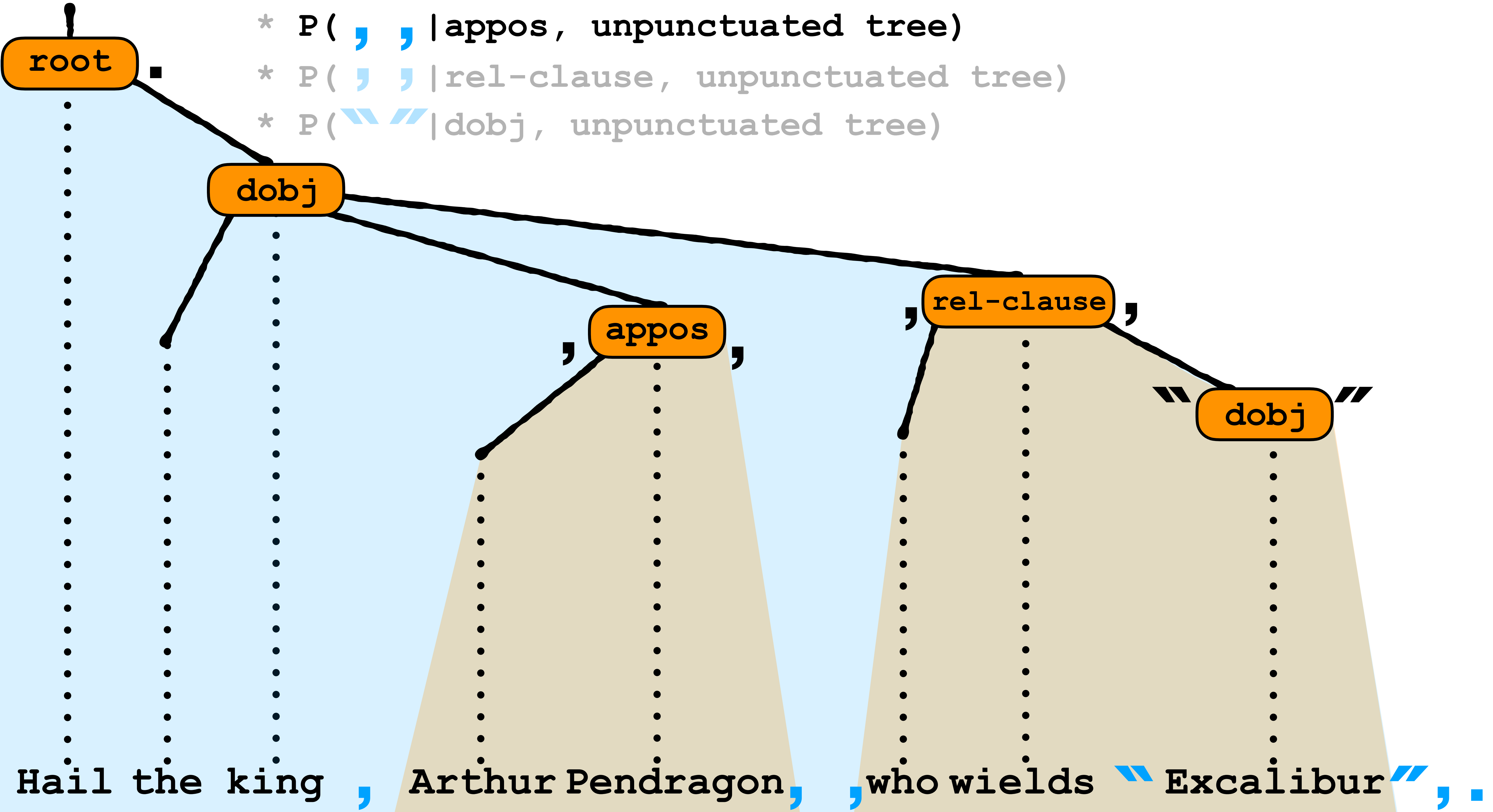
= P(∊ . |root, unpunctuated tree)

  * P(, , |appos, unpunctuated tree)

  * P(, , |rel-clause, unpunctuated tree)

  * P(“ ”|dobj, unpunctuated tree)

Attach



`P( , , | appos, unpunctuated tree)` = **0.4**

appos

left punct	right punct	Prob		f1	f2	f3	f4
,	,	0.4	$\xleftarrow{\theta}$	0	1	2	1
“	”	0.3	$\xleftarrow{\theta}$	0	1	0	0
ϵ	,	0.05	$\xleftarrow{\theta}$	0	0	0	1
,	ϵ	0.05	$\xleftarrow{\theta}$	0	0	0	1
ϵ	ϵ	0.2	$\xleftarrow{\theta}$	1	0	0	0

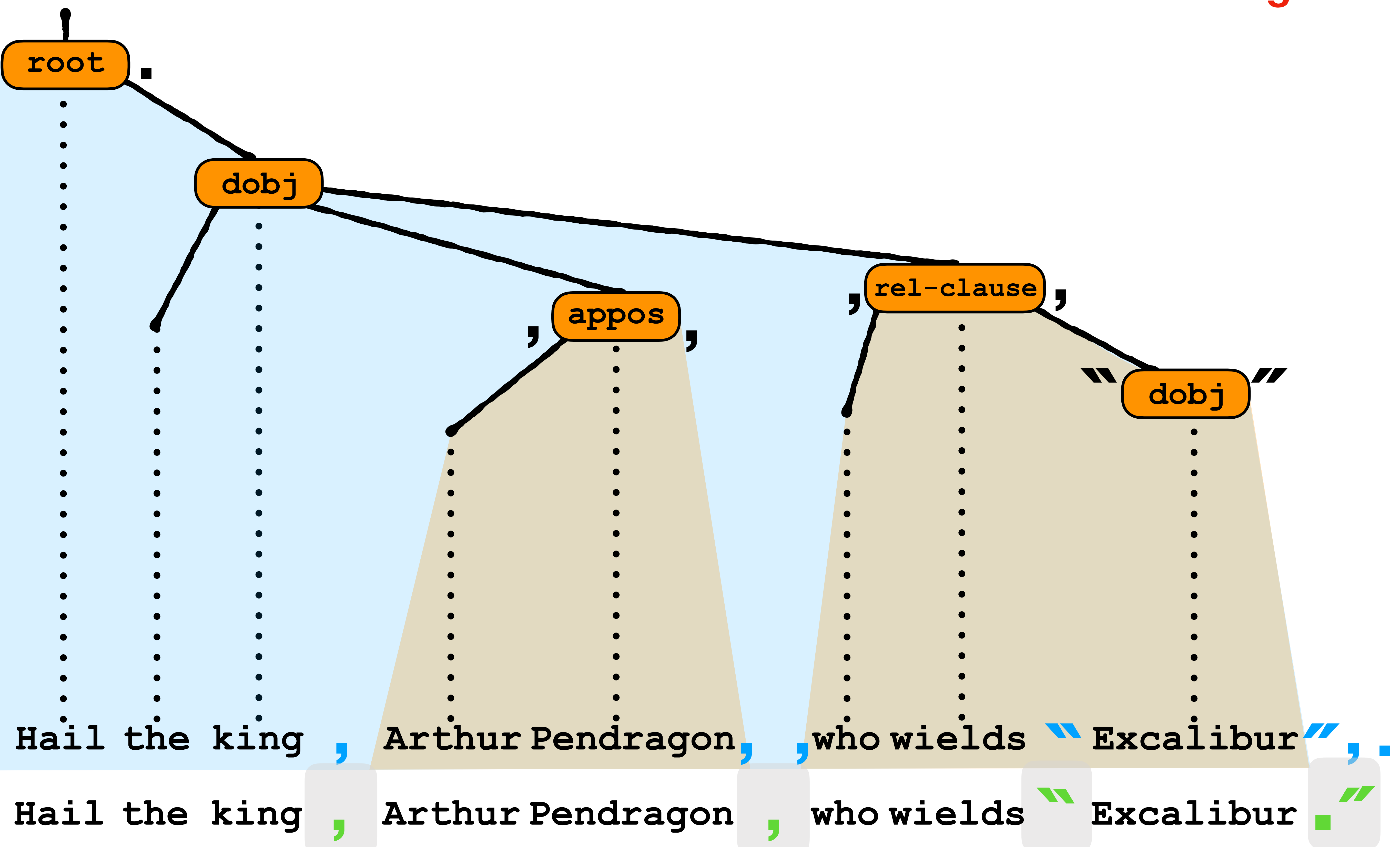
...

- **f1: indicator feature for no punctuation (ϵ ϵ).**
- **f2: indicator feature that checks for symmetry.**
- **f3: span length feature for attaching commas.**
- **f4: indicator feature that checks for internal comma.**

⋮

P(surface punct | punctuated tree)

