Context-Free Parsing

Adding Features & Improving Efficiency

Introduction to Natural Language Processing

Computer Science 585—Fall 2009

University of Massachusetts Amherst

David Smith

Some slides and whimsical example sentences due to Jason Eisner (JHU)

Overview

- We’ve seen several parsing algorithms
  - Backtracking, shift-reduce, CKY, Earley’s
- But what grammar should we use?
  - Refine constituents with features
  - Maintain context-free power
- Grammars are getting big! Speed parsing with
  - Grammar preindexing
  - Search pruning during parsing

3 views of a context-free rule

- generation (production): S → NP VP
- parsing (comprehension): S ← NP VP
- verification (checking): S = NP VP

- Today you should keep the third, declarative perspective in mind.

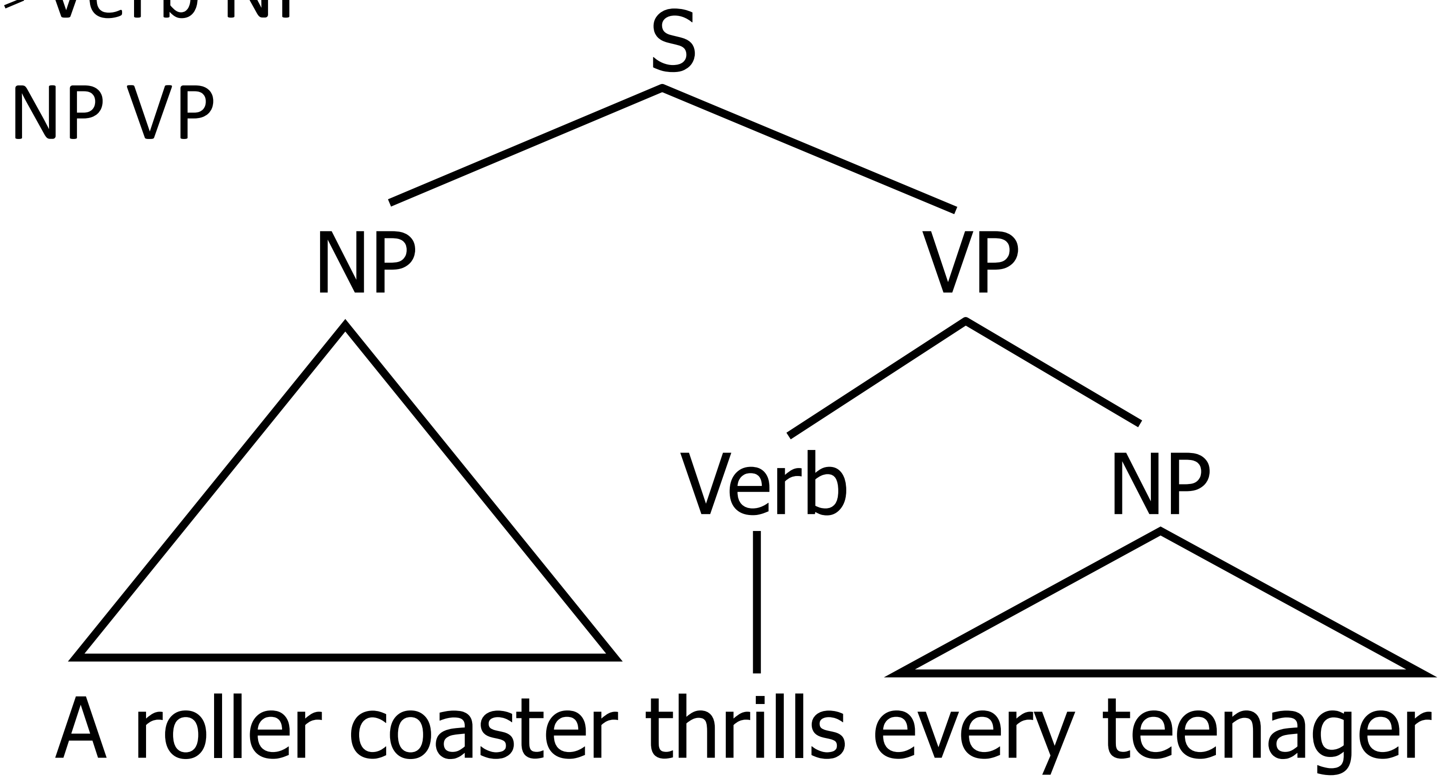
- Each phrase has
  - an interface (S) saying where it can go
  - an implementation (NP VP) saying what's in it
- To let the parts of the tree coordinate more closely with one another, enrich the interfaces:
  S[features…] = NP[features…] VP[features…]

Examples

Verb → thrills

VP→ Verb NP

S → NP VP



3 common ways to use features

morphology of a single word:

Verb[head=thrill, tense=present, num=sing, person=3,…] → thrills

projection of features up to a bigger phrase

VP[head=α, tense=β, num=γ…] → V[head=α, tense=β, num=γ…] NP

provided α is in the set TRANSITIVE-VERBS

agreement between sister phrases:

S[head=α, tense=β] → NP[num=γ,…] VP[head=α, tense=β, num=γ…]

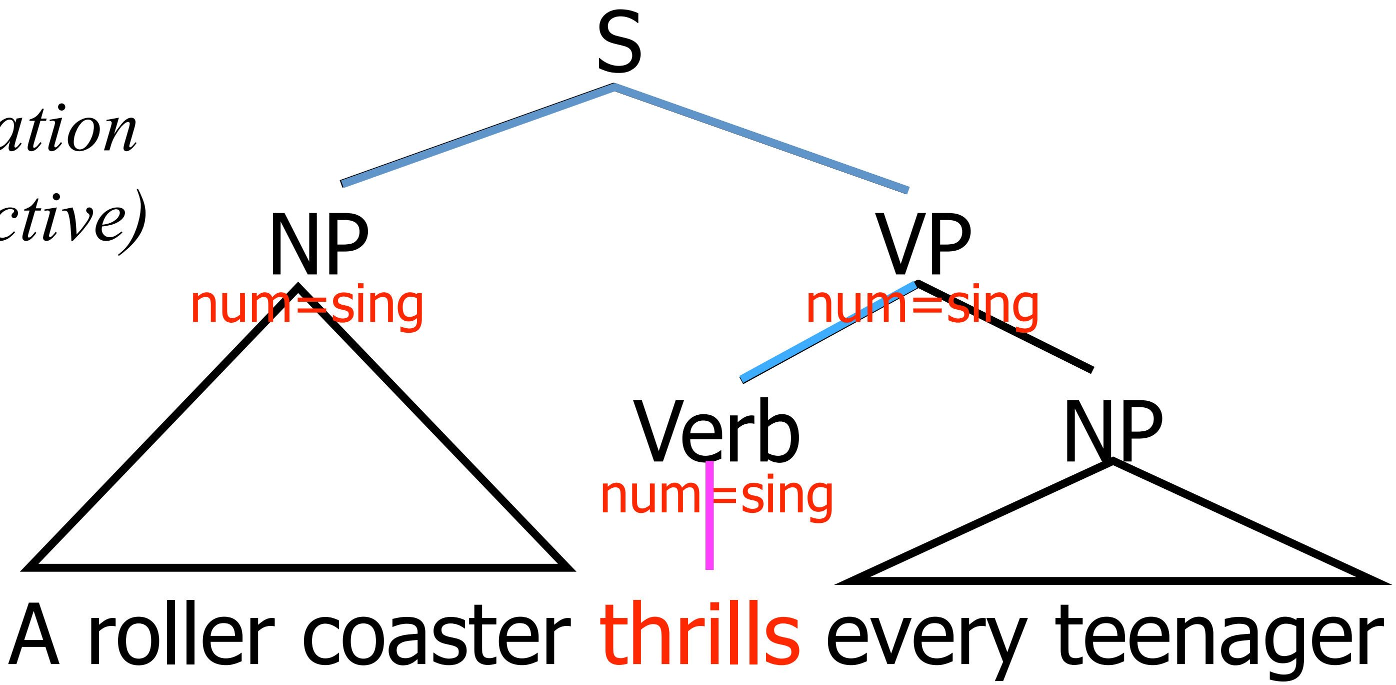
3 Common Ways to Use Features

Verb[head=thrill, tense=present, num=sing, person=3,…] → thrills

VP[head=α, tense=β, num=γ...] → V[head=α, tense=β, num=γ...] NP

S[head=α, tense=β] → NP[num=γ,...] VP[head=α, tense=β, num=γ...]

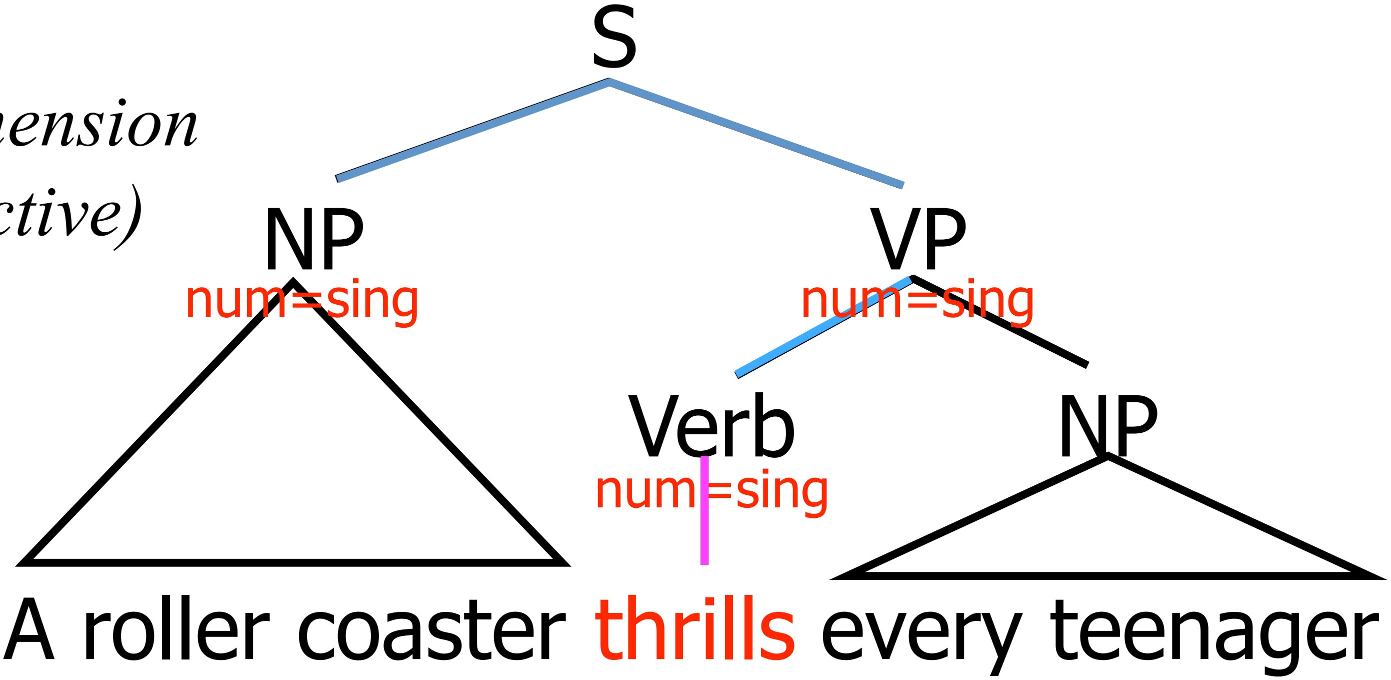
(generation perspective)

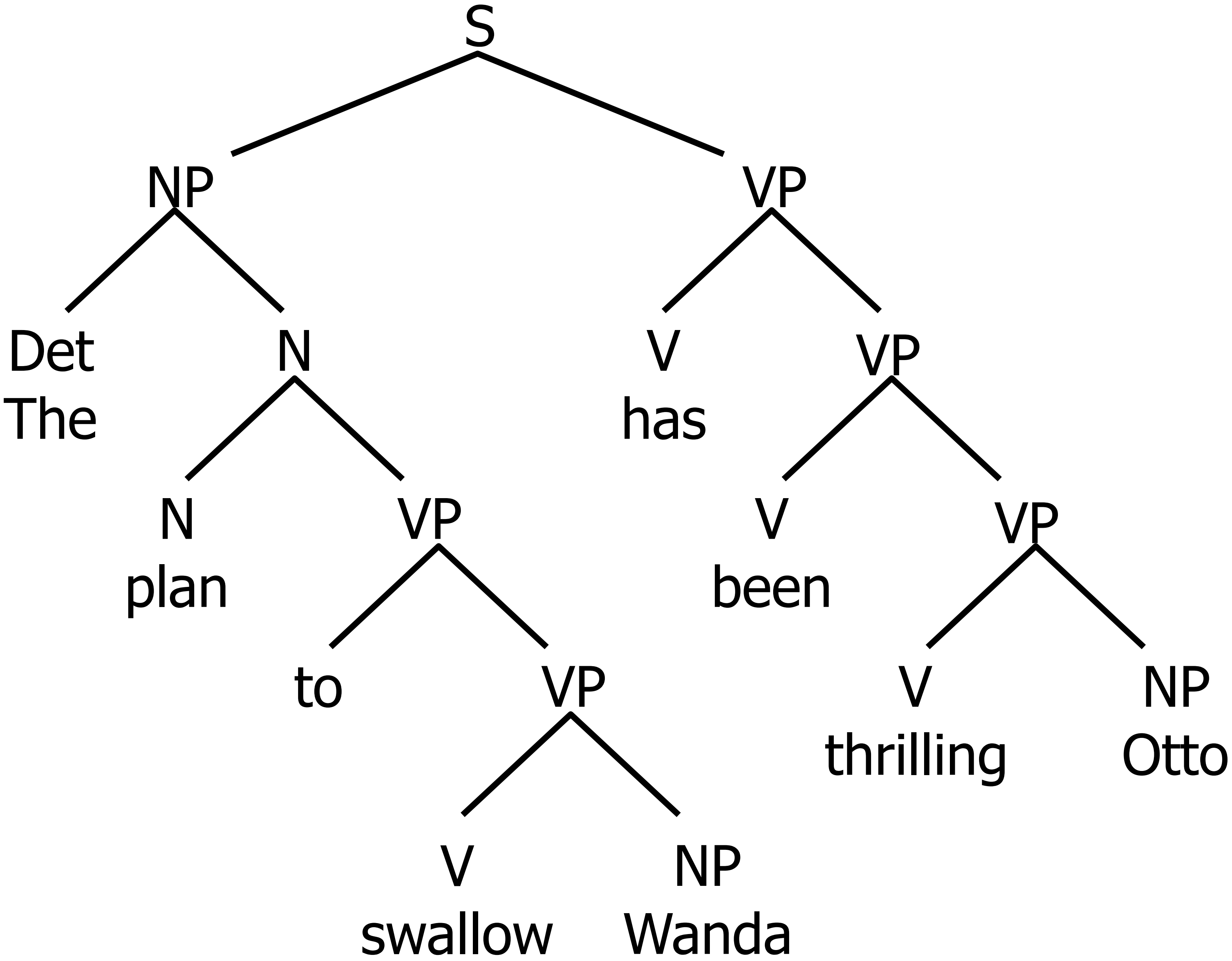


3 Common Ways to Use Features

Verb[head=thrill, tense=present, num=sing, person=3,…] → thrills
VP[head=α, tense=β, num=γ…] → V[head=α, tense=β, num=γ…] NP
S[head=α, tense=β] → NP[num=γ,…] VP[head=α, tense=β, num=γ…]

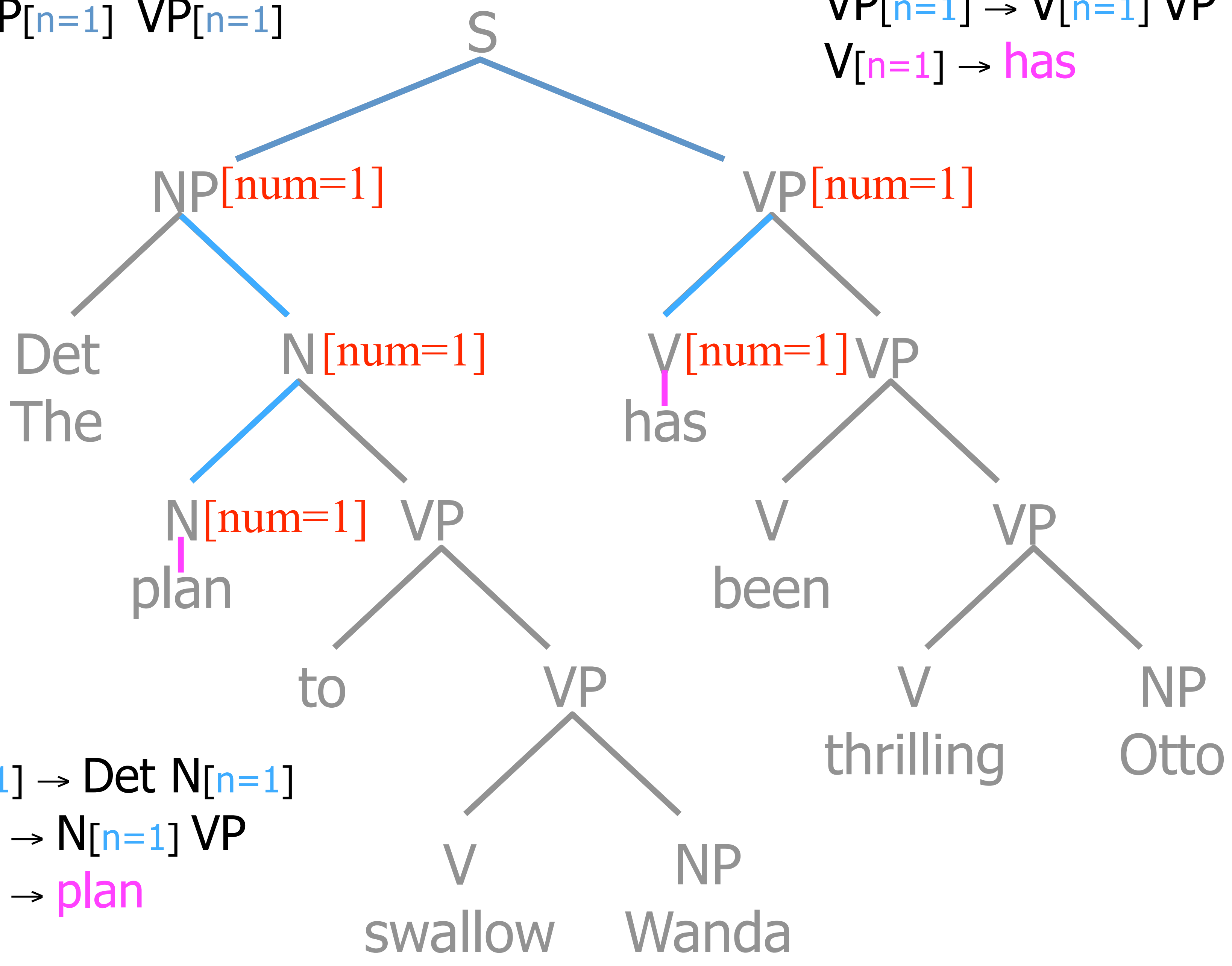
(comprehension perspective)