Rewriting



Rewriting — A Sliding Window Model

$\boxed{,\,,}\;,\,)\,”.$

Left-absorb $P(\,,\;,\mapsto\,,\,)$

Left-absorb $P(\,,\;,\mapsto\,,\,)$

Left-absorb $P(\,,\;)\mapsto\,)\,)$

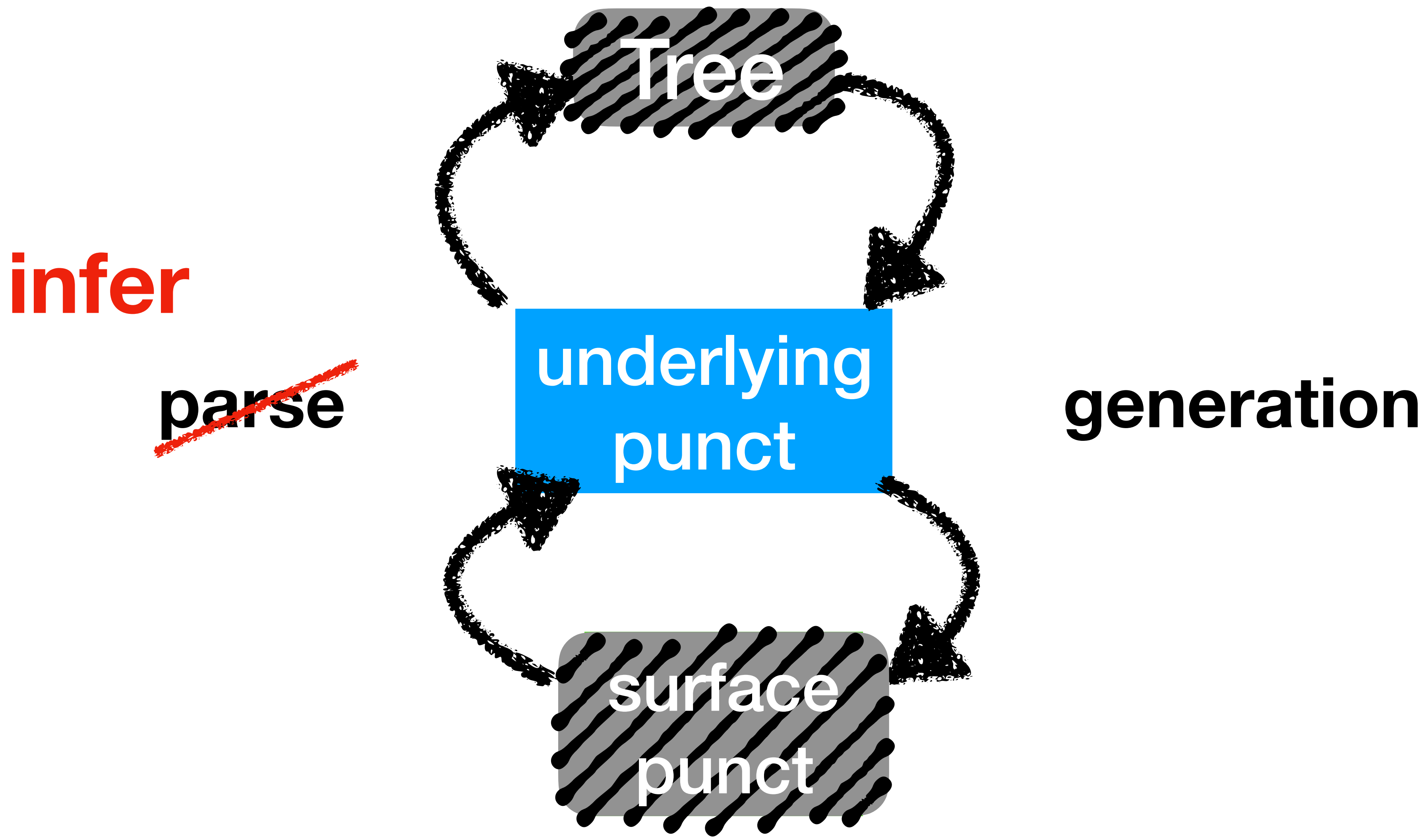
identity $P(\,)”\mapsto\,)”\,)$

swap $P(\,”\;.\mapsto\,.\;”\,)$

Geoffrey Nunberg

Summary generative story

- Underlying punctuation is generated at each tree node (not quite independently).
- Total underlying punctuation at each slot between words is rewritten into surface punctuation (independently).

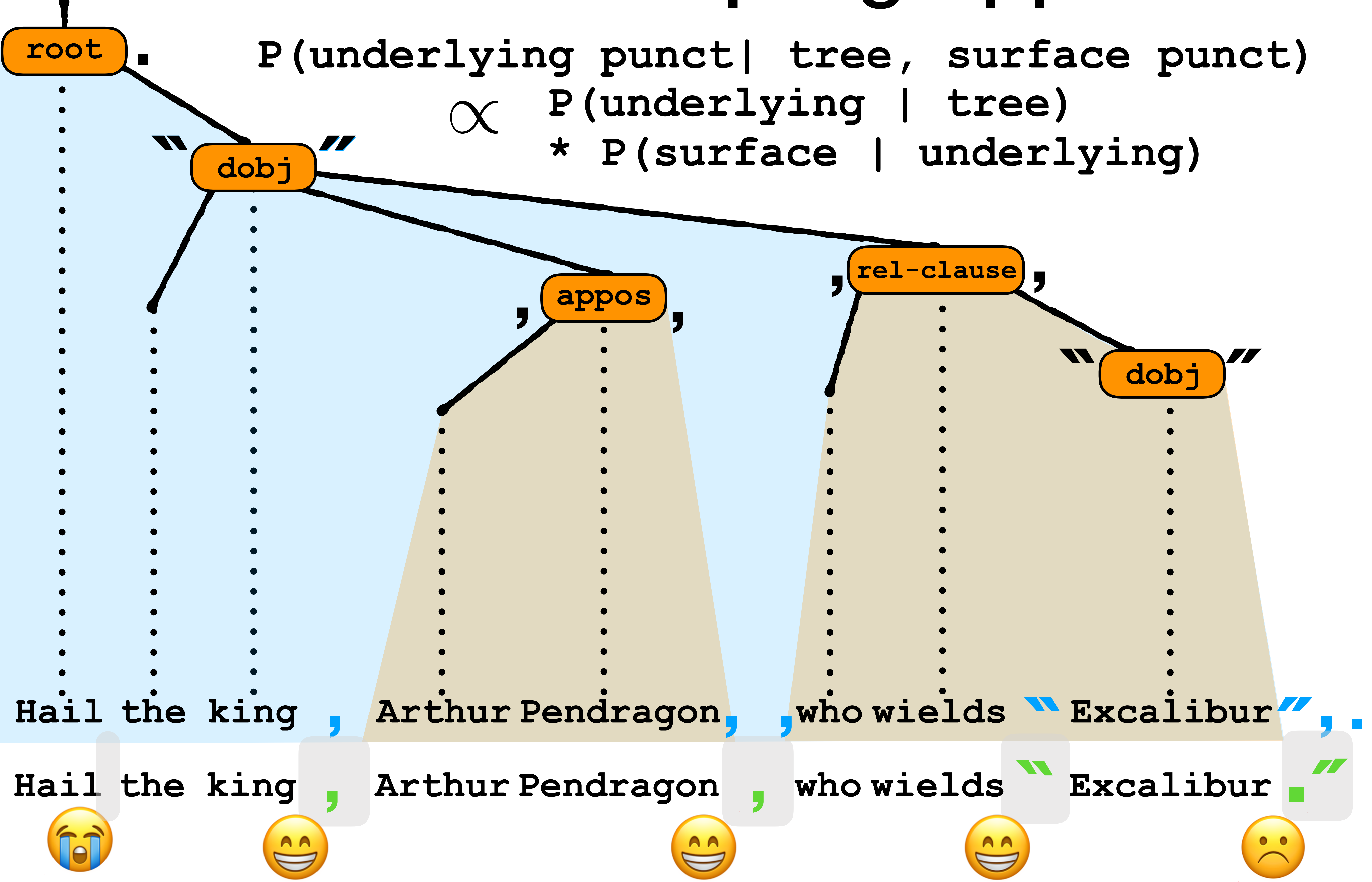


In training, we observe the tree and surface punct.
Want to recover the underlying punctuation.

`P(underlying punct | tree, surface punct)`

Method 1. A Sampling Approach

$$P(\text{underlying punct} \mid \text{tree, surface punct}) \propto P(\text{underlying} \mid \text{tree}) * P(\text{surface} \mid \text{underlying})$$

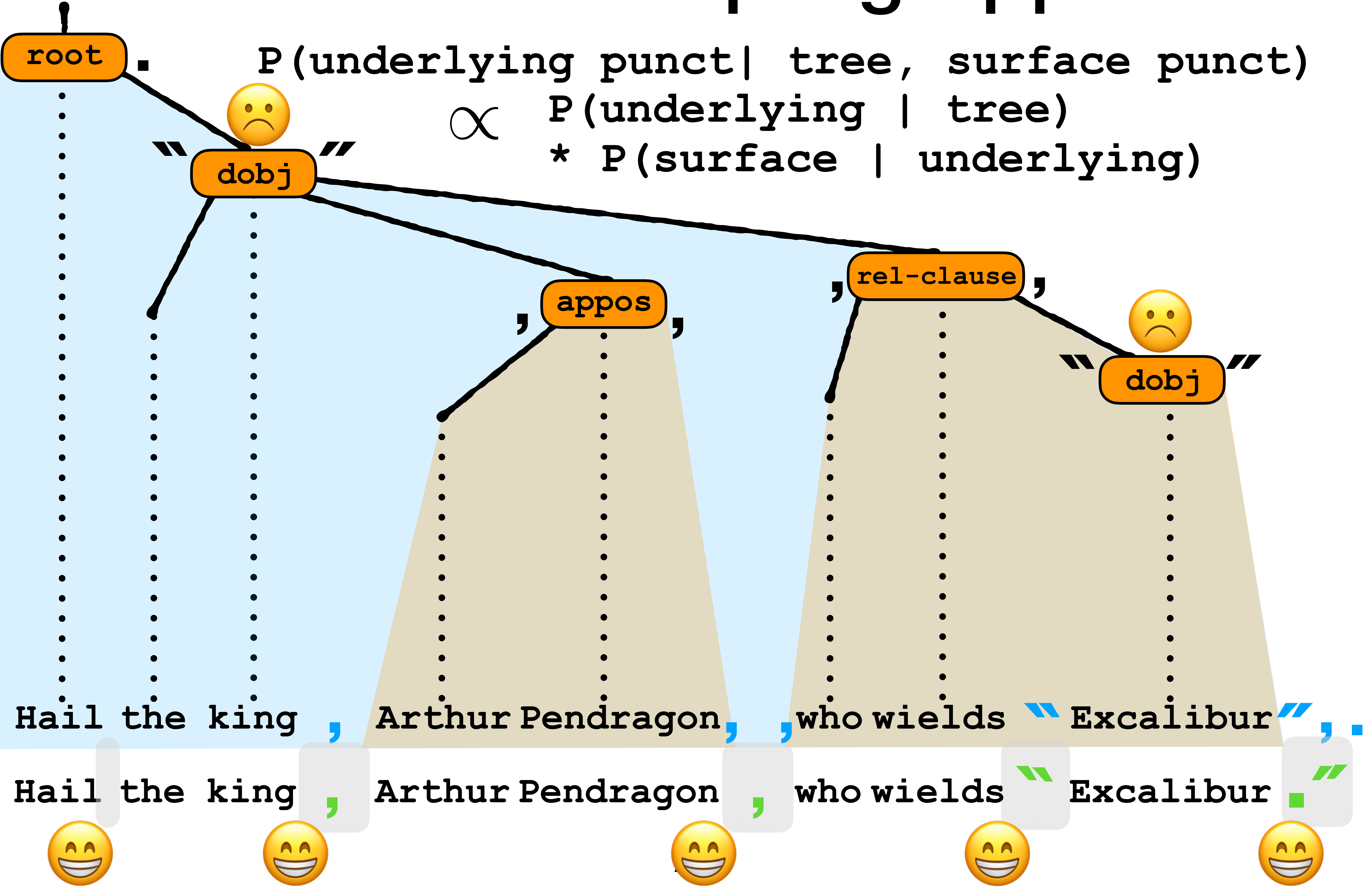


Method 1. A Sampling Approach

P(underlying punct| tree, surface punct)