S
NP
VP
Det
The
N
V
has
VP
N
plan
VP
V
been
VP
to
VP
V
thrilling
NP
Otto
V
swallow
NP
Wanda

S→ NP[n=1] VP[n=1]

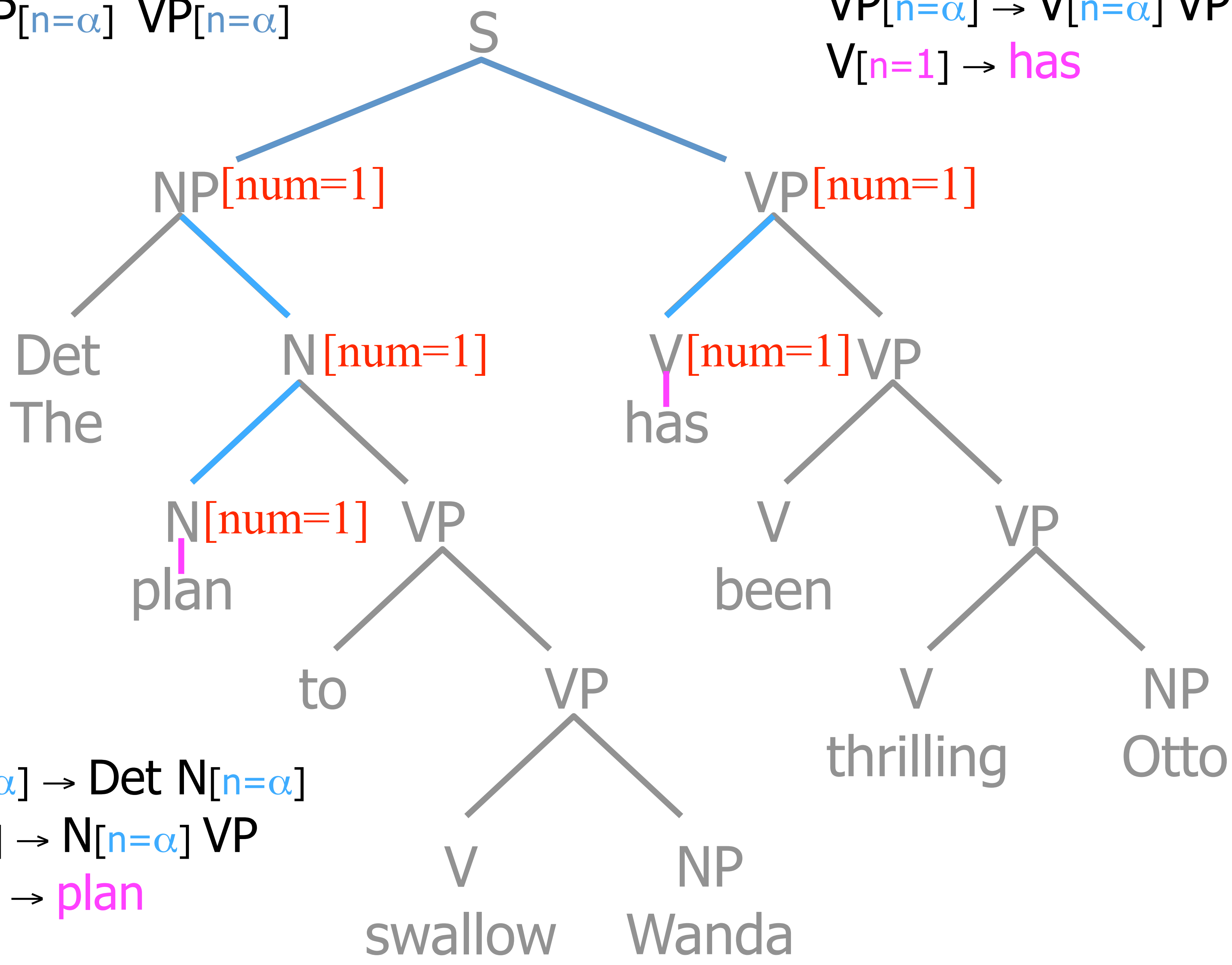
VP[n=1] → V[n=1] VP
V[n=1] → has



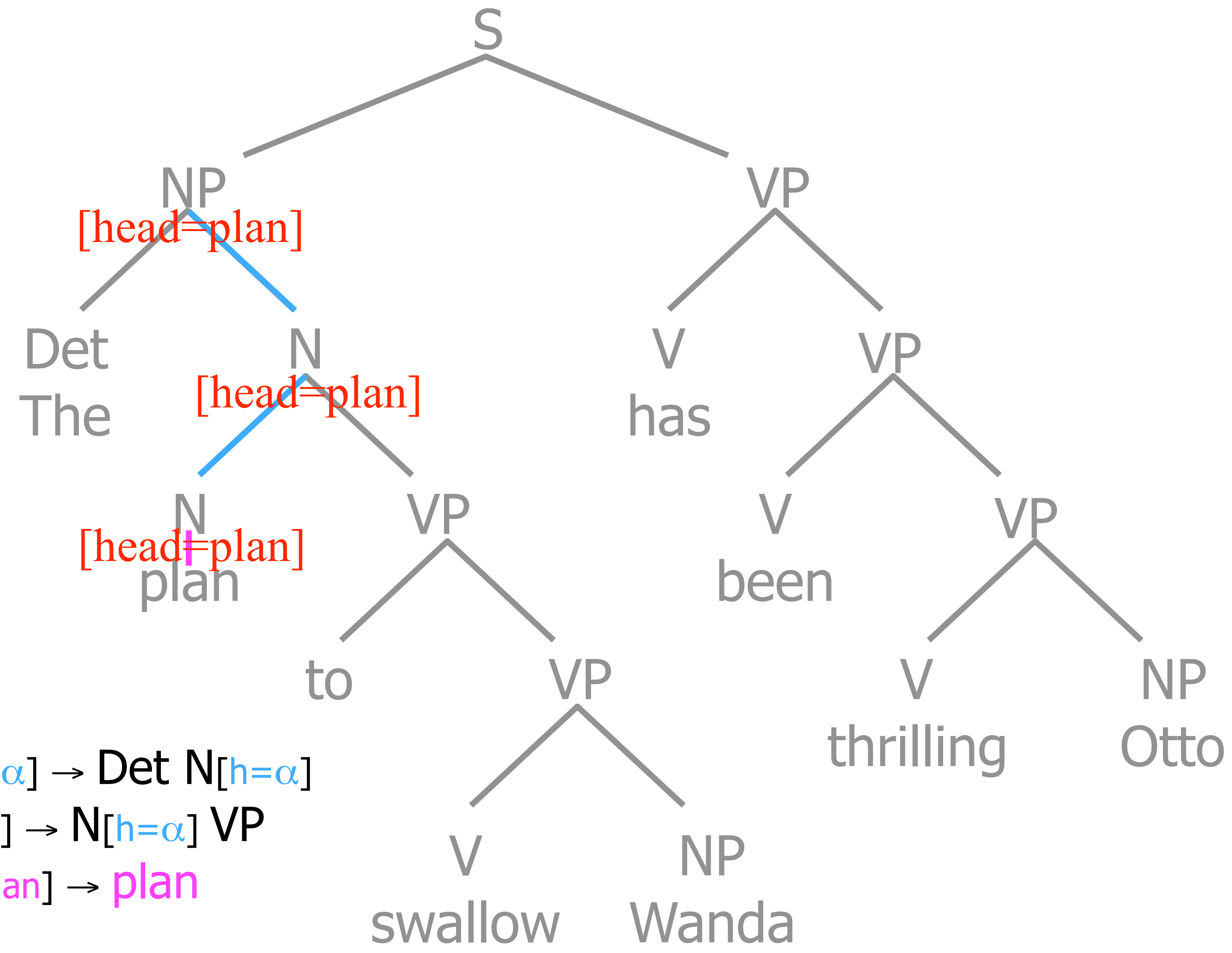
NP[n=1] → Det N[n=1]
N[n=1] → N[n=1] VP
N[n=1] → plan

S → NP[n=α] VP[n=α]

VP[n=α] → V[n=α] VP
V[n=1] → has



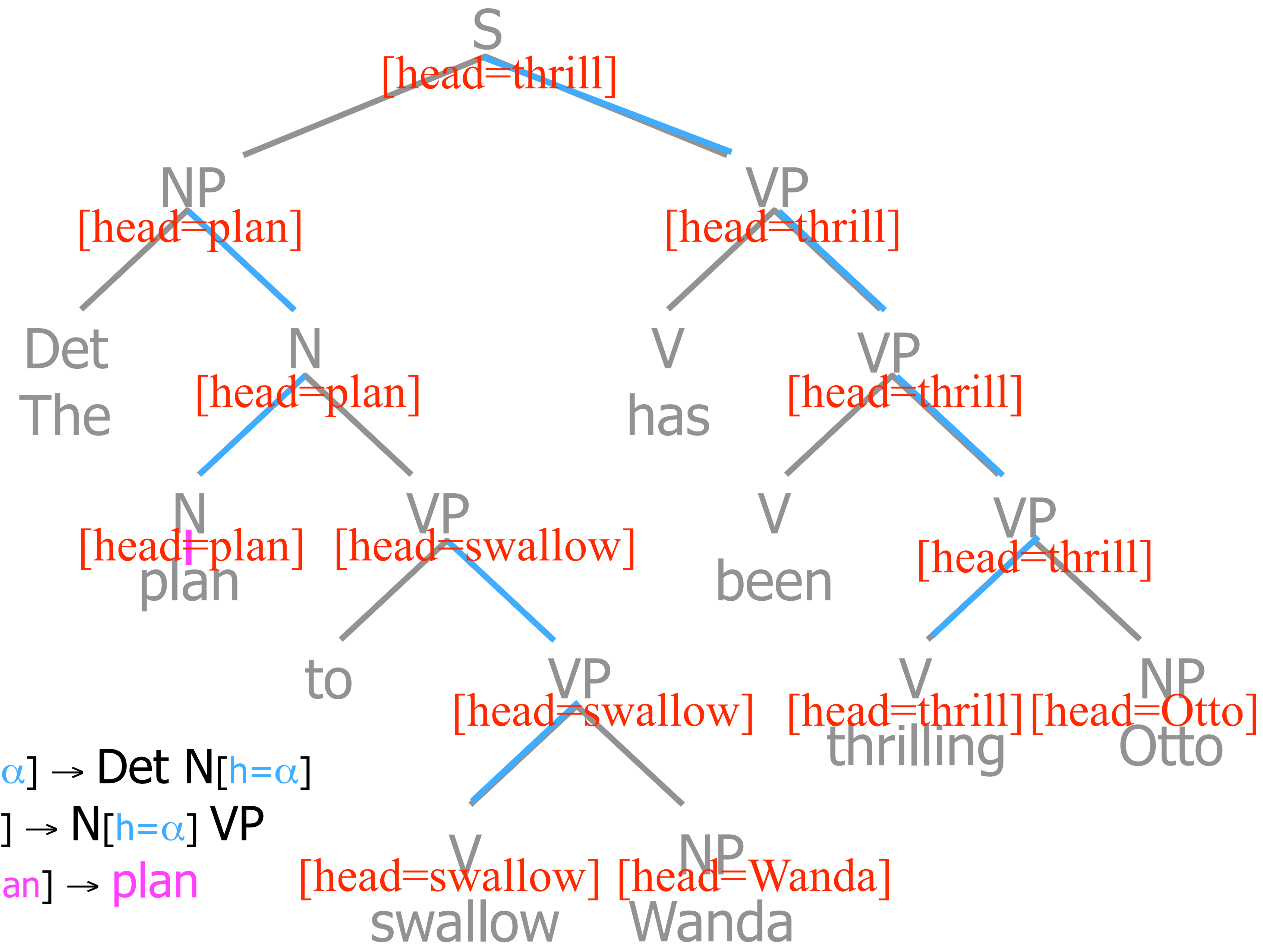
NP[n=α] → Det N[n=α]
N[n=α] → N[n=α] VP
N[n=1] → plan



NP[h=α] → Det N[h=α]