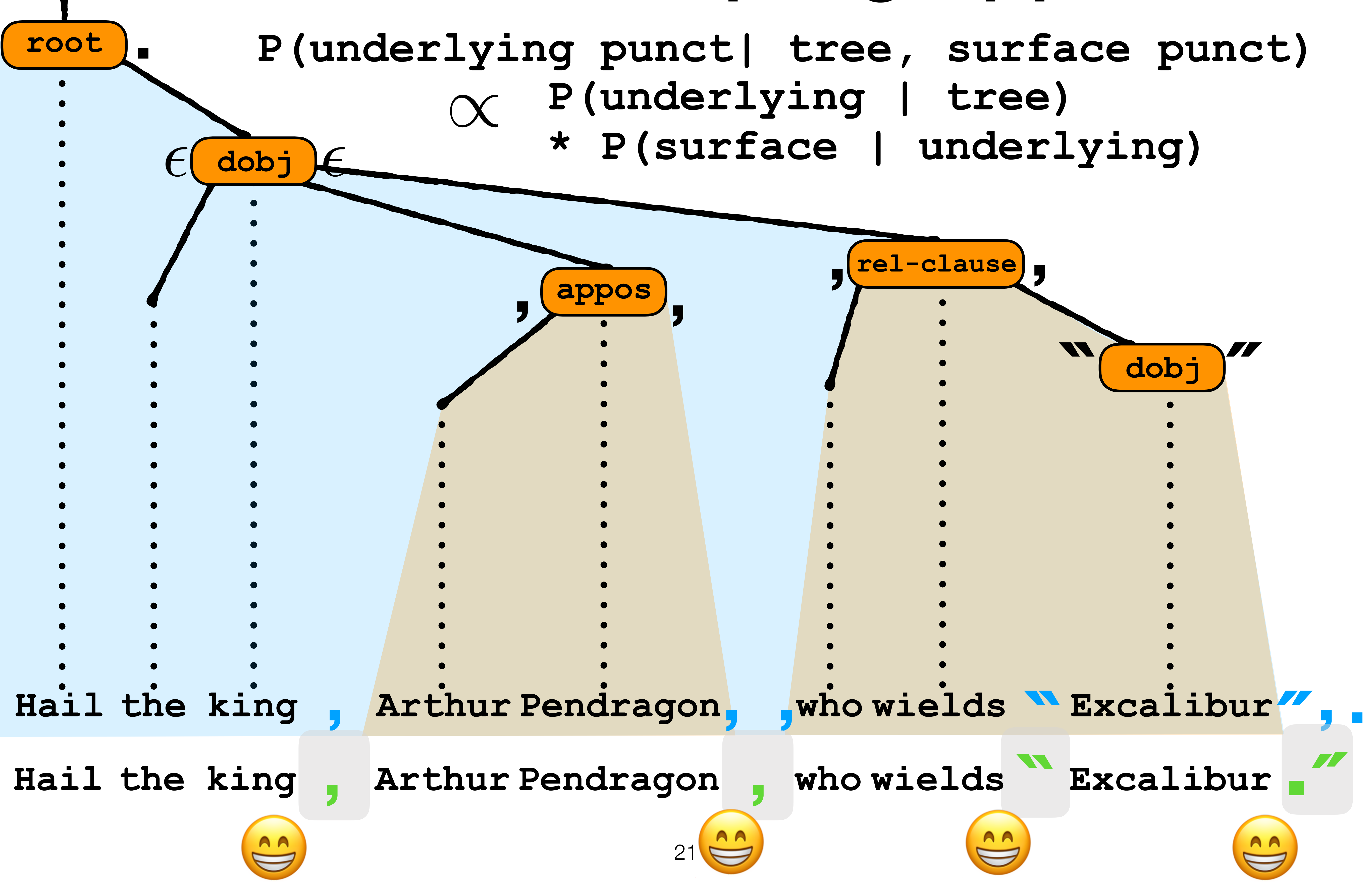
$\propto$ P(underlying | tree)

* P(surface | underlying)



Method 1. A Sampling Approach

```
P(underlying punct| tree, surface punct)
    ∝  P(underlying | tree)
       * P(surface | underlying)
```



Method 2. O(n) Dynamic Programming

Idea : Make a (weighted) CFG that generates just the possible underlyingly punctuated trees for *this* sentence

Soft constraint 1 :Underlying punctuation fits the tree structure

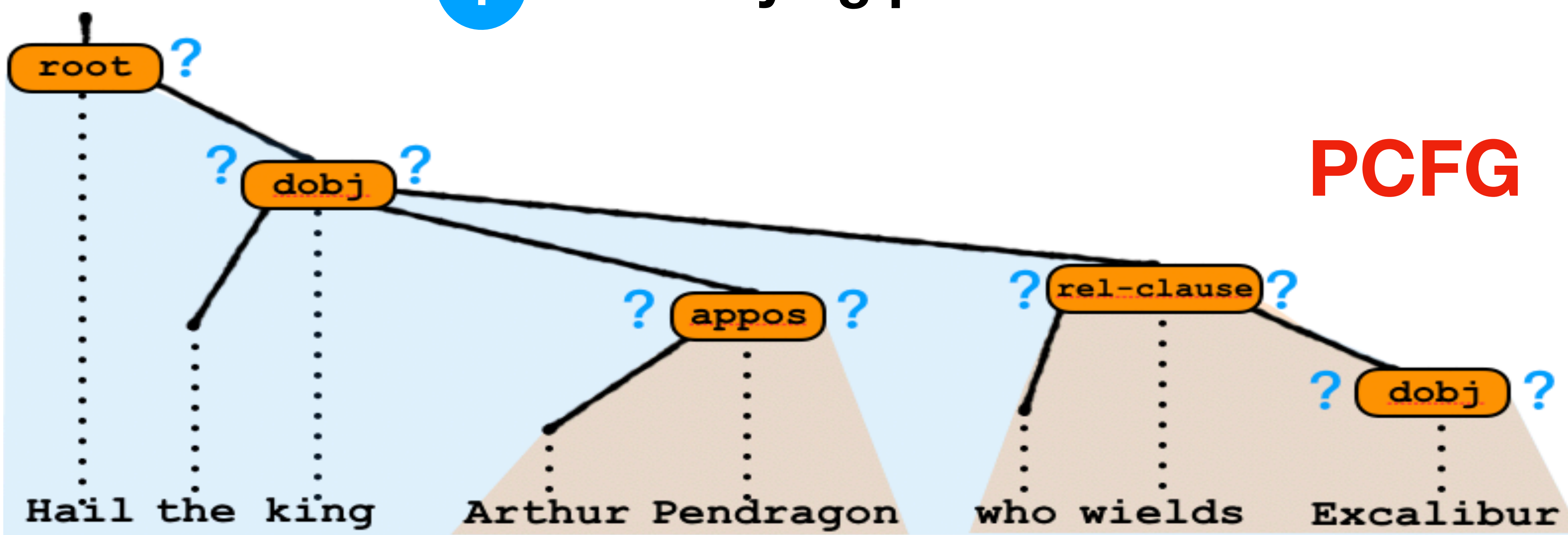
Soft constraint 2 :Underlying punctuation fits surface punctuation

Intersect 1 2 ⟶ our CFG

Method 2. O(n) Dynamic Programming

Idea : Make a (weighted) CFG that generates just the possible underlyingly punctuated trees for *this* sentence

Soft constraint 1 :Underlying punctuation fits the tree structure



PCFG

Soft constraint 2 :Underlying punctuation fits surface punctuation

Intersect 1 2 → our CFG

Method 2. O(n) Dynamic Programming

Idea : Make a (weighted) CFG that generates just the possible underlyingly punctuated trees for *this* sentence

Soft constraint 1 **:Underlying punctuation fits the tree structure**

PCFG

Soft constraint 2 **:Underlying punctuation fits surface punctuation**

weighted FSA

? ? ? ?

`Hail the king , Arthur Pendragon , who wields “ Excalibur .”`

Intersect 1 2 ⟶ **our CFG**

Method 2. O(n) Dynamic Programming

Idea : Make a (weighted) CFG that generates just the possible underlyingly punctuated trees for *this* sentence

Soft constraint 1 **:Underlying punctuation fits the tree structure**

PCFG

Soft constraint 2 **:Underlying punctuation fits surface punctuation**

weighted FSA

Intersect 1 2 $\longrightarrow$ **our CFG**

PCFG $\cap$ **weighted FSA** = **weighted CFG**

Method 2. O(n) Dynamic Programming

Idea : Make a (weighted) CFG that generates just the possible underlyingly punctuated trees for *this* sentence