N[h=α] → N[h=α] VP

N[h=plan] → plan

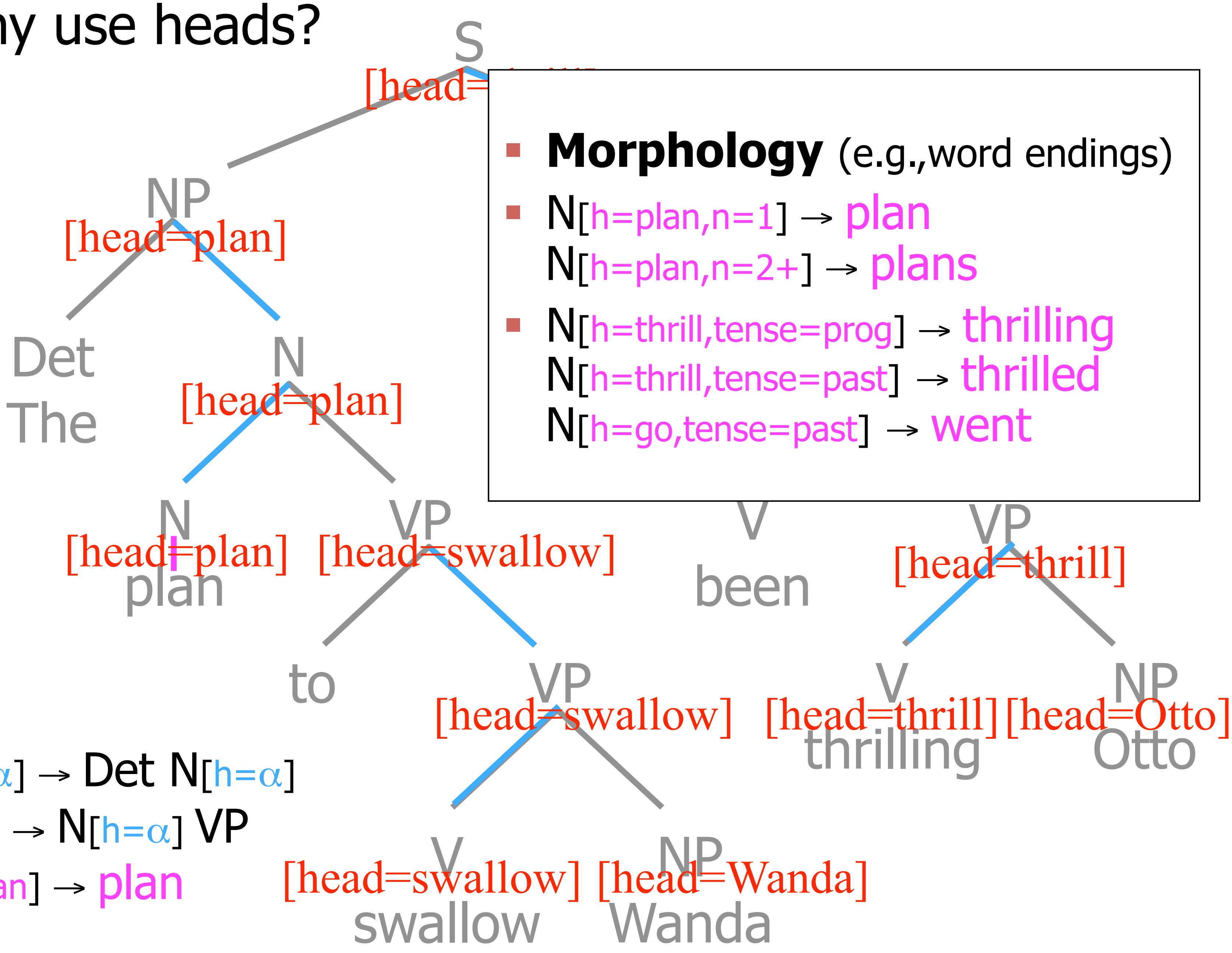


$\mathrm{NP}_{[h=\alpha]} \rightarrow \mathrm{Det}\ \mathrm{N}_{[h=\alpha]}$

$\mathrm{N}_{[h=\alpha]} \rightarrow \mathrm{N}_{[h=\alpha]}\ \mathrm{VP}$

$\mathrm{N}_{[h=\mathrm{plan}]} \rightarrow \mathrm{plan}$

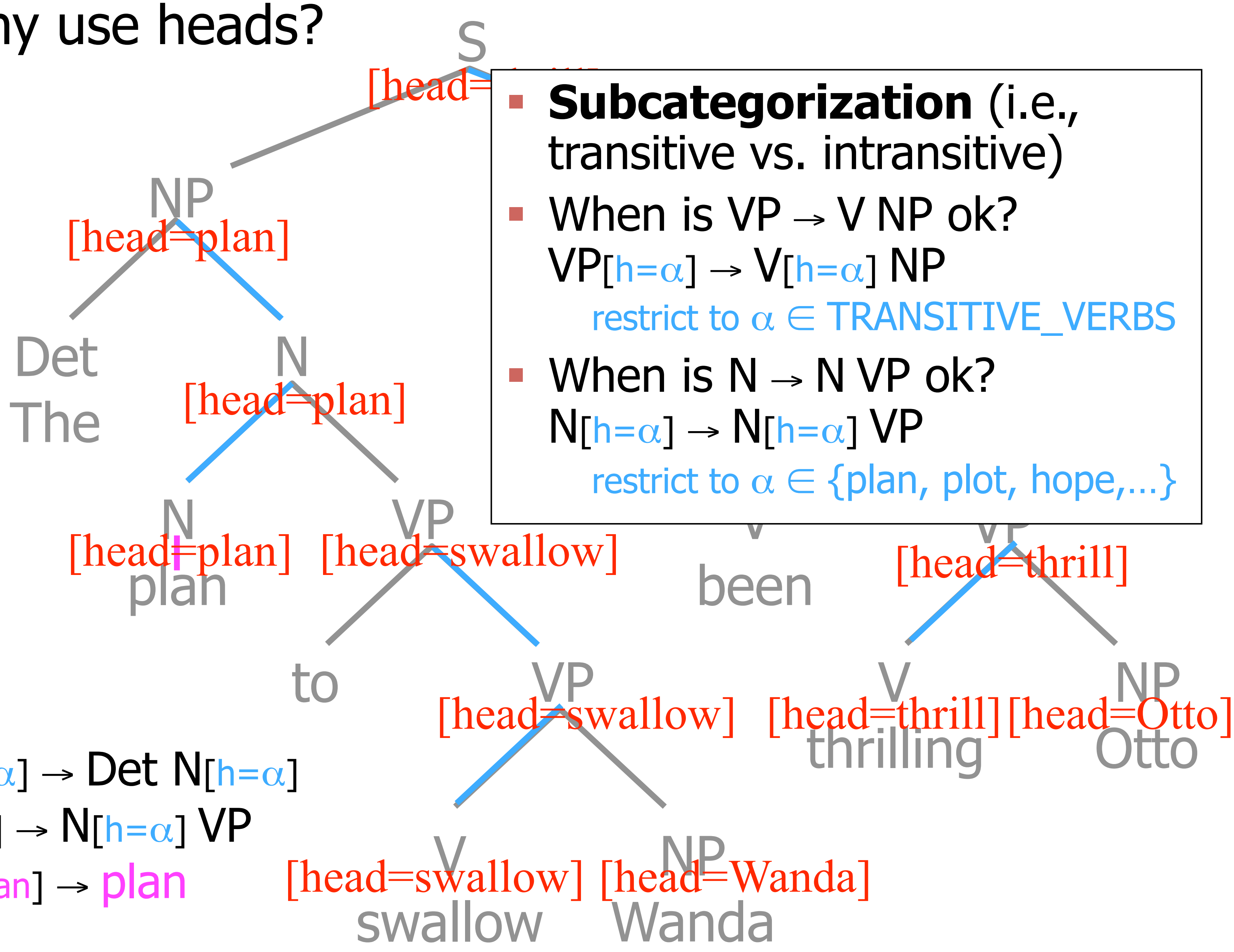
- Why use heads?



- **Morphology** (e.g.,word endings)
- N[h=plan,n=1] → plan
  N[h=plan,n=2+] → plans
- N[h=thrill,tense=prog] → thrilling
  N[h=thrill,tense=past] → thrilled
  N[h=go,tense=past] → went

NP[h=α] → Det N[h=α]
N[h=α] → N[h=α] VP
N[h=plan] → plan

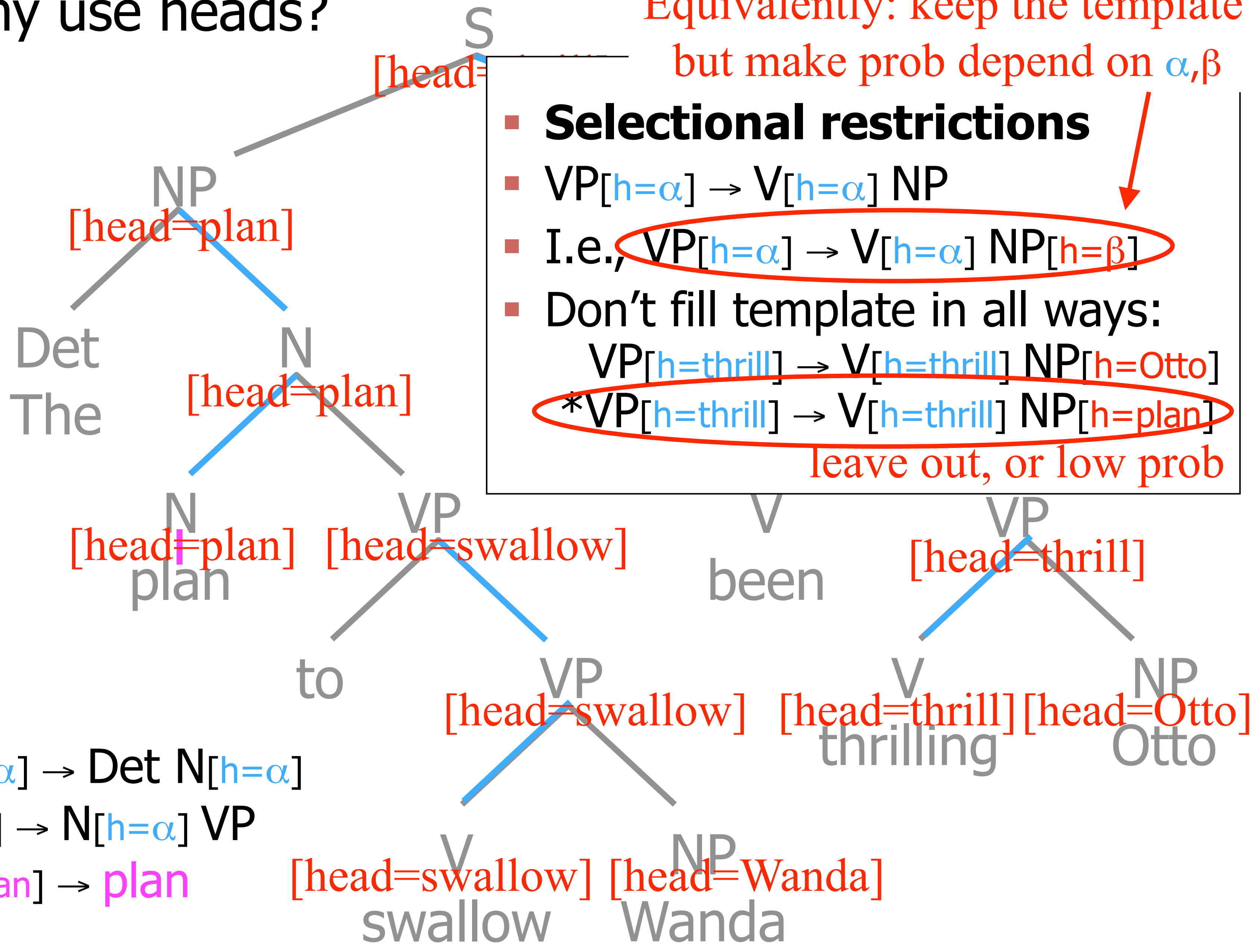
- Why use heads?



- **Subcategorization** (i.e., transitive vs. intransitive)
- When is VP → V NP ok?
  VP[h=α] → V[h=α] NP
  restrict to α ∈ TRANSITIVE_VERBS
- When is N → N VP ok?
  N[h=α] → N[h=α] VP
  restrict to α ∈ {plan, plot, hope,...}

NP[h=α] → Det N[h=α]
N[h=α] → N[h=α] VP
N[h=plan] → plan

- Why use heads?



Equivalently: keep the template but make prob depend on α,β

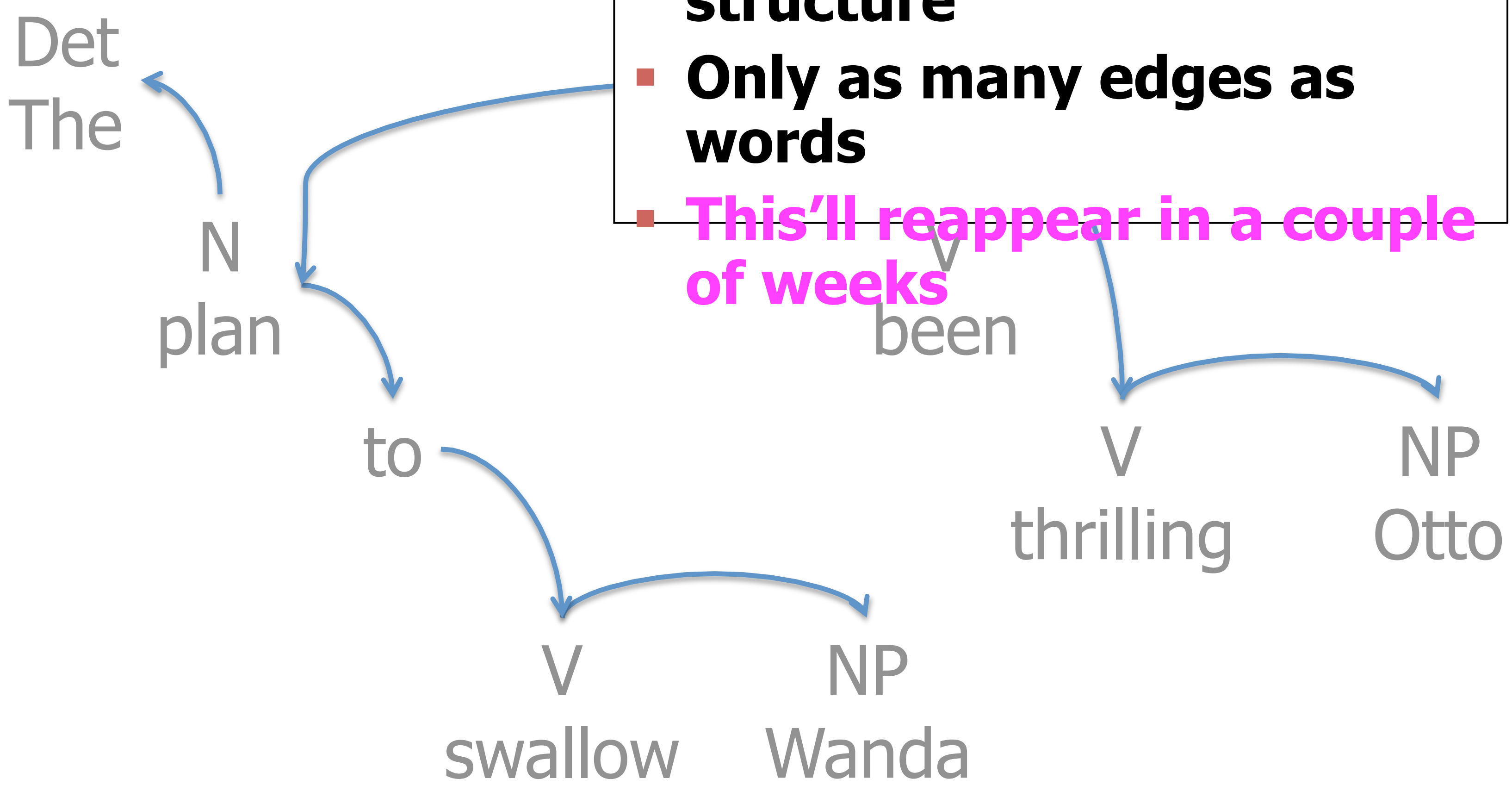
- **Selectional restrictions**
- VP[h=α] → V[h=α] NP
- I.e., VP[h=α] → V[h=α] NP[h=β]
- Don't fill template in all ways:
  VP[h=thrill] → V[h=thrill] NP[h=Otto]
  *VP[h=thrill] → V[h=thrill] NP[h=plan]

leave out, or low prob

NP[h=α] → Det N[h=α]
N[h=α] → N[h=α] VP
N[h=plan] → plan

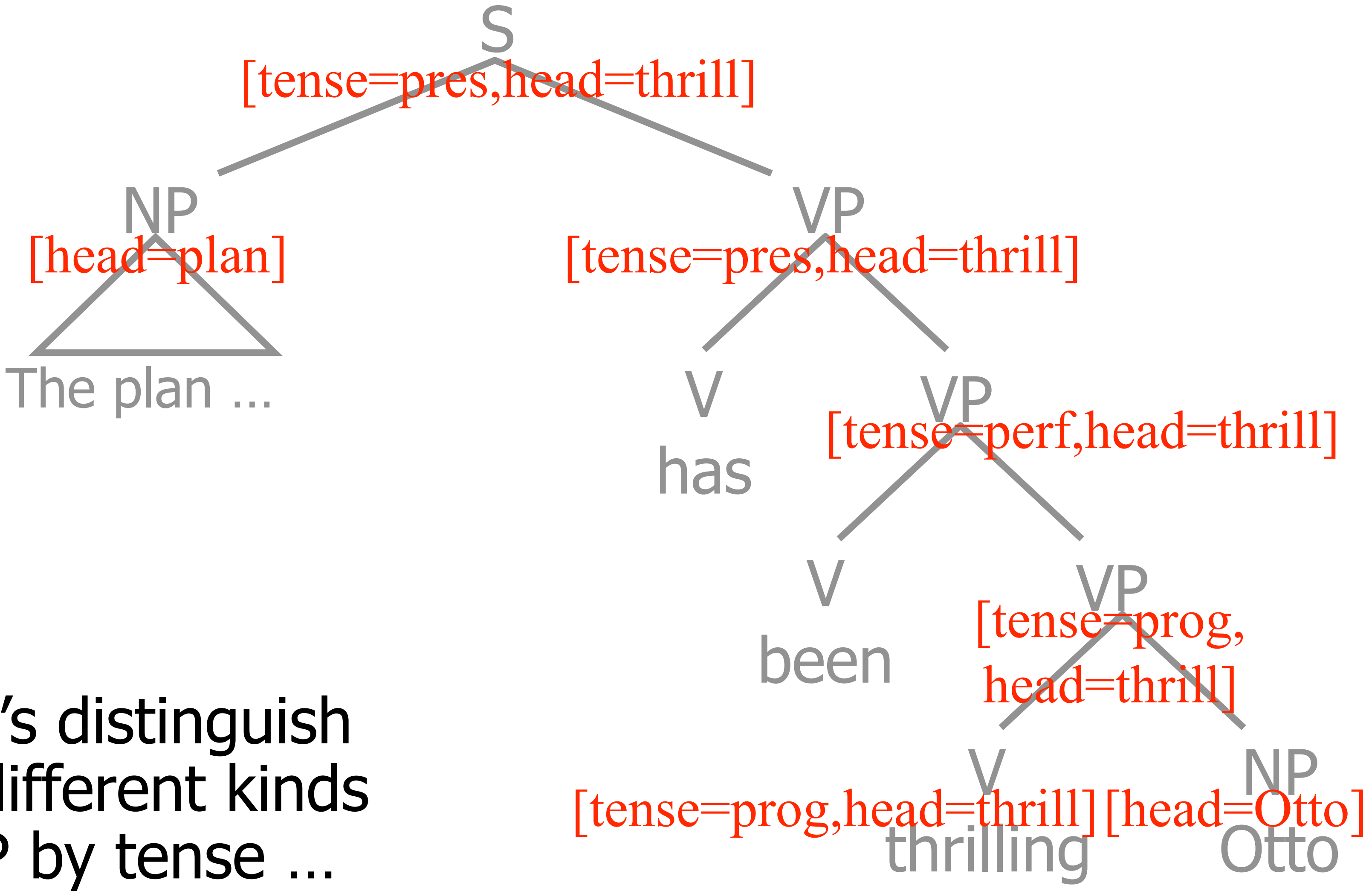
- Why use anything *but* heads?

- **Dependency grammar**
  - **Aka “word grammar”**
- **Maps closely to argument structure**
- **Only as many edges as words**
- **This’ll reappear in a couple of weeks**

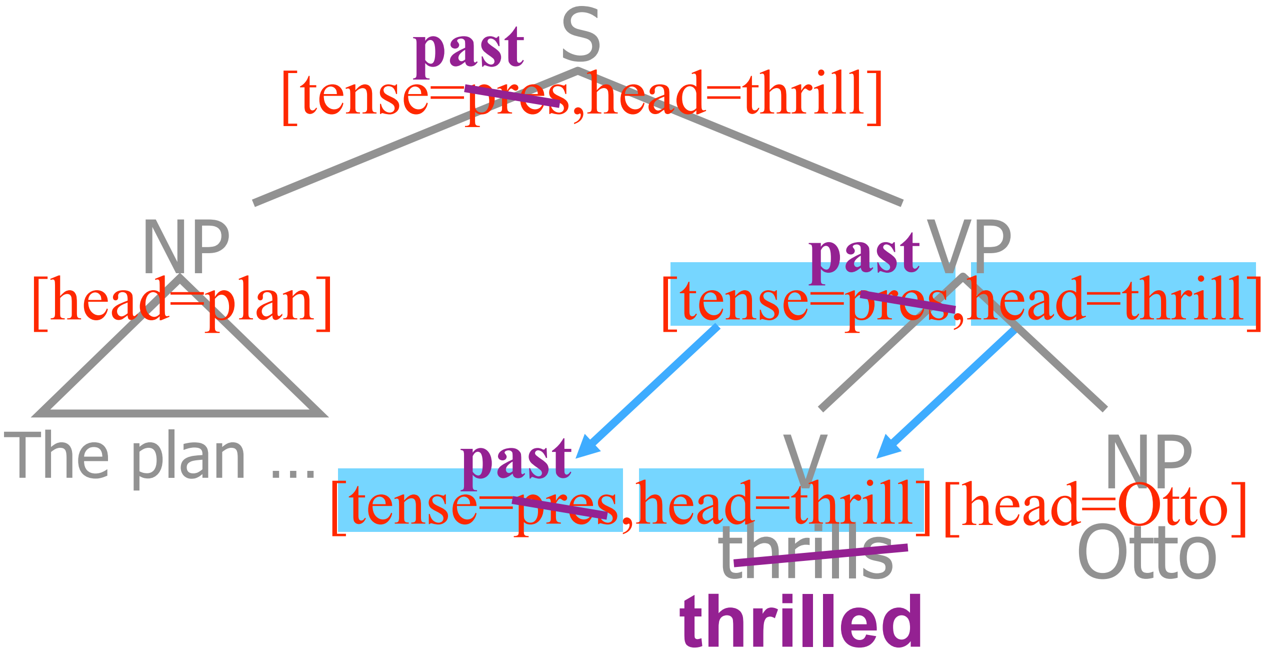


Part of the English Tense/Aspect System

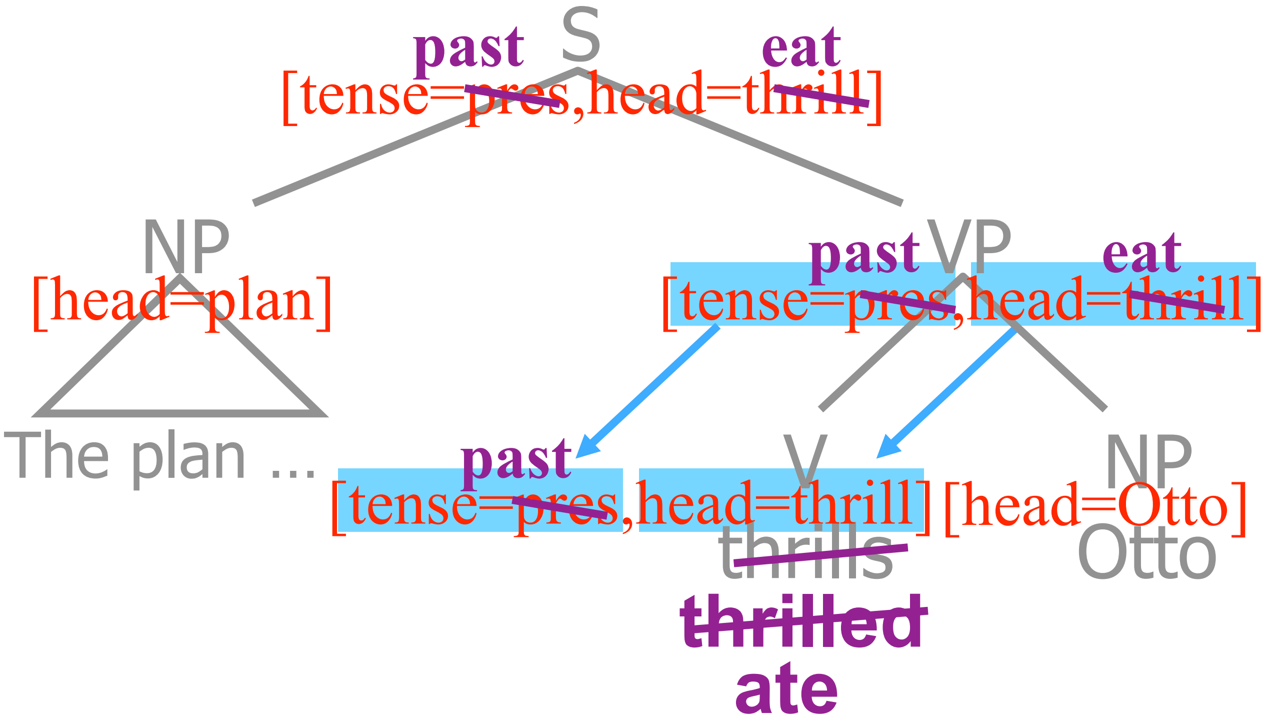
	Present	Past	Future	Infinitive
Simple	eats	ate	will eat	to eat
Perfect	has eaten	had eaten	will have eaten	to have eaten
progressive	is eating	was eating	will be eating	to be eating
Perfect+ progressive	has been eating	had been eating	will have been eating	to have been eating



- Let’s distinguish the different kinds of VP by tense …

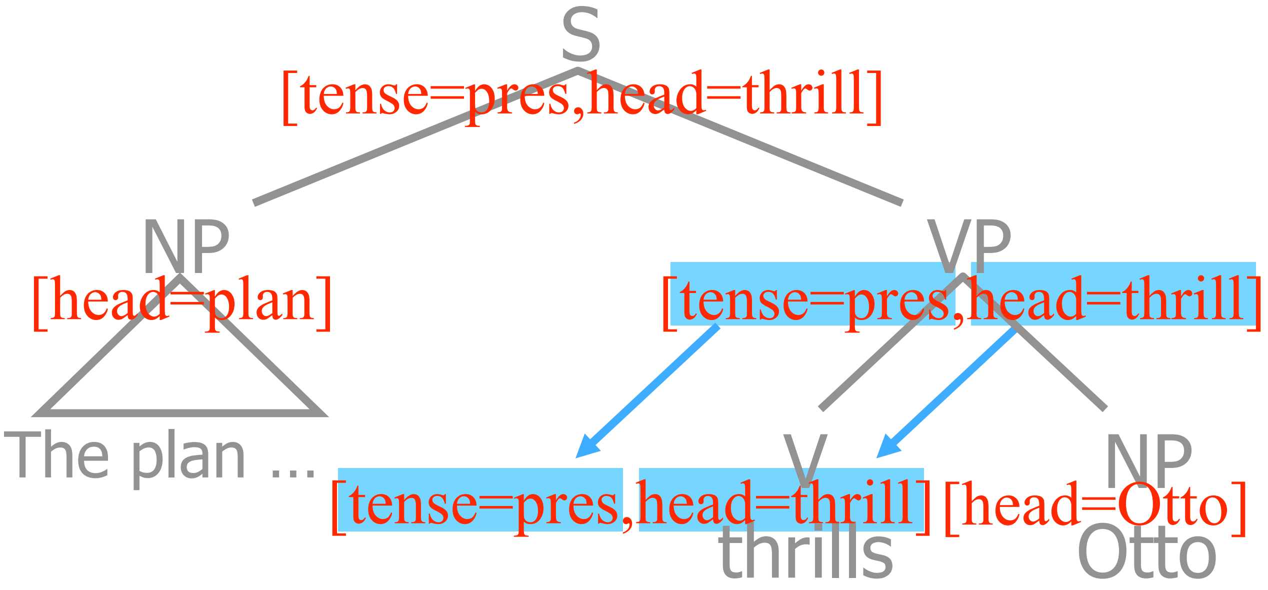

S
past
[tense=~~pres~~,head=thrill]
NP
[head=plan]
The plan …
past
VP
[tense=~~pres~~,head=thrill]
past
[tense=~~pres~~,head=thrill]
V
~~thrills~~
thrilled
NP
[head=Otto]
Otto

- **Past** ~~Present~~ tense

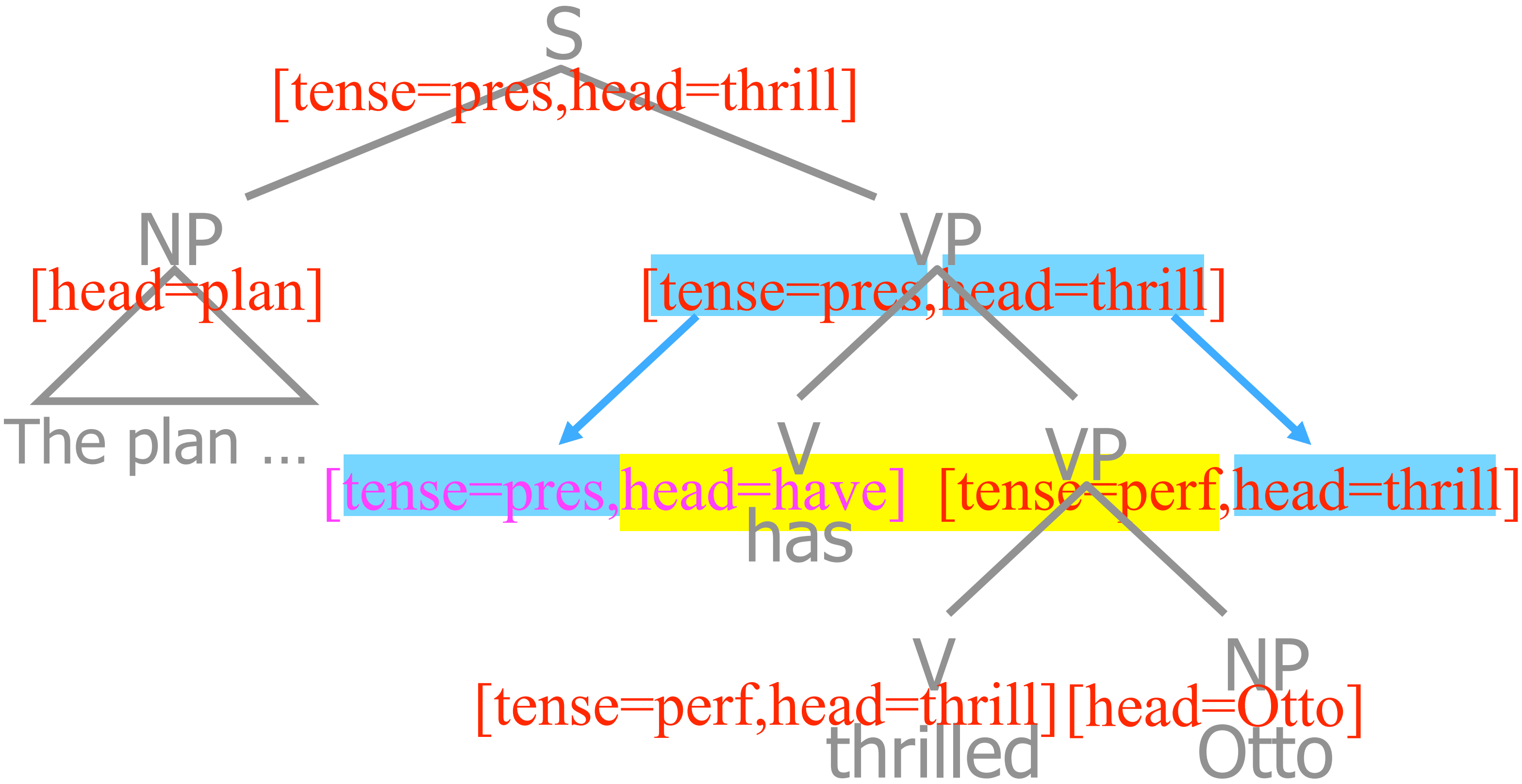

S
past
eat
[tense=~~pres~~,head=~~thrill~~]
NP
[head=plan]
The plan ...
VP
past
eat
[tense=~~pres~~,head=~~thrill~~]
V
past
[tense=~~pres~~,head=thrill]
~~thrills~~
~~thrilled~~
ate
NP
[head=Otto]
Otto

Past

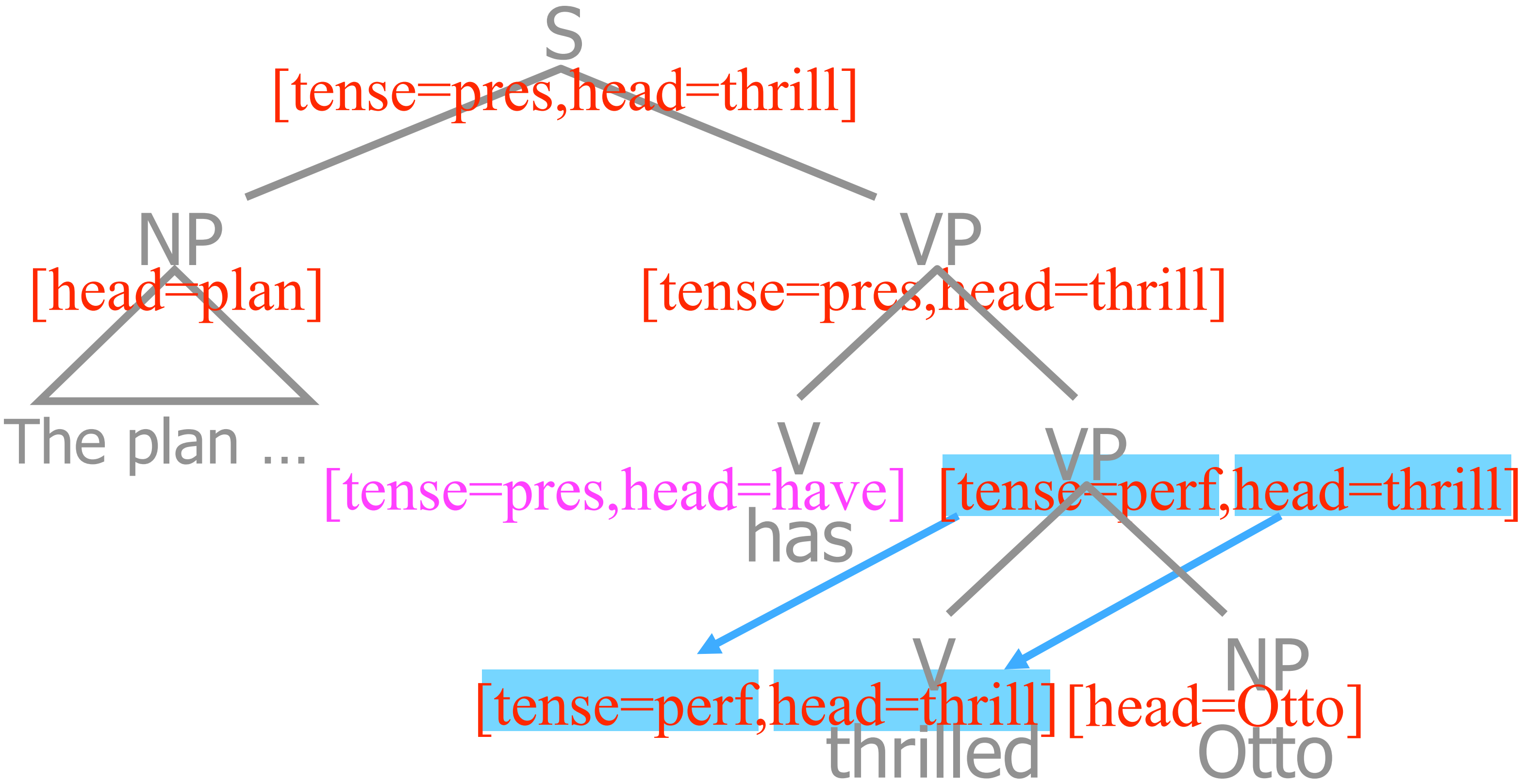
- ~~Present~~ tense



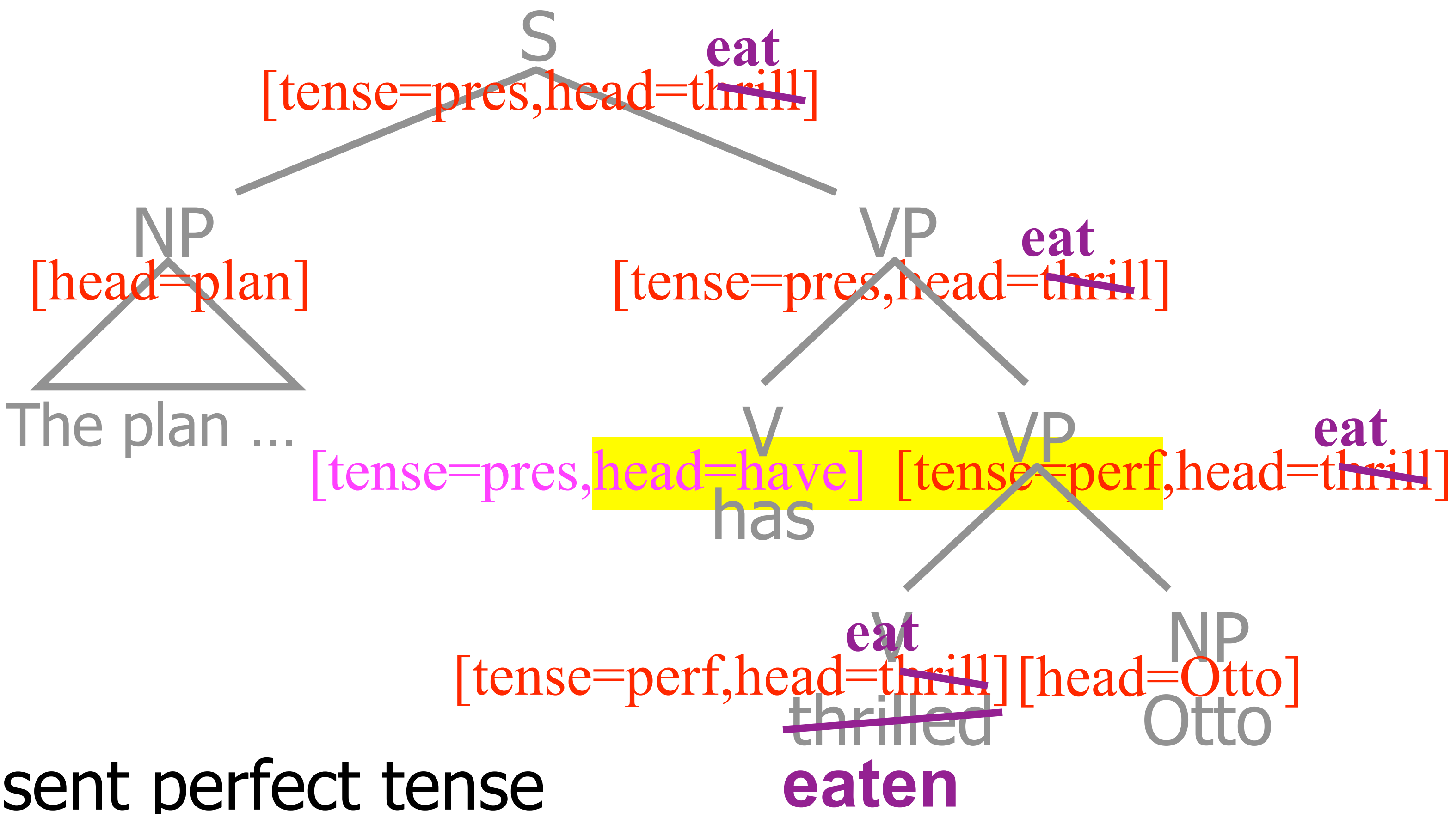
- Present tense (again)