Soft constraint 1 :Underlying punctuation fits the tree structure

Soft constraint 2 :Underlying punctuation fits surface punctuation

Intersect 1 2 → our CFG

Now we can find the best underlyingly punctuated tree

- **or sum over all of them for the model likelihood in training**

For our CFG, this can be done in O(n) time

Results

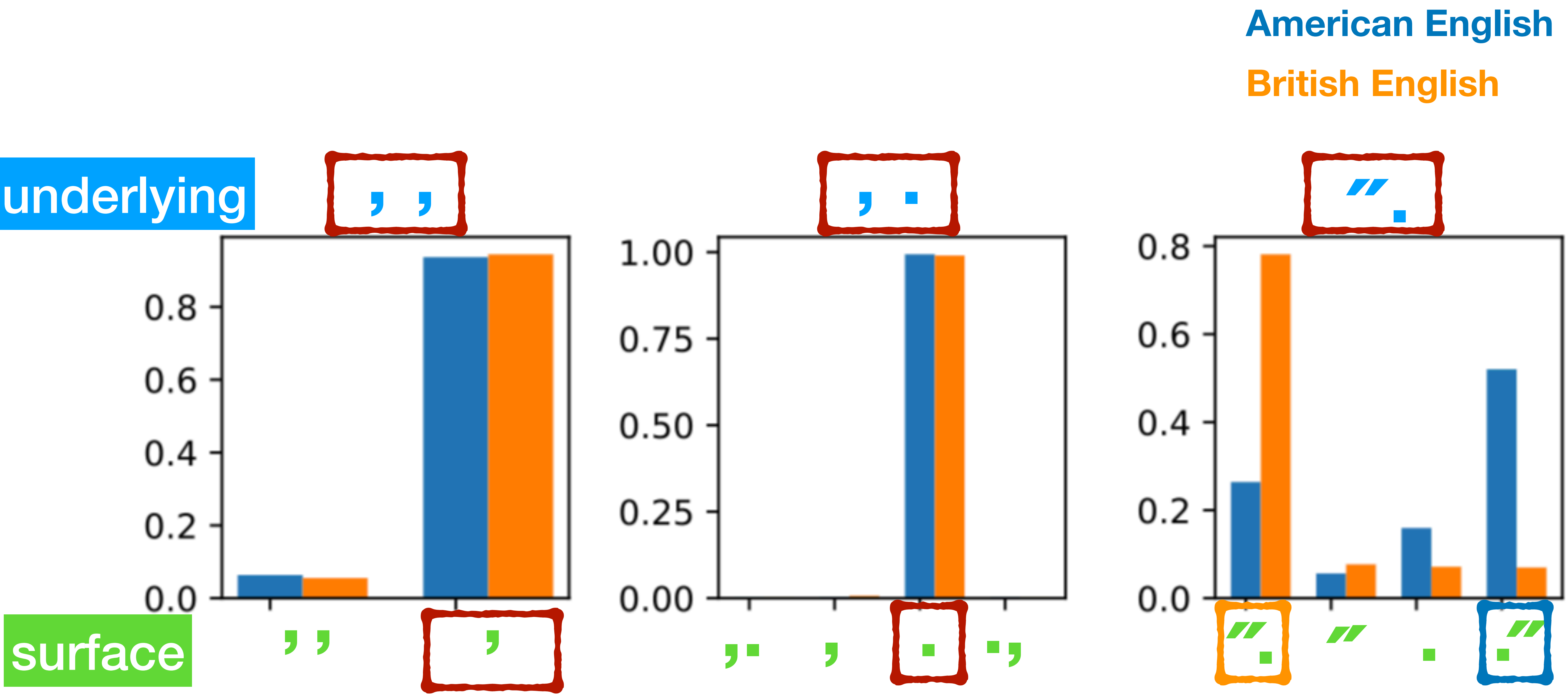
Rules Learned from the Noisy Channel

Analysis of Attachment model

Punctuation Restoration

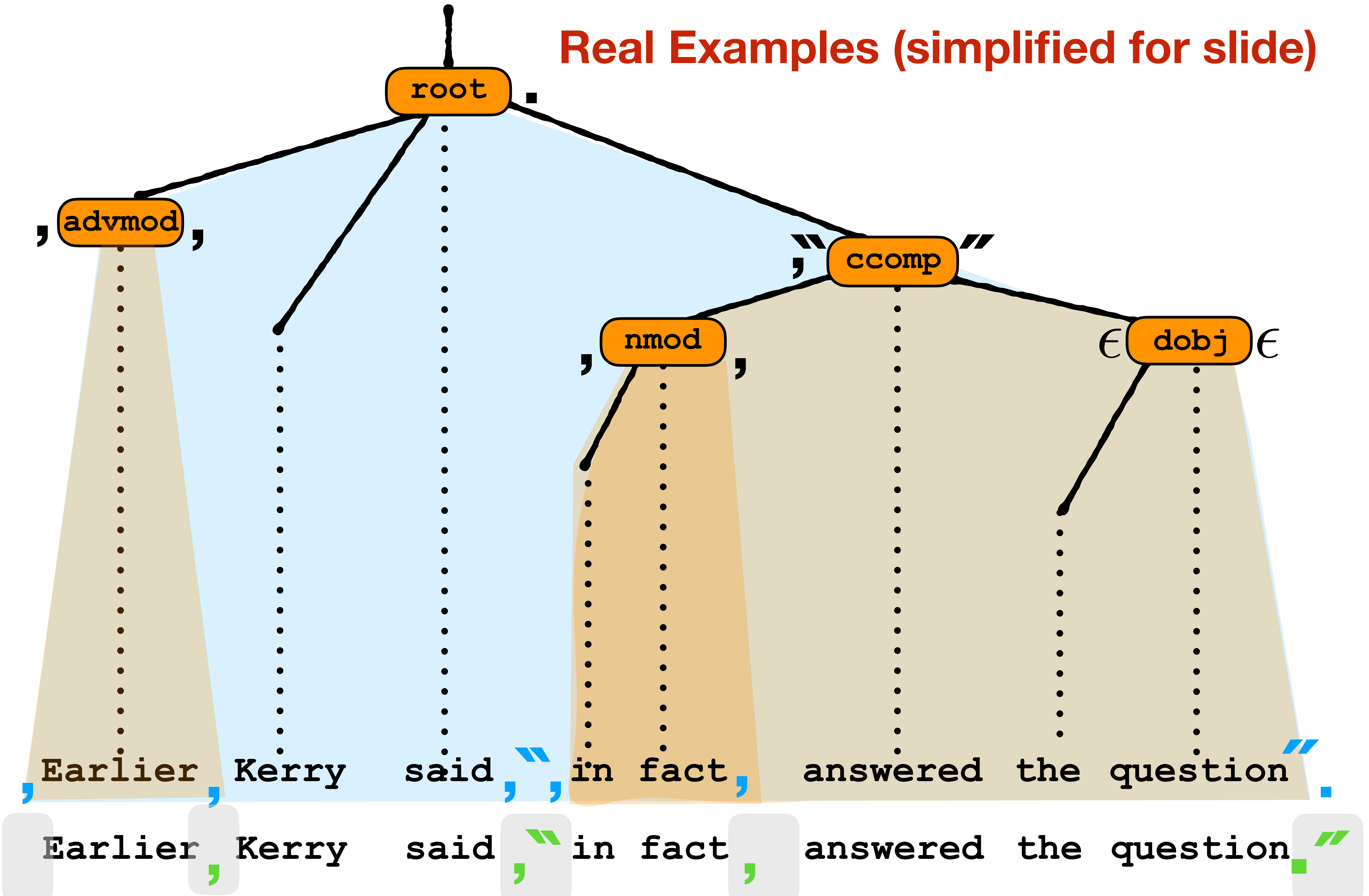
Syntactic Rephrasing

We learn high probability for Nunberg's English Rules

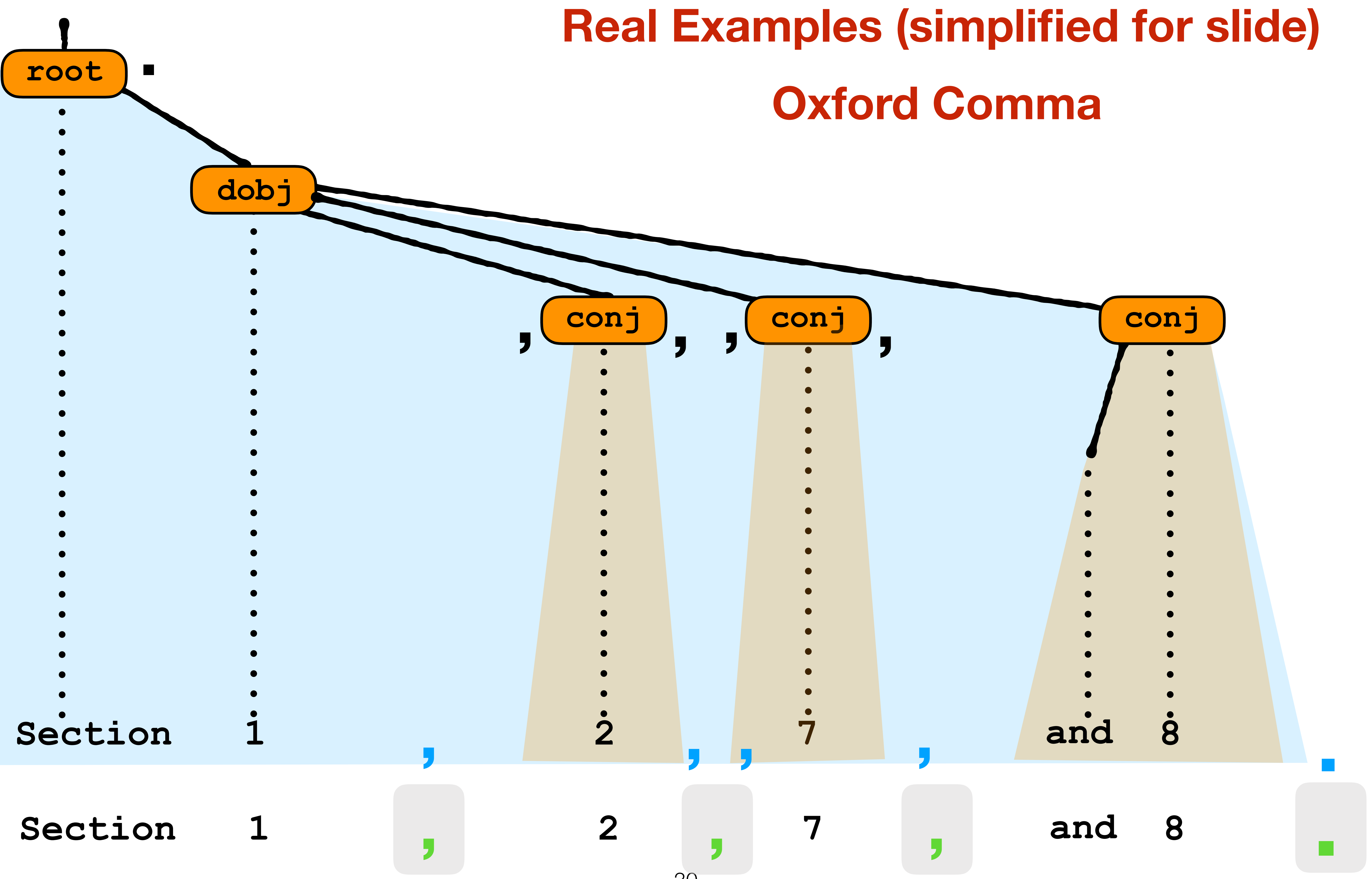


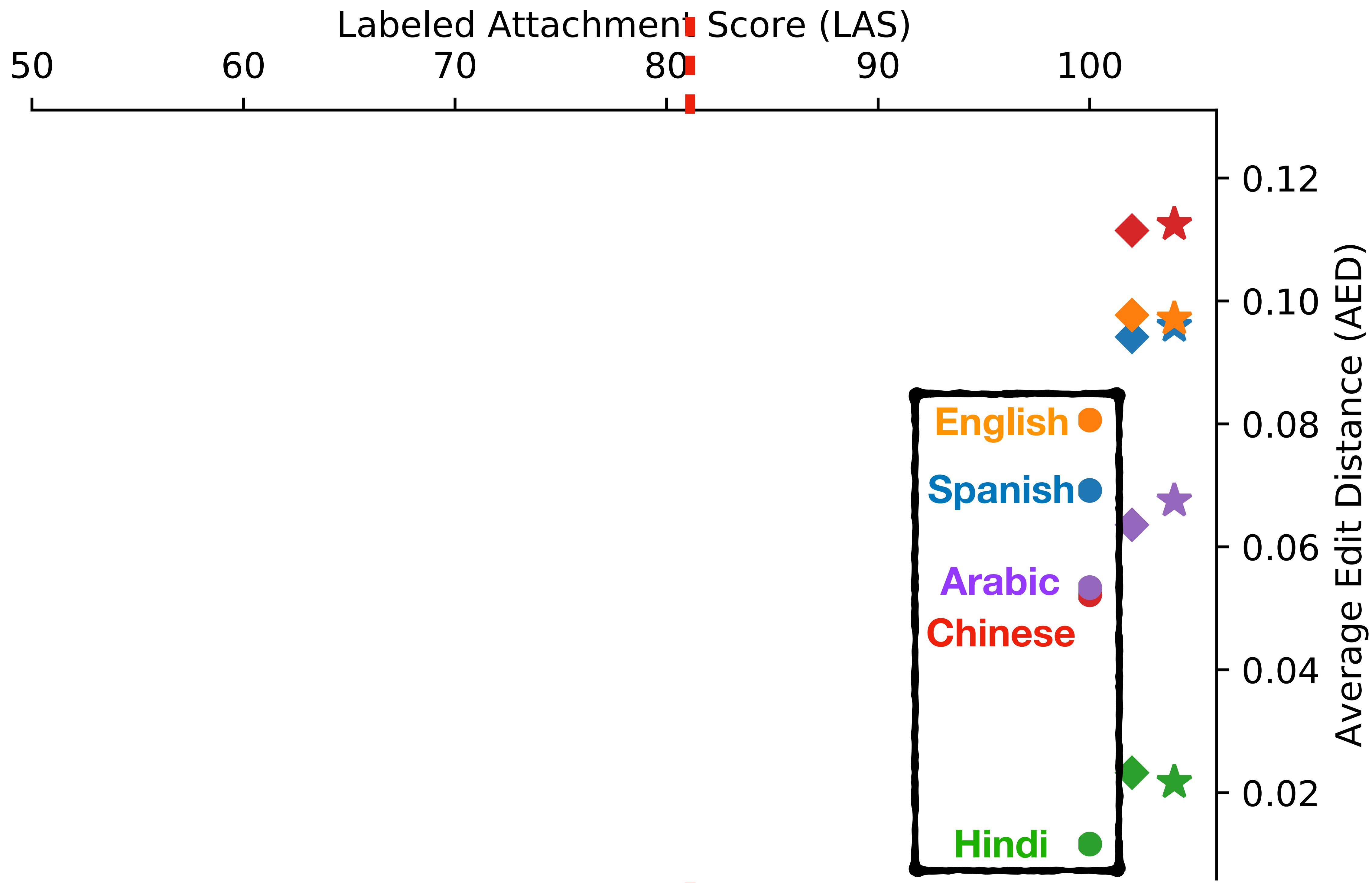
Distribution over rewrite rules

Viterbi recovers good underlying punctuation



Viterbi recovers good underlying punctuation

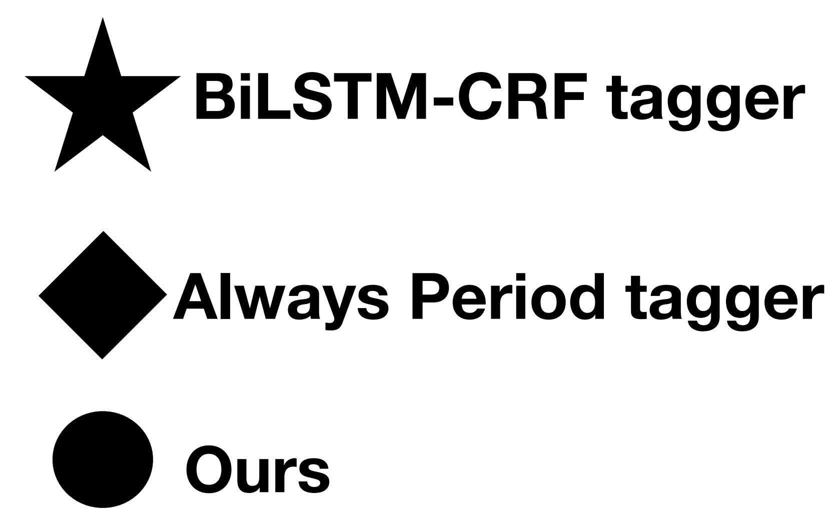


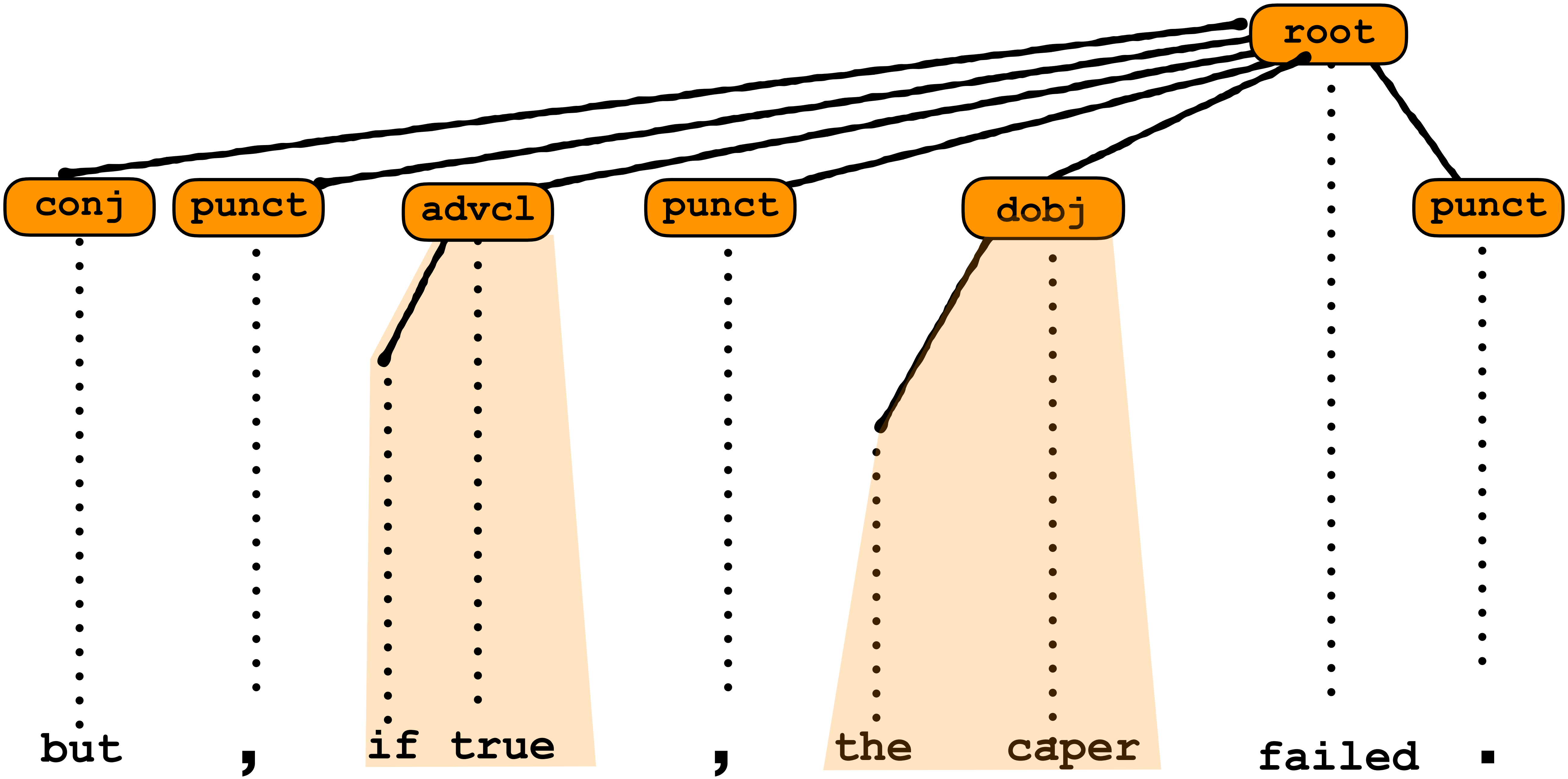


What if we don't have gold parses?

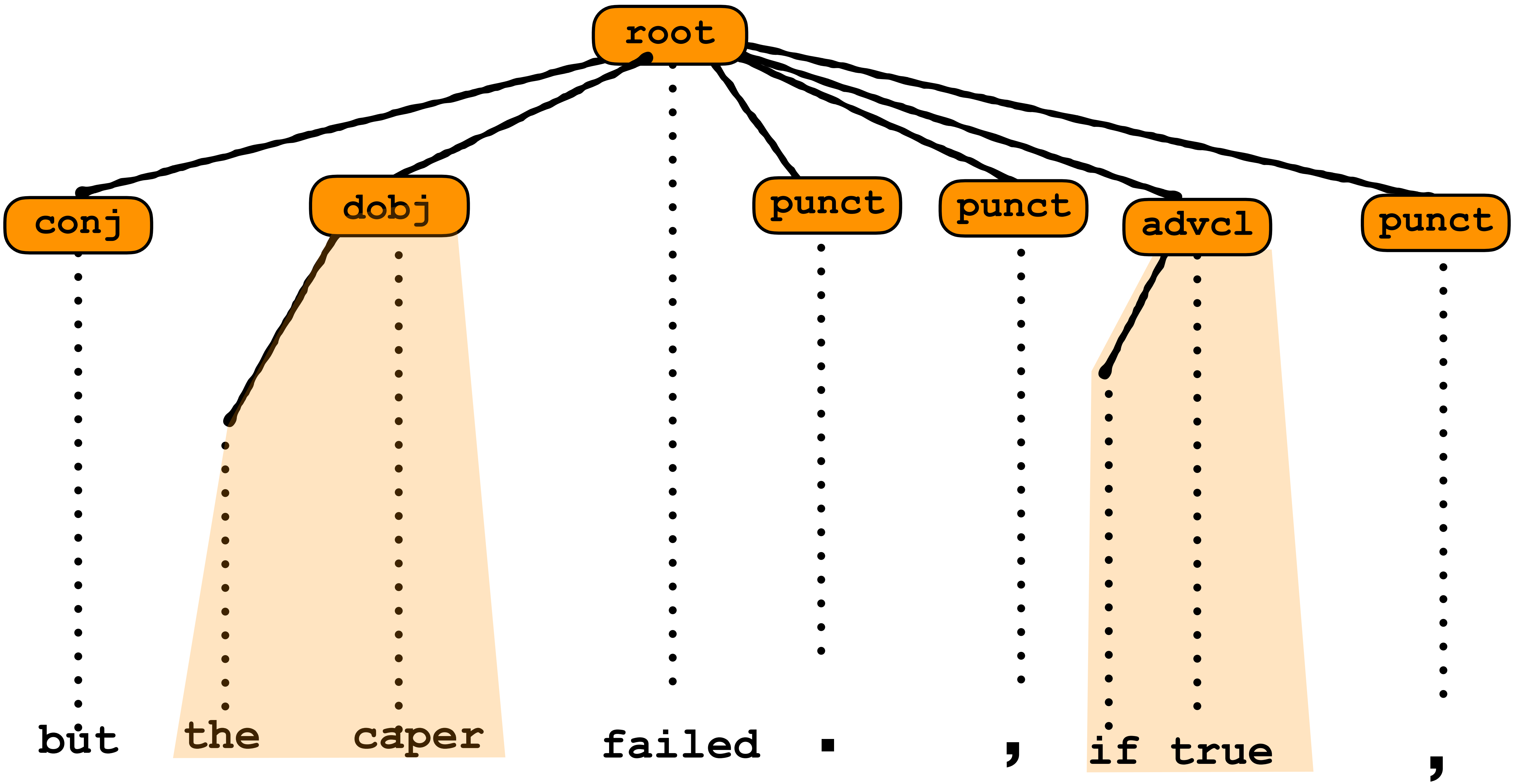
Punctuation Restoration

average edit distance [the lower the better]