- Present perfect tense

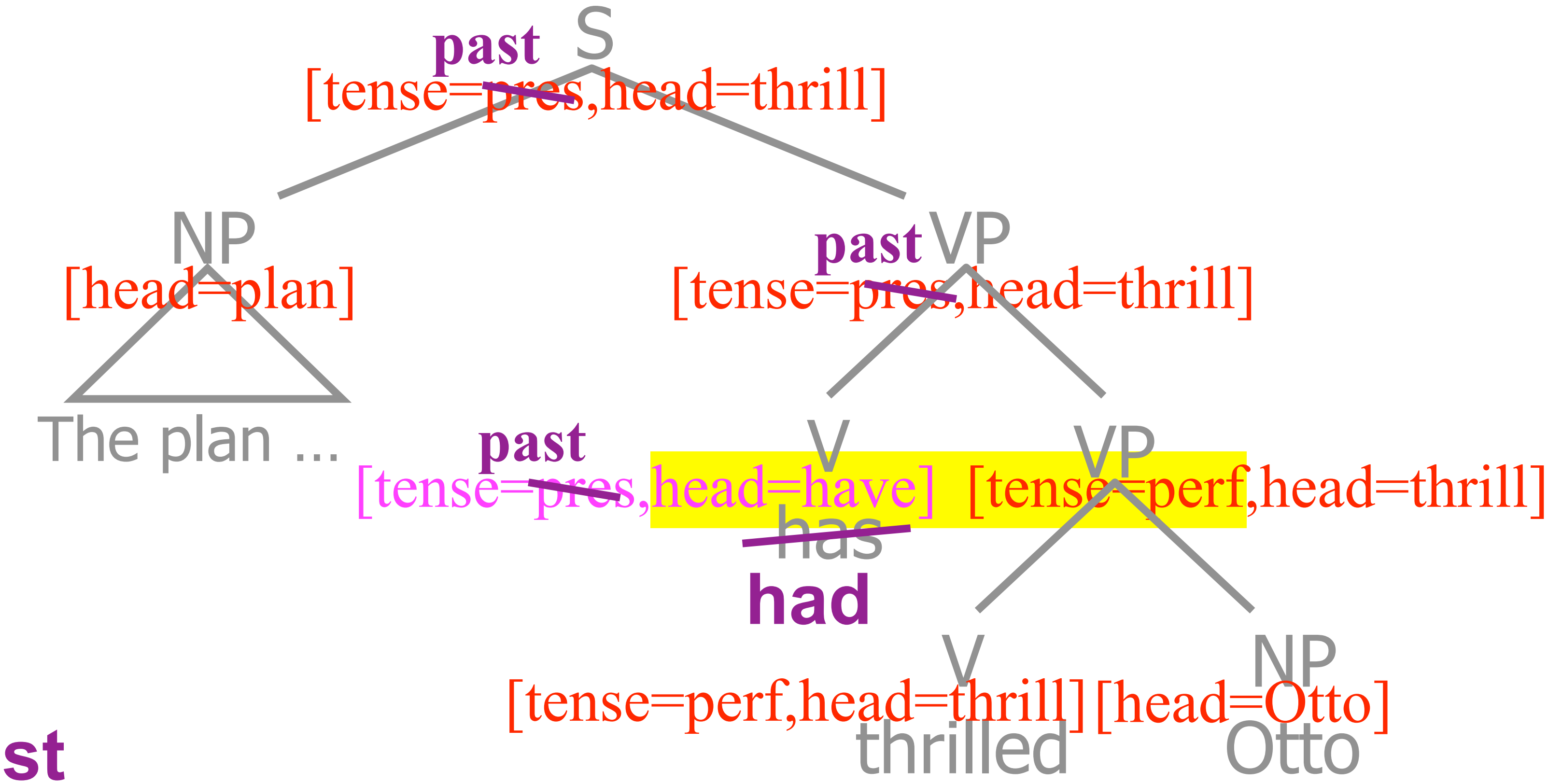


- Present perfect tense


S
[tense=pres,head=~~thrill~~]
eat
NP
[head=plan]
The plan …
VP
[tense=pres,head=~~thrill~~]
eat
V
[tense=pres,head=have]
has
VP
[tense=perf,head=~~thrill~~]
eat
V
[tense=perf,head=~~thrill~~]
eat
~~thrilled~~
eaten
NP
[head=Otto]
Otto

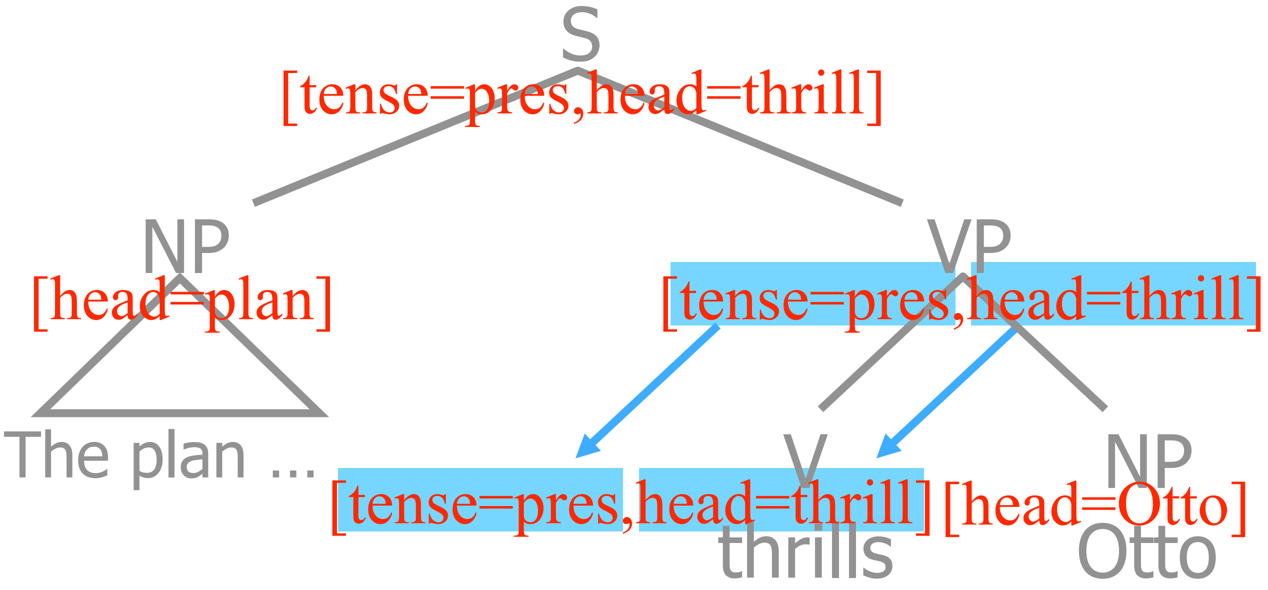
- Present perfect tense
- The yellow material makes it a perfect tense – what effects?

not **ate** – why?

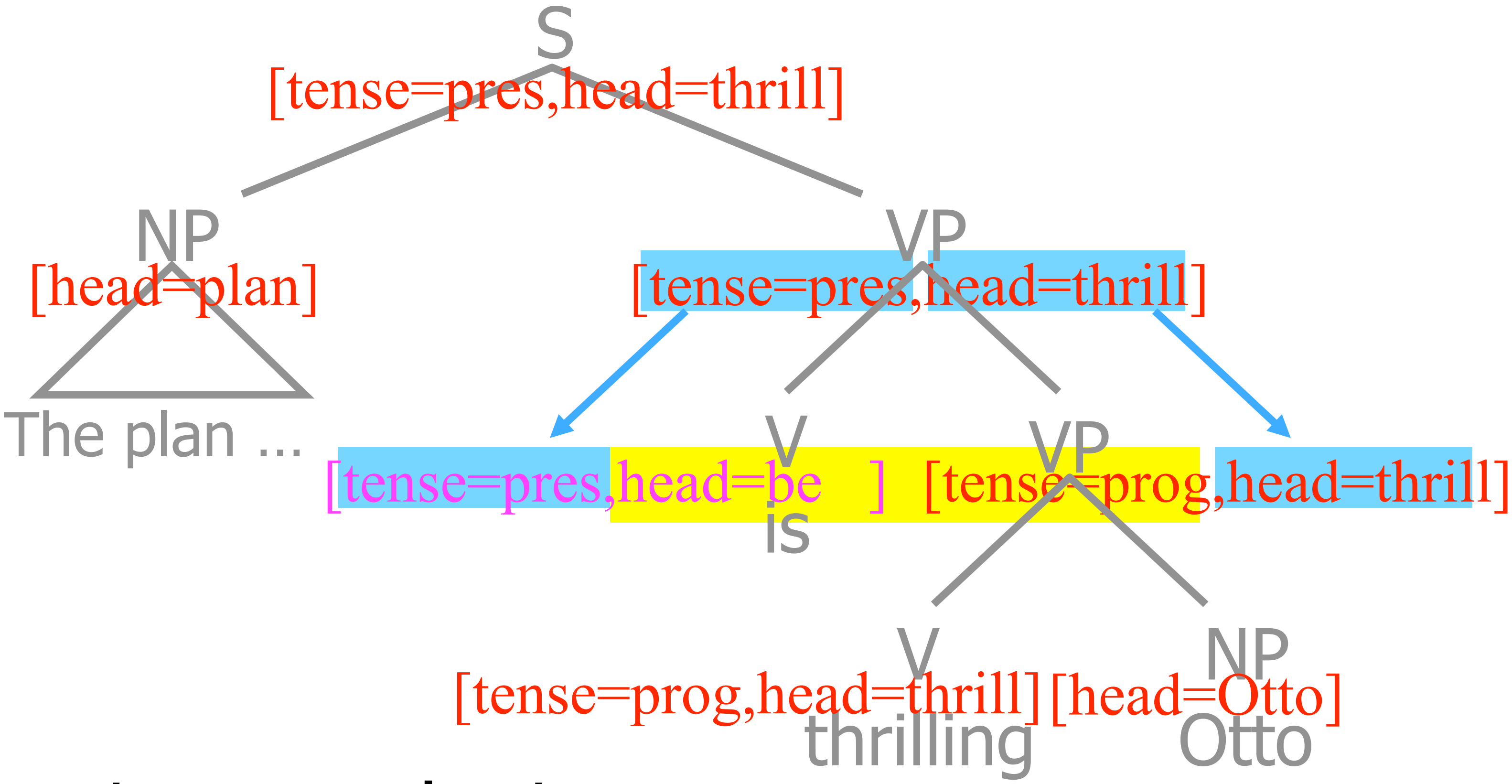


Past

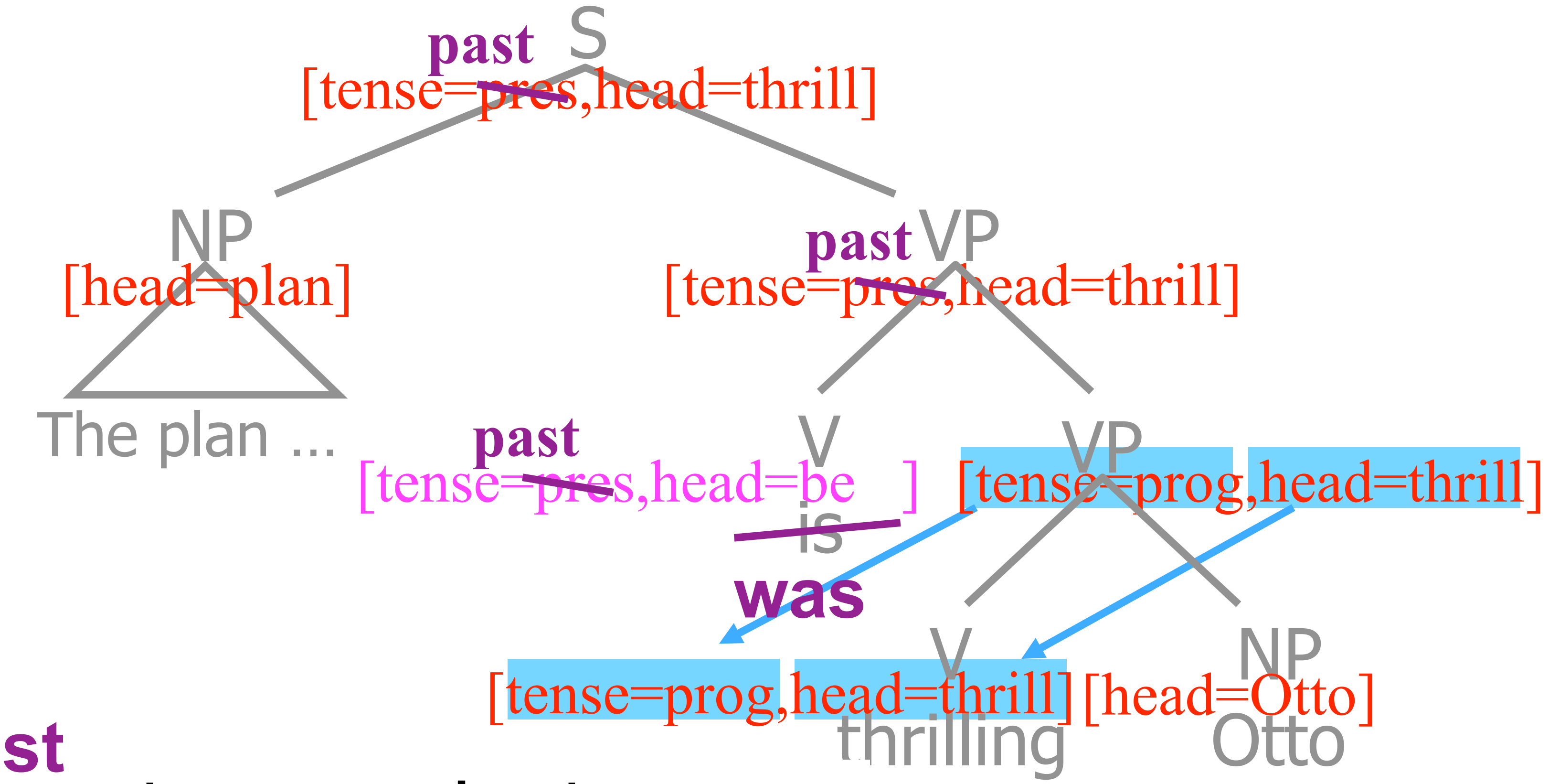
- ~~Present~~ perfect tense



- Present tense (again)

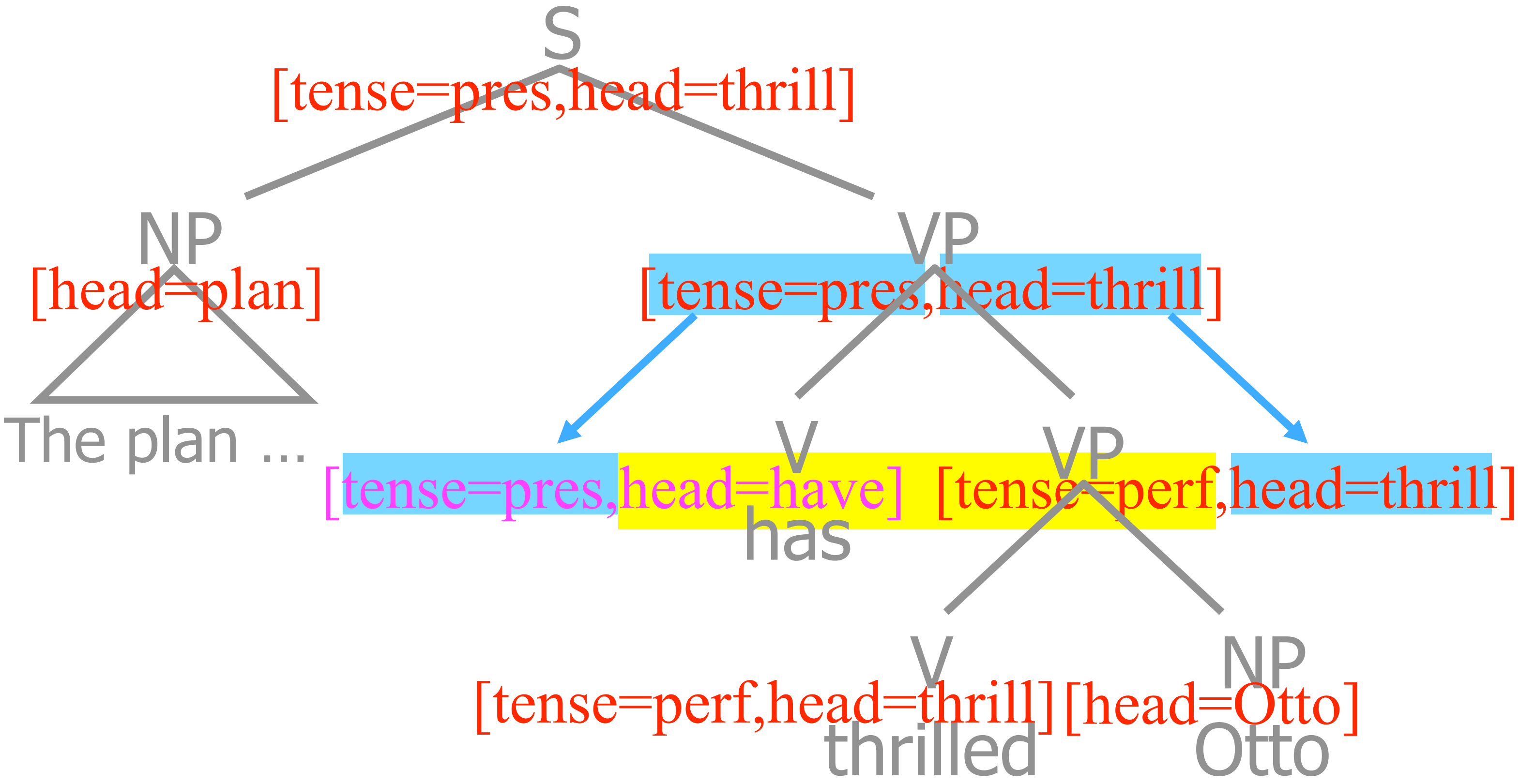


- Present progressive tense



Past

- ~~Present~~ progressive tense



S

[tense=pres,head=thrill]