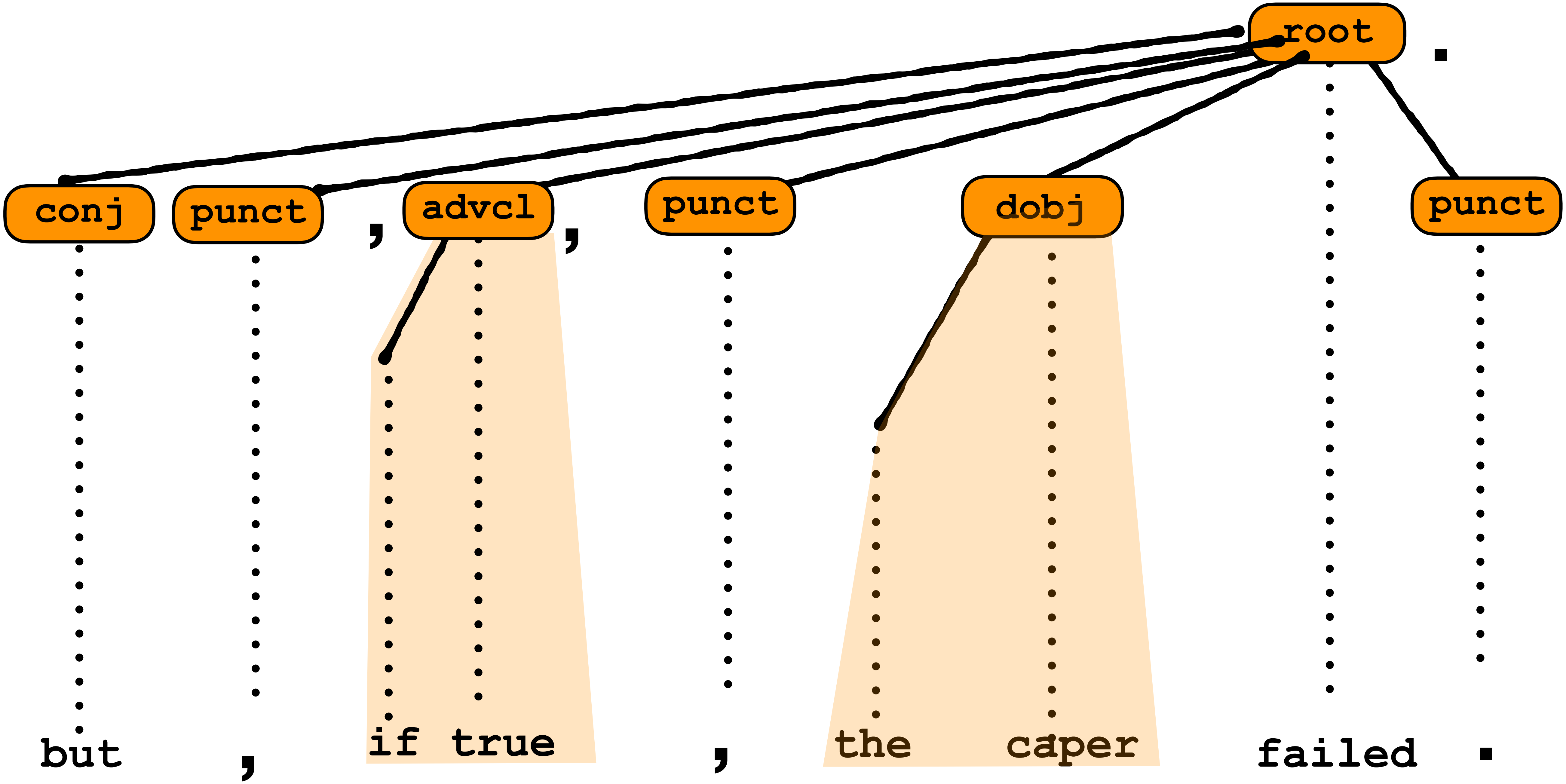


Syntactically Transform Sentence from Treebank

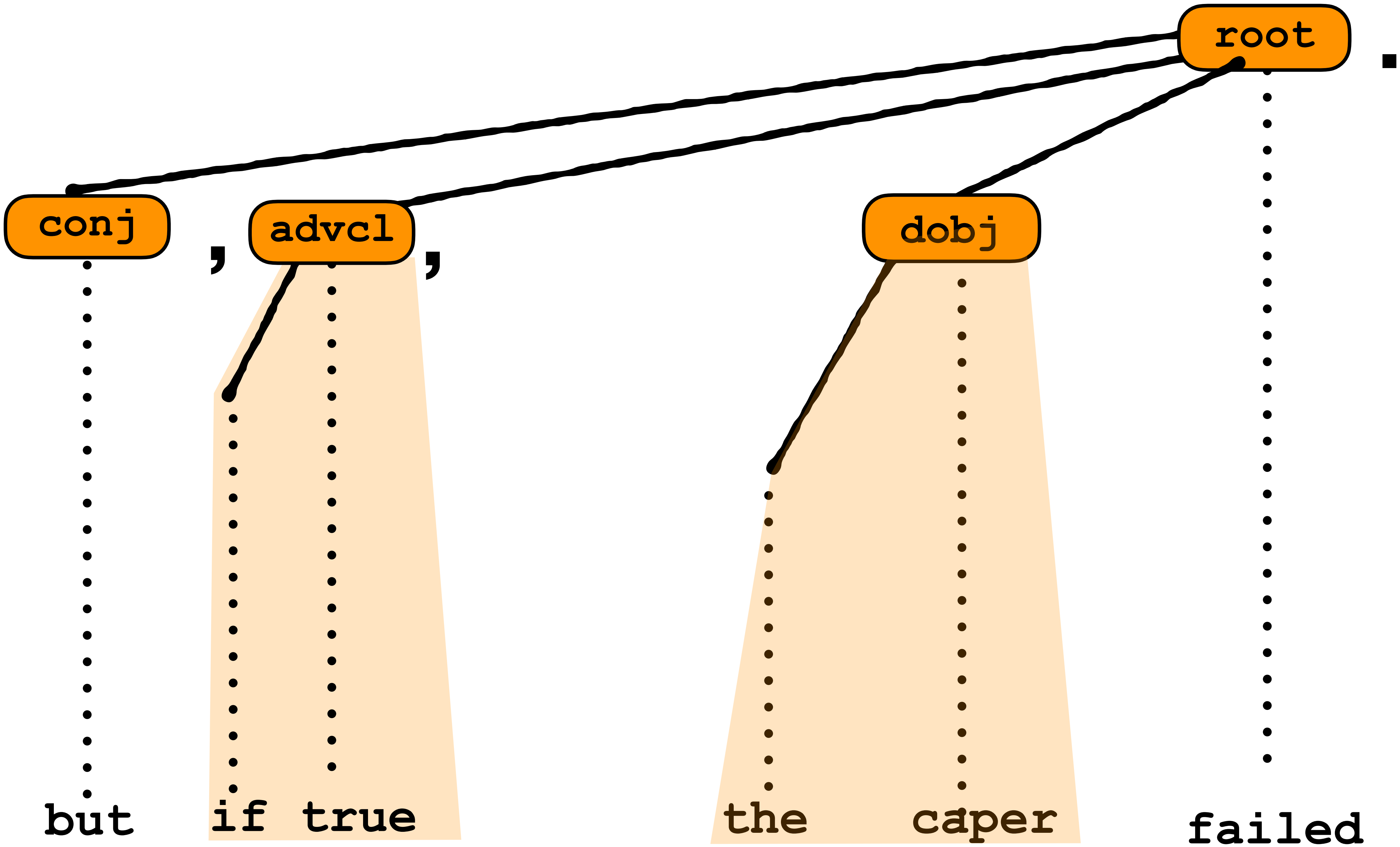


obviously ugly, and high perplexity under an LM

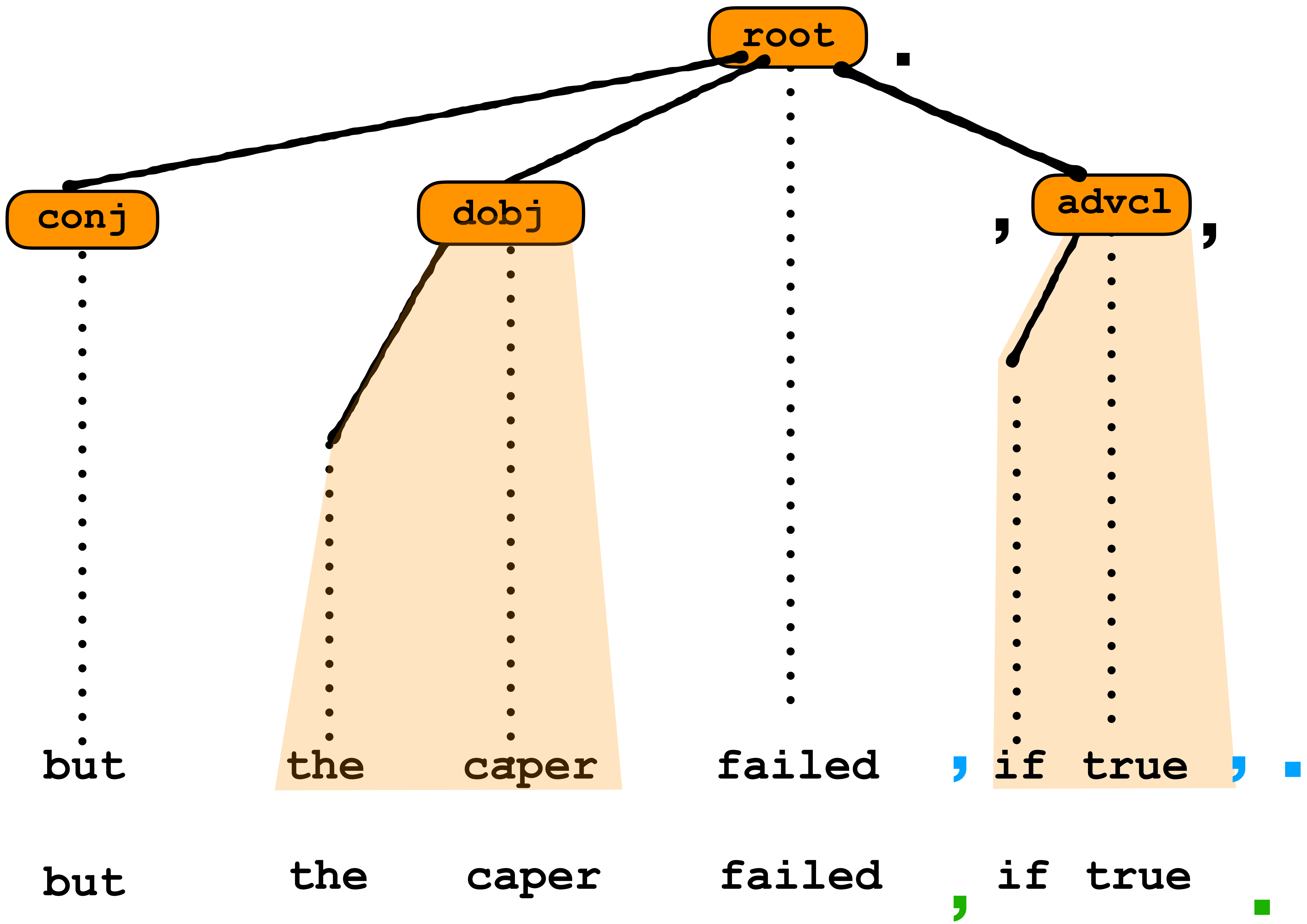
Syntactically Transform Sentence from Treebank