NP

[head=plan]

The plan …

VP

[tense=pres,head=thrill]

V

[tense=pres,head=have]

has

VP

[tense=perf,head=thrill]

V

[tense=perf,head=thrill]

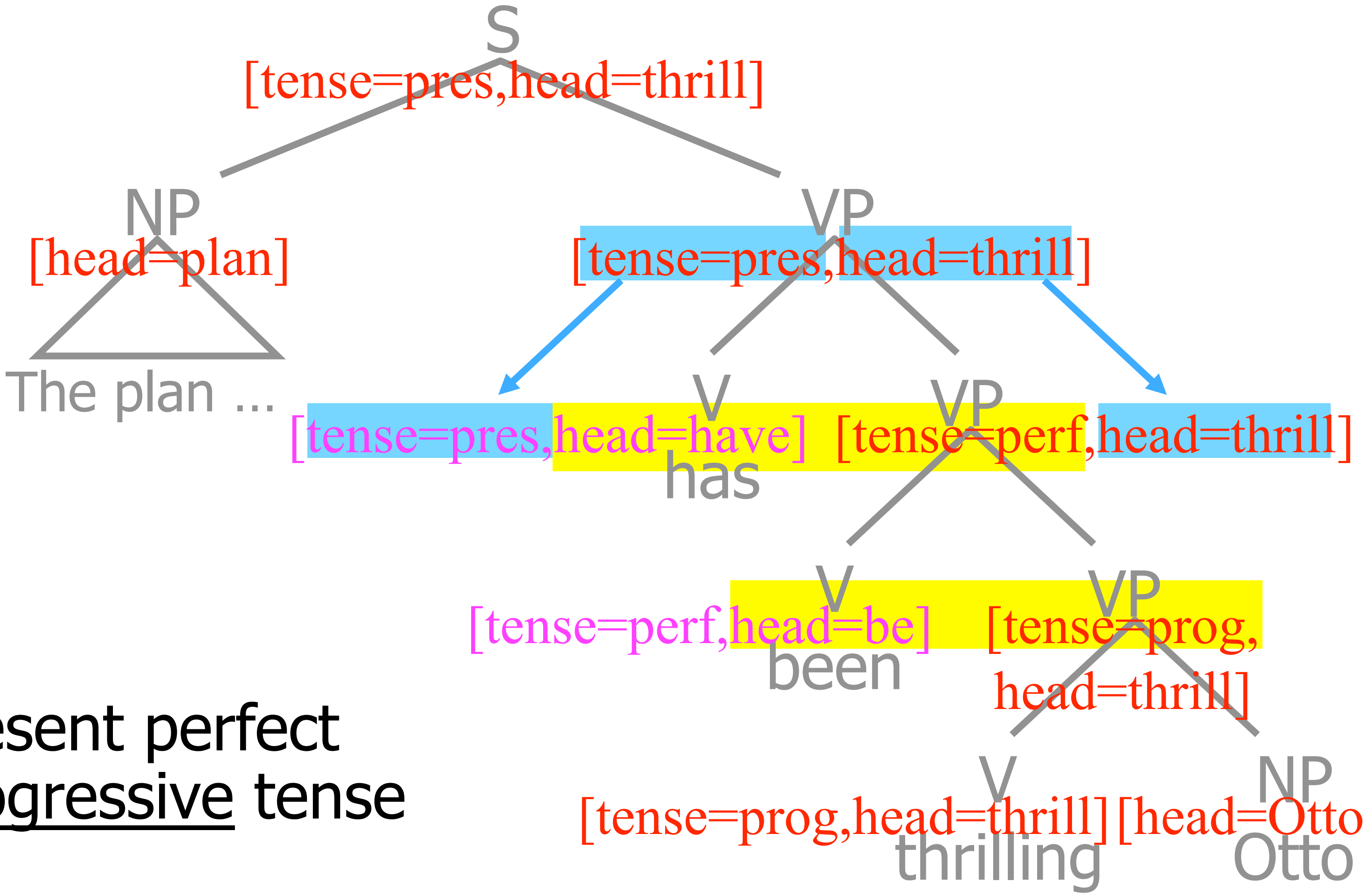
thrilled

NP

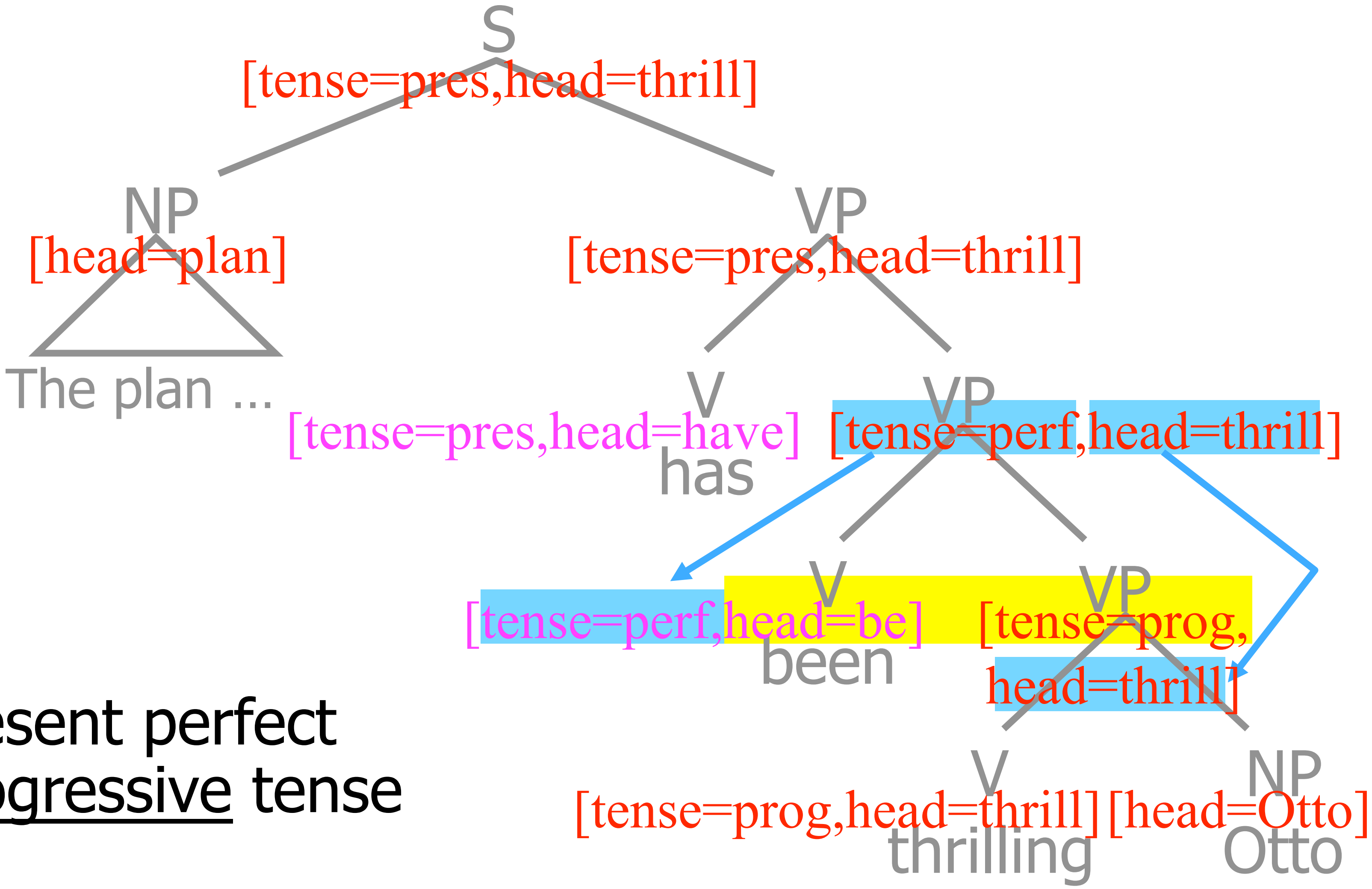
[head=Otto]

Otto

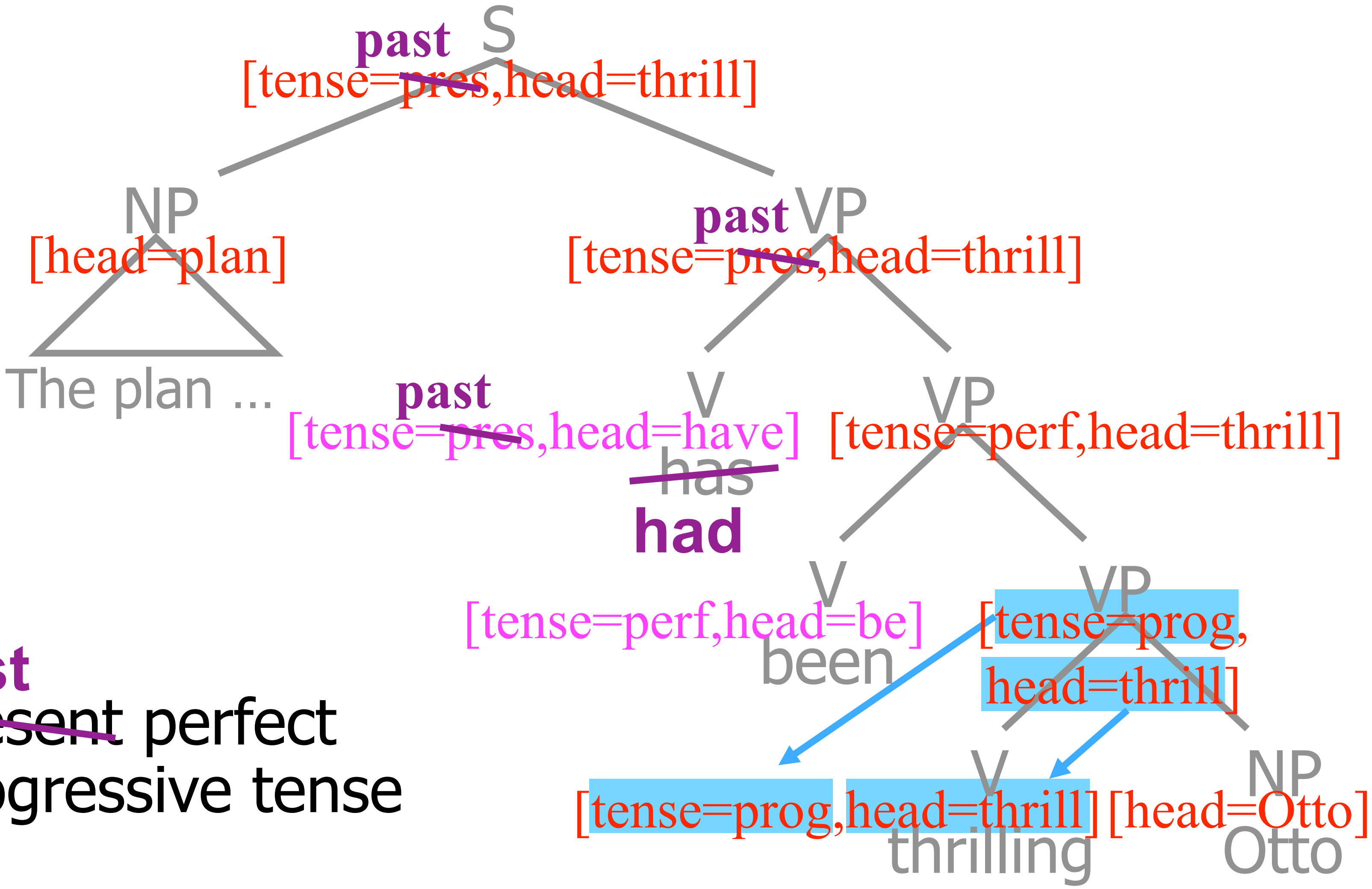
- Present perfect tense (again)