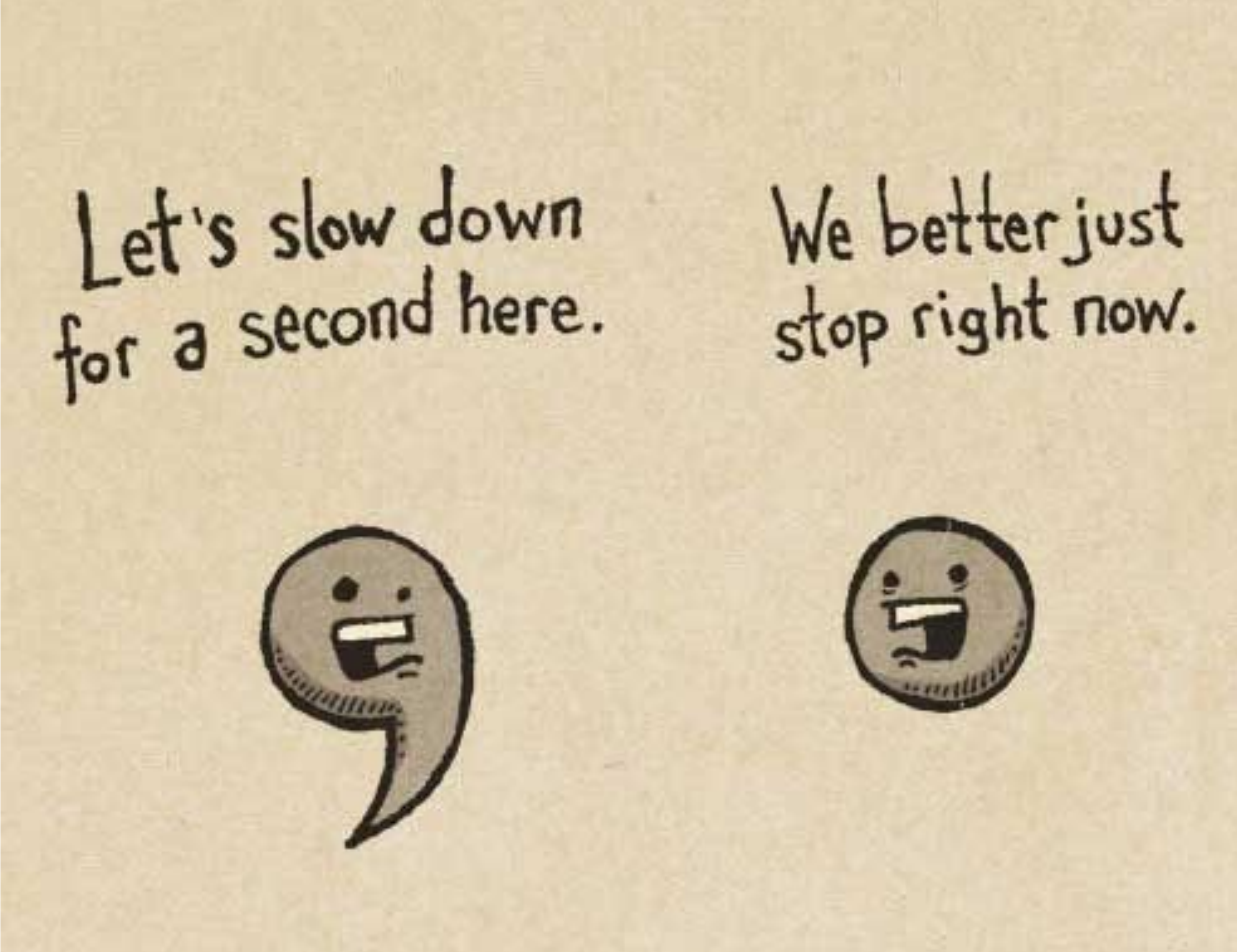
Syntactically Transform Sentence with our Annotation



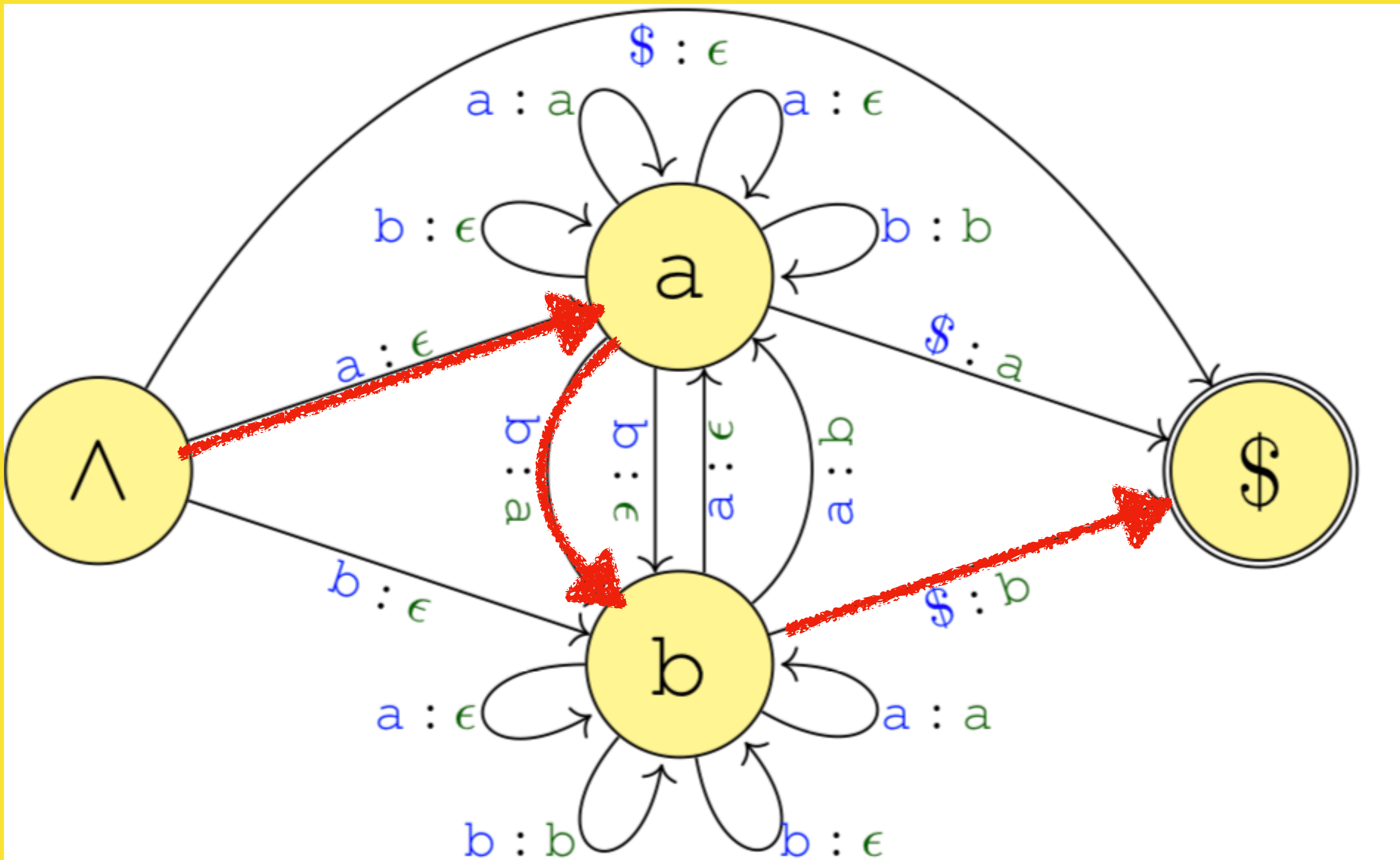
Syntactically Transform Sentence with our Annotation



Syntactically Transform Sentence with our Annotation


Let's slow down for a second here.
We better just stop right now.

Implemented by Finite State Transducer



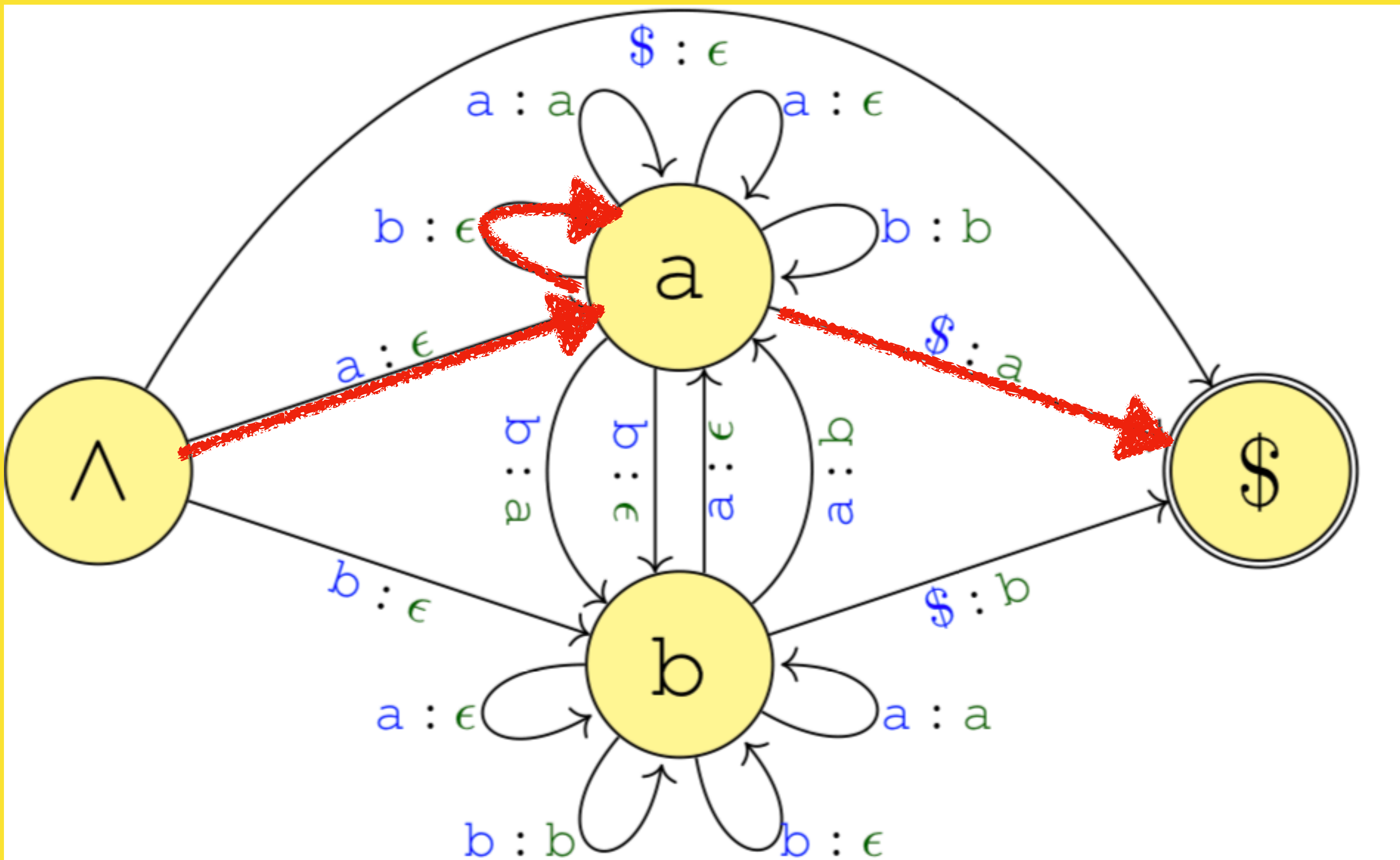
$ab \rightarrow ab$

$ab \rightarrow a$

$ab \rightarrow b$

$ab \rightarrow ba$

Implemented by Finite State Transducer

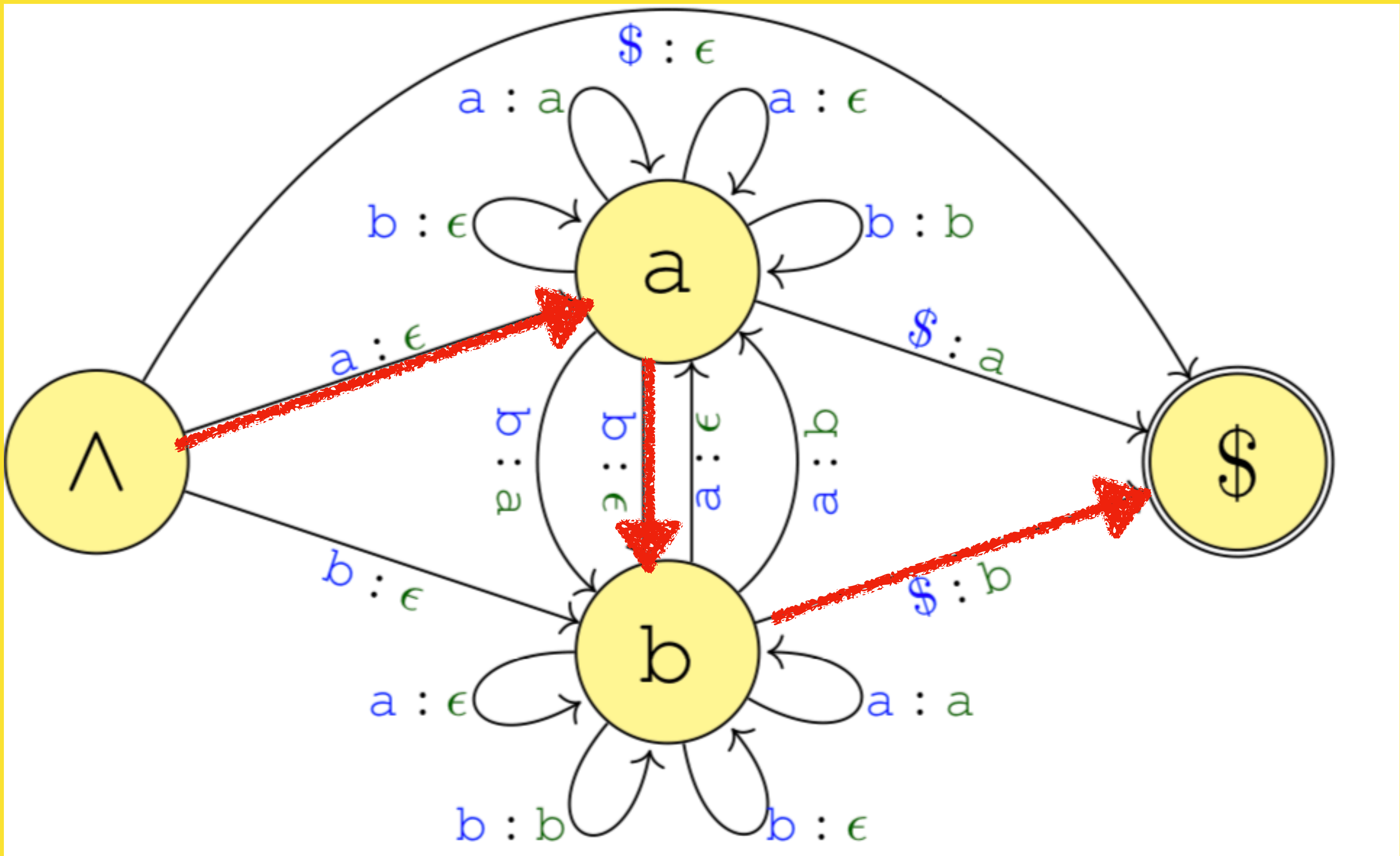


$ab \to ab$

$ab \to a$

$ab \to b$

$ab \to ba$



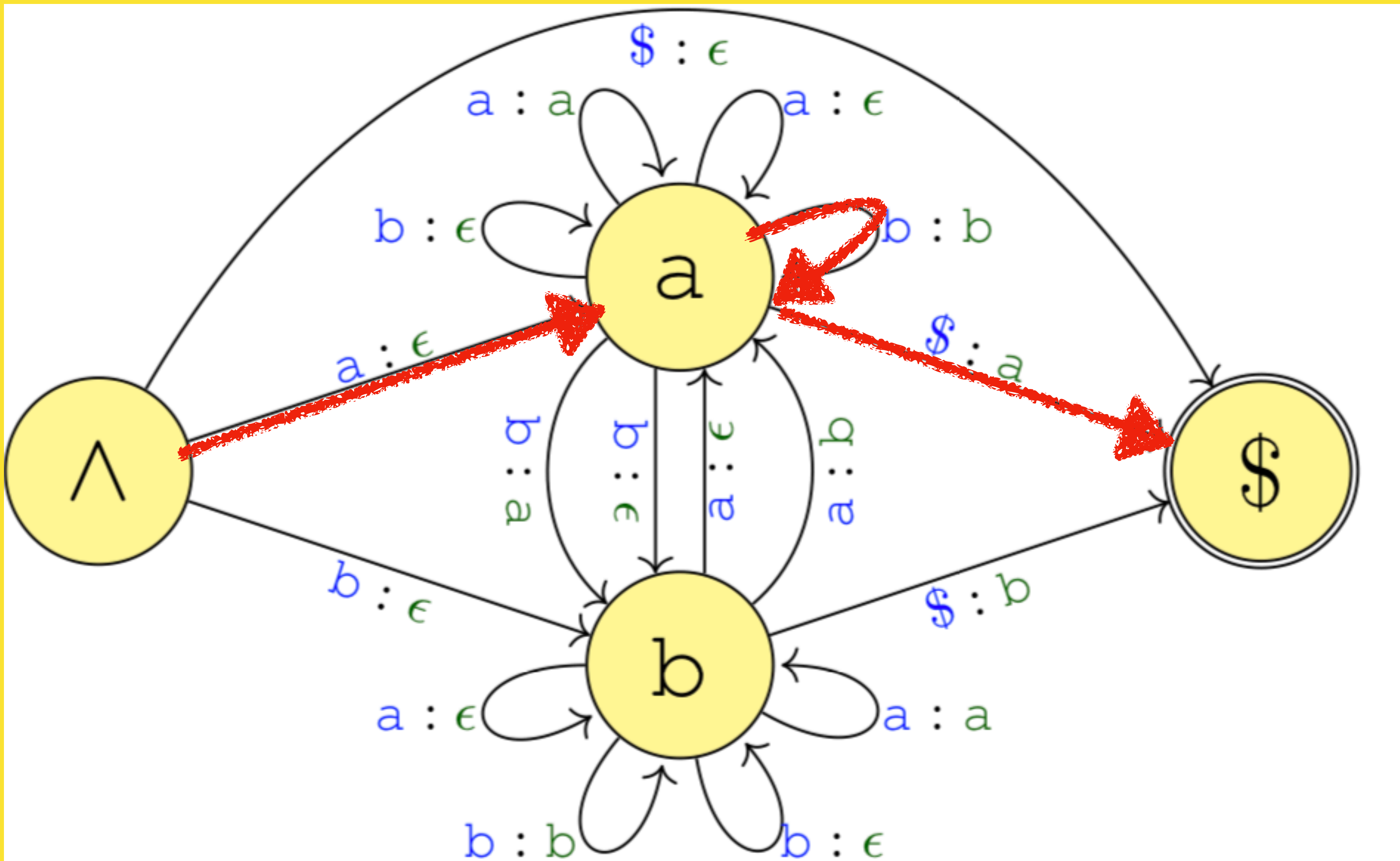
$ab \to ab$

$ab \to a$

$ab \to b$

$ab \to ba$

Implemented by Finite State Transducer



$ab \to ab$

$ab \to a$

$ab \to b$

$ab \to ba$