- Present perfect <u>progressive</u> tense

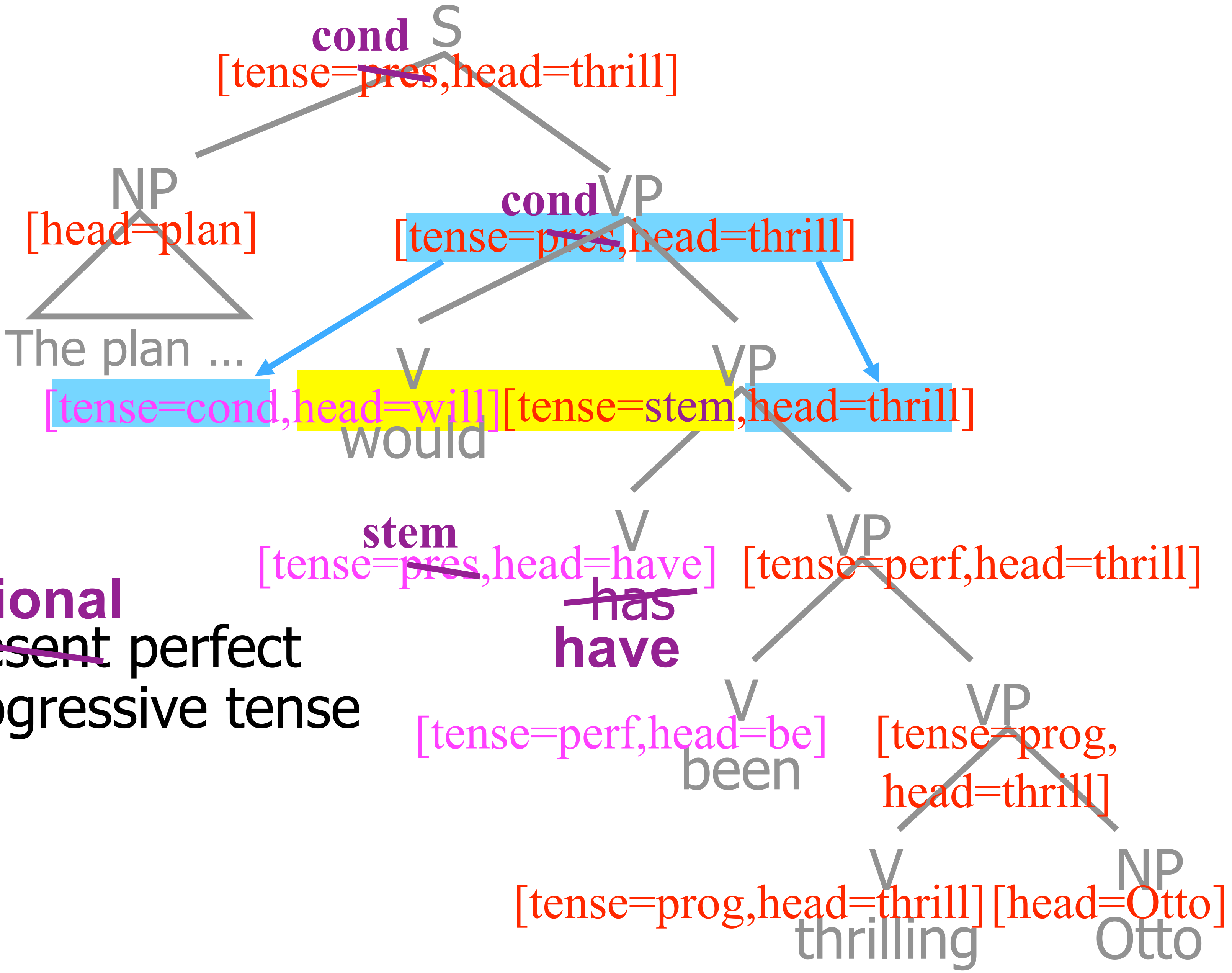


- Present perfect progressive tense



Past

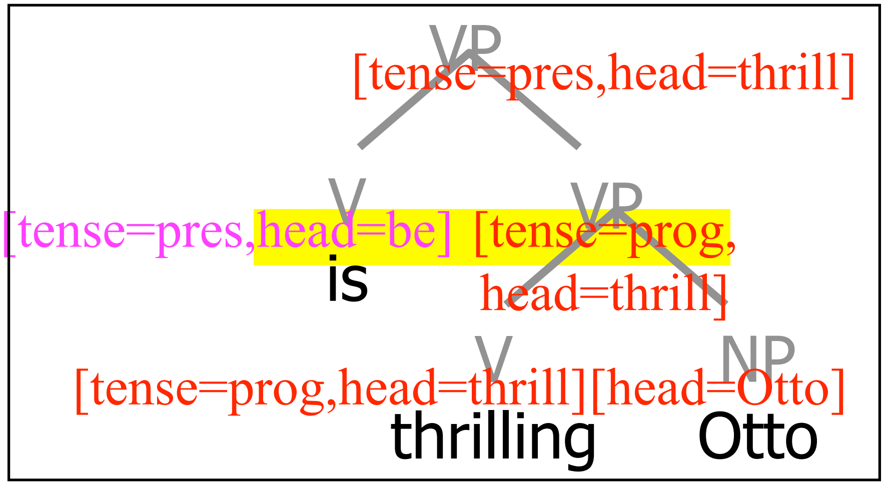
- ~~Present~~ perfect progressive tense

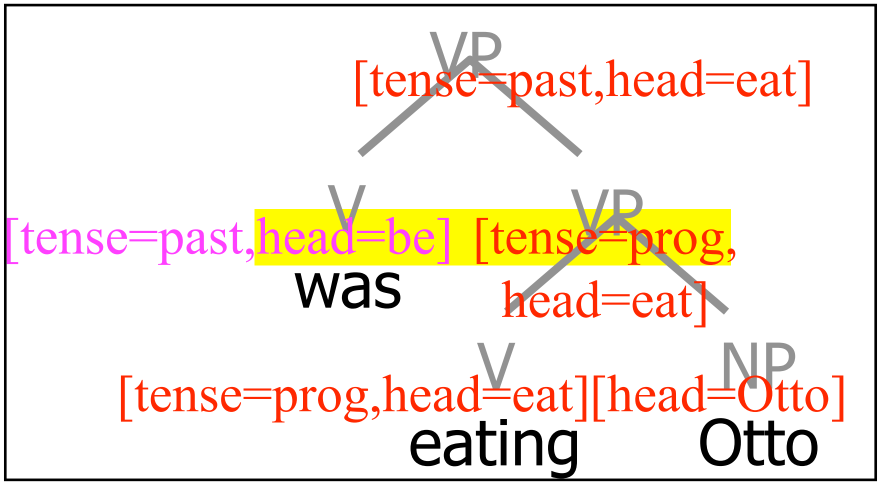


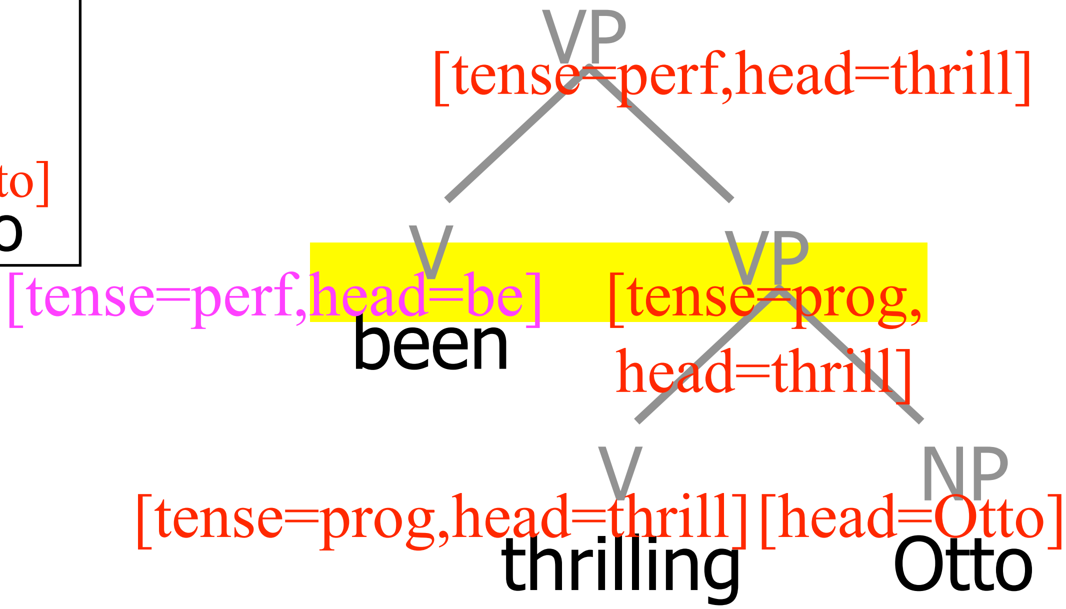
Conditional

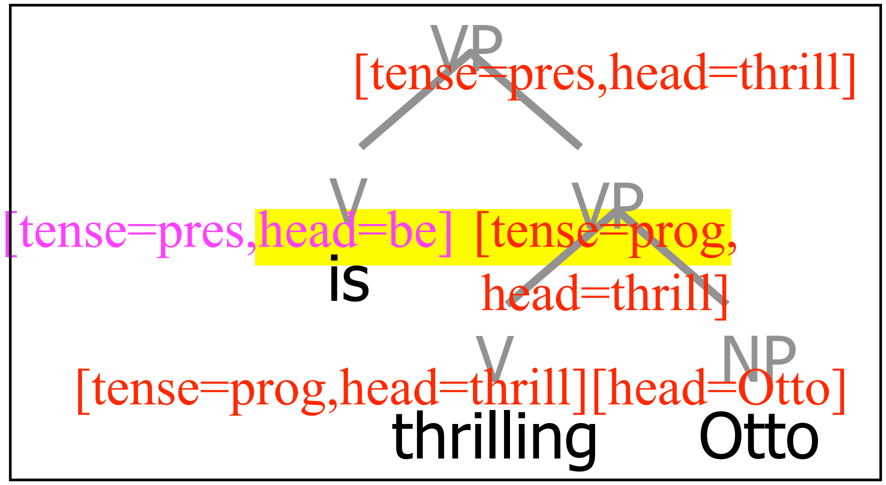
- ~~Present~~ perfect progressive tense

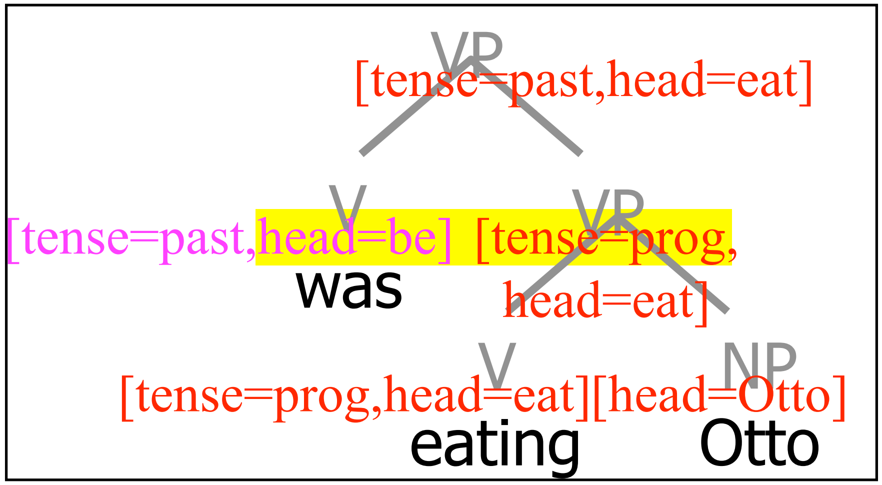
- So what pattern do **all** progressives follow?



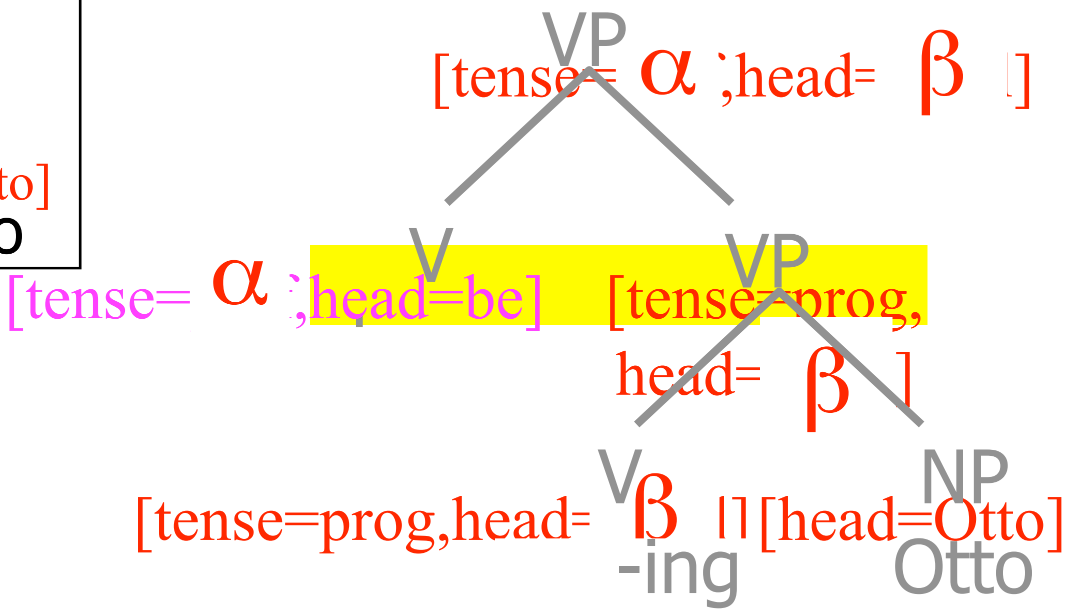








- So what pattern do **all** progressives follow?



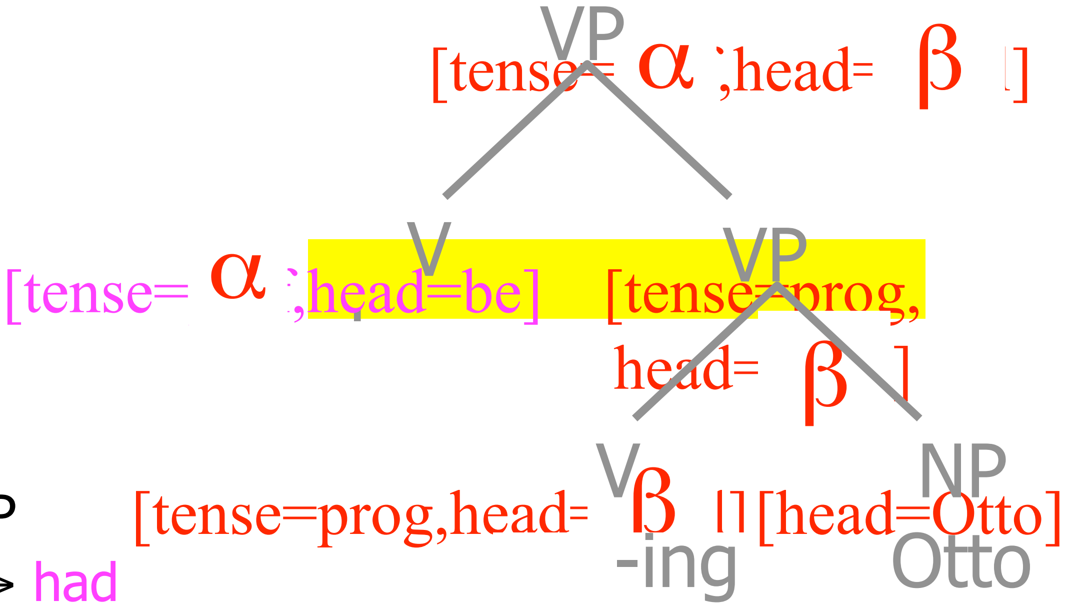
Progressive: VP[tense=α, head=β, …] → V[tense=α, stem=be …]
VP[tense=prog, head=β …]

Perfect: VP[tense=α, head=β, …] → V[tense=α, stem=have …]
VP[tense=perf, head=β …]

Future or conditional: VP[tense=α, head=β, …] → V[tense=α, stem=will …]
VP[tense=stem, head=β …]

Infinitive: VP[tense=inf, head=β, …] → to
VP[tense=stem, head=β …]

Etc.



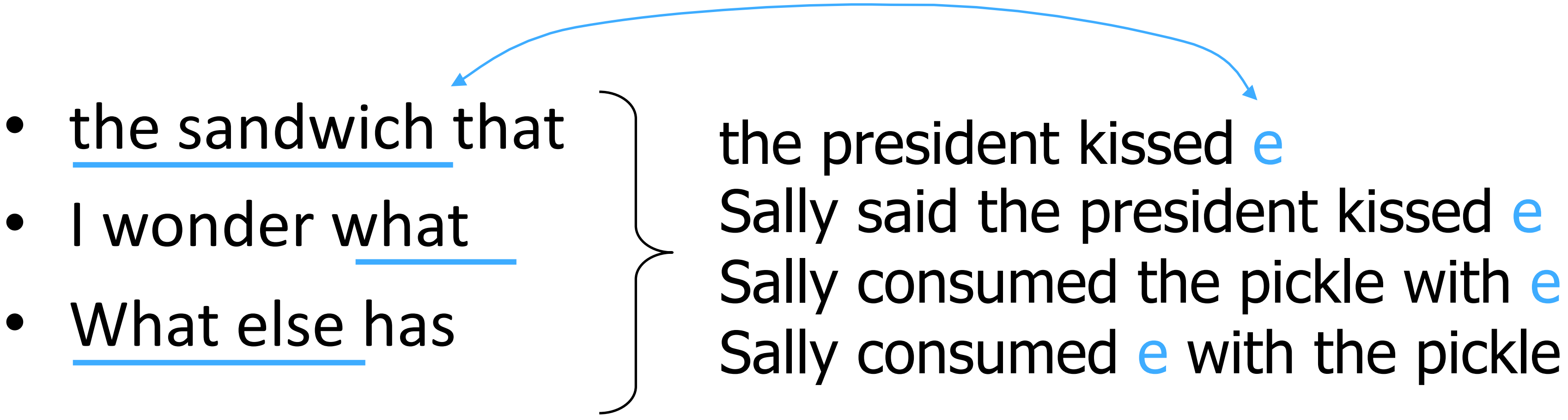
As well as the “ordinary” rules:

VP[tense=α, head=β, …]
→ V[tense=α, head=β, …] NP

V[tense=past, head=have …] → had

Gaps ("deep" grammar!)

- Pretend "kiss" is a pure transitive verb.
- Is "the president kissed" grammatical?
  - If so, what type of phrase is it?



- the sandwich that
- I wonder what
- What else has

the president kissed e
Sally said the president kissed e
Sally consumed the pickle with e
Sally consumed e with the pickle

Gaps

- **Object gaps:**
- the sandwich that
- I wonder what
- What else has

the president kissed e
Sally said the president kissed e
Sally consumed the pickle with e
Sally consumed e with the pickle

[how could you tell the difference?]

- **Subject gaps:**
- the sandwich that
- I wonder what
- What else has

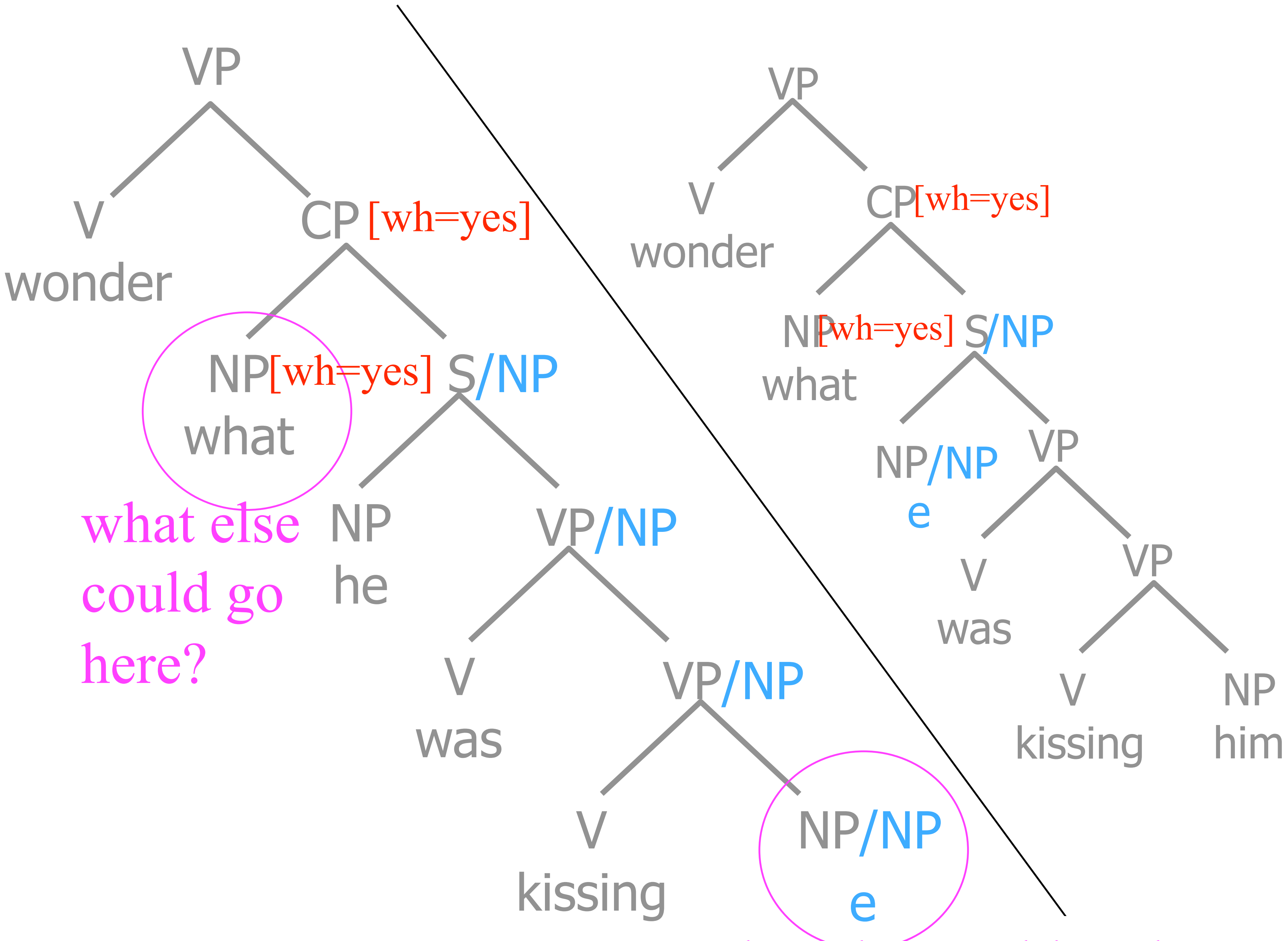
e kissed the president
Sally said e kissed the president

Gaps

- **All gaps are really the same – a missing NP:**
- the sandwich that
- I wonder what
- What else has

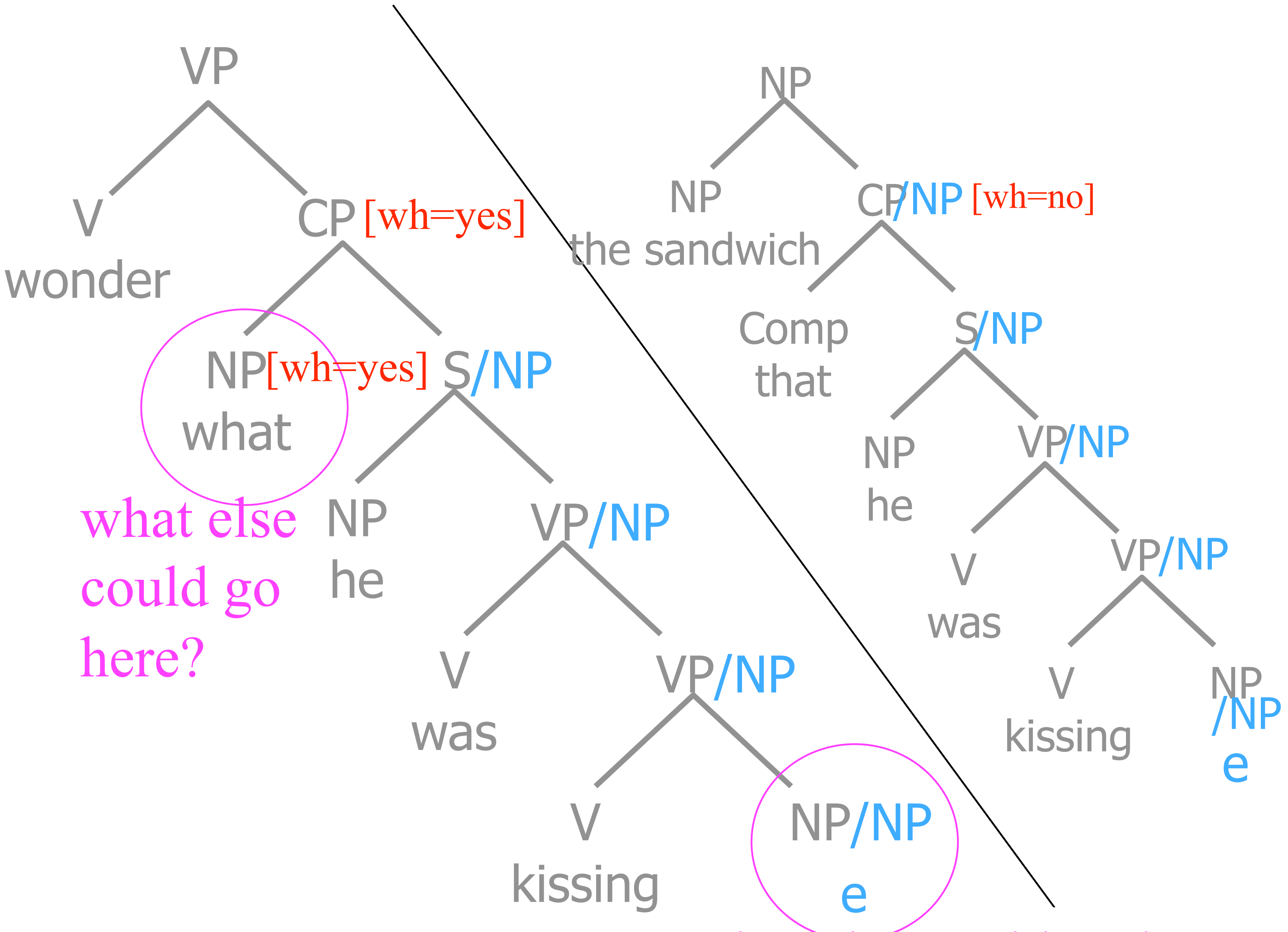
the president kissed e
Sally said the president kissed e
Sally consumed the pickle with e
Sally consumed e with the pickle
e kissed the president
Sally said e kissed the president

Phrases with missing NP:
X[missing=NP]
or just X/NP for short



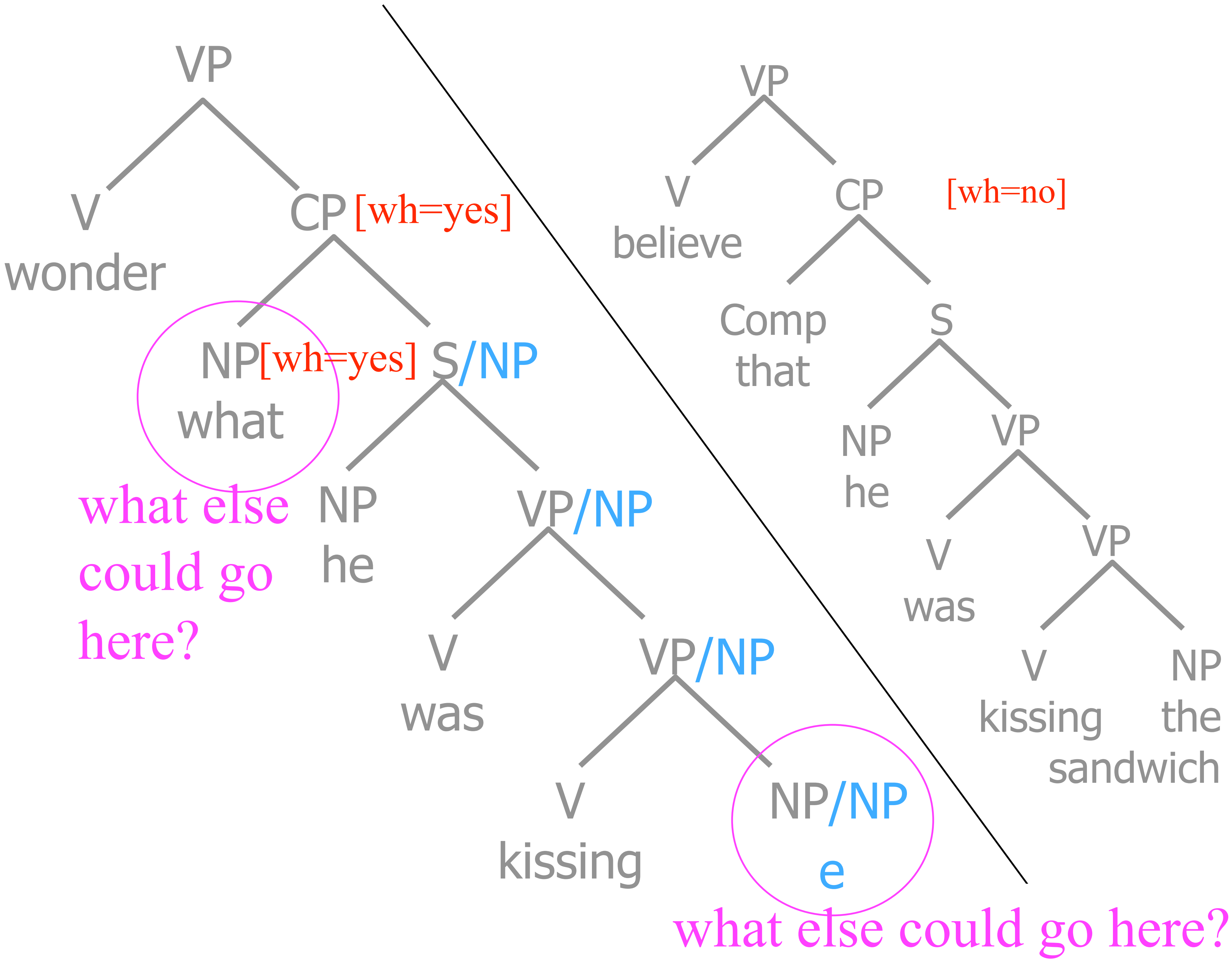
what else could go here?

what else could go here?



what else could go here?

what else could go here?

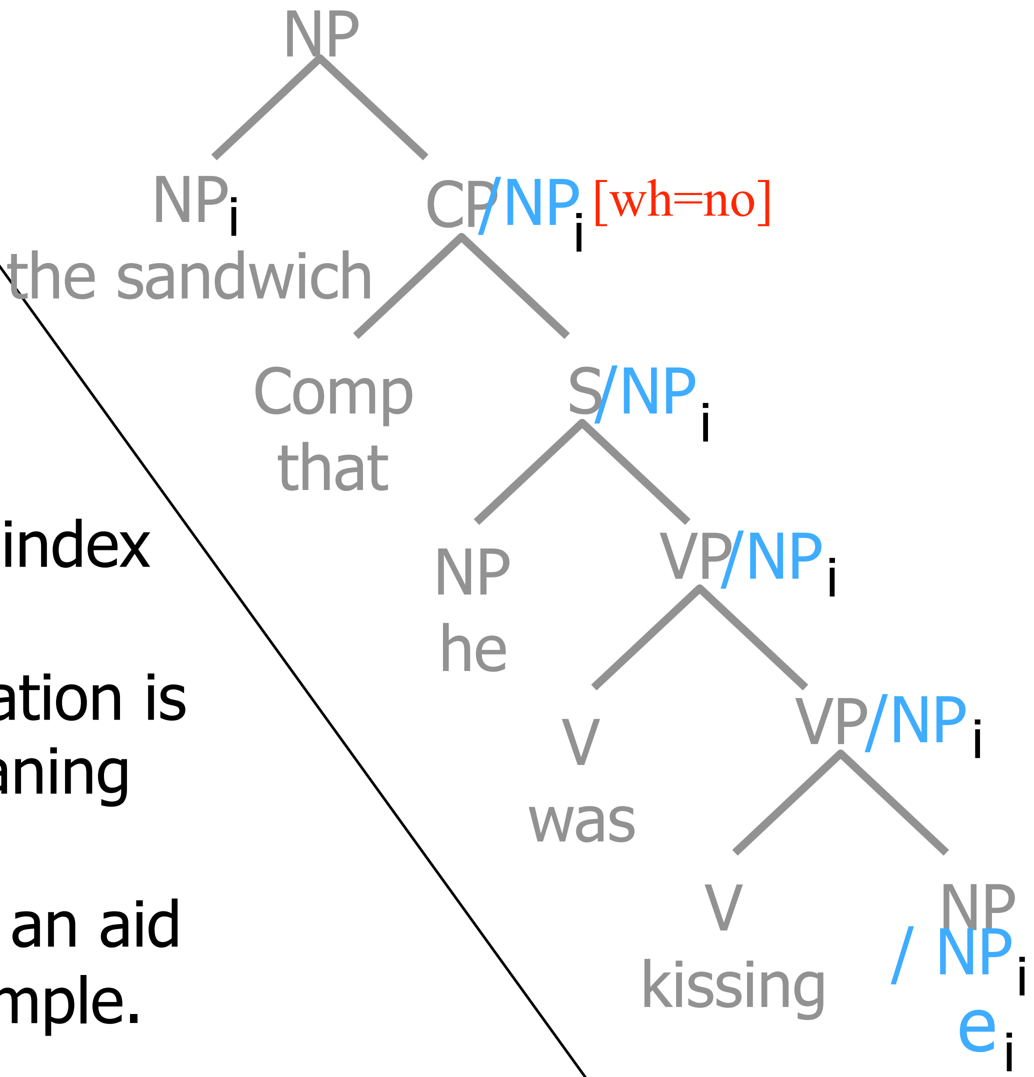


what else could go here?

what else could go here?

To indicate what fills a gap, people sometimes “coindex” the gap and its filler.

- Each phrase has a unique index such as “i”.
- In some theories, coindexation is used to help extract a meaning from the tree.
- In other theories, it is just an aid to help you follow the example.



the money$_i$ I spend e$_i$ on the happiness$_j$ I hope to buy e$_j$
which violin$_i$ is this sonata$_j$ easy to play e$_j$ on e$_i$

Parsing Tricks

Left-Corner Parsing

- Technique for 1 word of lookahead in algorithms like Earley's

- (can also do multi-word lookahead but it's harder)

Basic Earley's Algorithm

0 Papa 1	
0 ROOT . S	0 NP Papa .
0 S . NP VP	0 S NP . VP
0 NP . Det N	0 NP NP . PP
0 NP . NP PP	
0 NP . Papa	
0 Det . the	
0 Det . a	

attach

0 Papa 1	
0 ROOT . S	0 NP Papa .
0 S . NP VP	0 S NP . VP
0 NP . Det N	0 NP NP . PP
0 NP . NP PP	1 VP . V NP
0 NP . Papa	1 VP . VP PP
0 Det . the	
0 Det . a	

predict

0 Papa 1	
0 ROOT . S	0 NP Papa .
0 S . NP VP	0 S NP . VP
0 NP . Det N	0 NP NP . PP
0 NP . NP PP	1 VP . V NP
0 NP . Papa	1 VP . VP PP
0 Det . the	1 PP . P NP
0 Det . a	

predict

0 Papa 1	
0 ROOT . S	0 NP Papa .
0 S . NP VP	0 S NP . VP
0 NP . Det N	0 NP NP . PP
0 NP . NP PP	1 VP . V NP
0 NP . Papa	1 VP . VP PP
0 Det . the	1 PP . P NP
0 Det . a	1 V . ate
	1 V . drank
	1 V . snorted

predict

- .V makes us add all the verbs in the vocabulary!
- **Slow** – we’d like a shortcut.

0 Papa 1	
0 ROOT . S	0 NP Papa .
0 S . NP VP	0 S NP . VP
0 NP . Det N	0 NP NP . PP
0 NP . NP PP	1 VP . V NP
0 NP . Papa	1 VP . VP PP
0 Det . the	1 PP . P NP
0 Det . a	1 V . ate
	1 V . drank
	1 V . snorted

predict

- Every .VP adds all VP → ... rules again.
- Before adding a rule, check it's not a duplicate.
- **Slow** if there are > 700 VP → ... rules, so what will you do in Homework 3?

0 Papa 1	
0 ROOT . S	0 NP Papa .
0 S . NP VP	0 S NP . VP
0 NP . Det N	0 NP NP . PP
0 NP . NP PP	1 VP . V NP
0 NP . Papa	1 VP . VP PP
0 Det . the	1 PP . P NP
0 Det . a	1 V . ate
	1 V . drank
	1 V . snorted
	1 P . with

predict

- .P makes us add all the prepositions …

1-word lookahead would help

0 Papa 1 ate

0 ROOT . S	0 NP Papa .
0 S . NP VP	0 S NP . VP
0 NP . Det N	0 NP NP . PP
0 NP . NP PP	1 VP . V NP
0 NP . Papa	1 VP . VP PP
0 Det . the	1 PP . P NP
0 Det . a	1 V . ate
	~~1 V . drank~~
	~~1 V . snorted~~
	~~1 P . with~~

No point in adding words other than ate

1-word lookahead would help

0 Papa	1 ate
0 ROOT . S	0 NP Papa .
0 S . NP VP	0 S NP . VP
0 NP . Det N	~~0 NP NP . PP~~
0 NP . NP PP	1 VP . V NP
0 NP . Papa	1 VP . VP PP
0 Det . the	~~1 PP . P NP~~
0 Det . a	1 V . ate
	~~1 V . drank~~
	~~1 V . snorted~~
	~~1 P . with~~

In fact, no point in adding any constituent that can't start with ate

Don't bother adding PP, P, etc.

No point in adding words other than ate

With Left-Corner Filter

0 Papa	1
0 ROOT . S	0 NP Papa .
0 S . NP VP	0 S NP . VP
0 NP . Det N	~~0 NP NP . PP~~
0 NP . NP PP	
0 NP . Papa	
0 Det . the	
0 Det . a	

ate

attach

PP can’t start with ate

Pruning– now we won’t predict

1 PP . P NP

1 PP . ate

either!

Need to know that ate can’t start PP

Take closure of all categories that it does start …

0 Papa 1 ate

0 ROOT . S	0 NP Papa .
0 S . NP VP	0 S NP . VP
0 NP . Det N	~~0 NP NP . PP~~
0 NP . NP PP	1 VP . V NP
0 NP . Papa	1 VP . VP PP
0 Det . the	
0 Det . a	

predict

0 Papa 1 ate

0 Papa	1
0 ROOT . S	0 NP Papa .
0 S . NP VP	0 S NP . VP
0 NP . Det N	~~0 NP NP . PP~~
0 NP . NP PP	1 VP . V NP
0 NP . Papa	1 VP . VP PP
0 Det . the	1 V . ate
0 Det . a	~~1 V . drank~~
	~~1 V . snorted~~

predict

0 Papa	1 ate
0 ROOT . S	0 NP Papa .
0 S . NP VP	0 S NP . VP
0 NP . Det N	~~0 NP NP . PP~~
0 NP . NP PP	1 VP . V NP
0 NP . Papa	1 VP . VP PP
0 Det . the	1 V . ate
0 Det . a	~~1 V . drank~~
	~~1 V . snorted~~

predict

Merging Right-Hand Sides

- Grammar might have rules

  X → A G H P

  X → B G H P

- Could end up with both of these in chart:

  (2, X → A . G H P) in column 5

  (2, X → B . G H P) in column 5

- But these are now interchangeable: if one produces X then so will the other
- To avoid this redundancy, can always use dotted rules of this form: **X → ... G H P**

Merging Right-Hand Sides

- Similarly, grammar might have rules

  X → A G H P

  X → A G H Q

- Could end up with both of these in chart:

  (2, X → A . G H P) in column 5

  (2, X → A . G H Q) in column 5

- Not interchangeable, but we'll be processing them in parallel for a while …
- Solution: write grammar as **X → A G H (P|Q)**

Merging Right-Hand Sides

- Combining the two previous cases:

  X → A G H P

  X → A G H Q

  X → B G H P

  X → B G H Q

becomes

  X → (A | B) G H (P | Q)

- And often nice to write stuff like

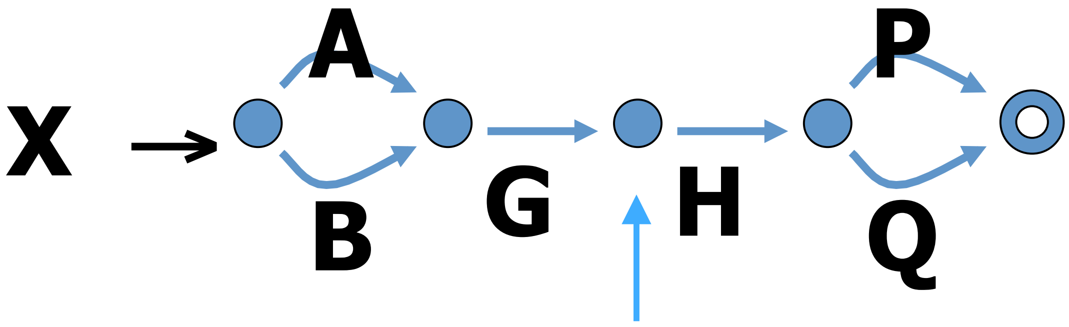
  NP → (Det | ε) Adj* N

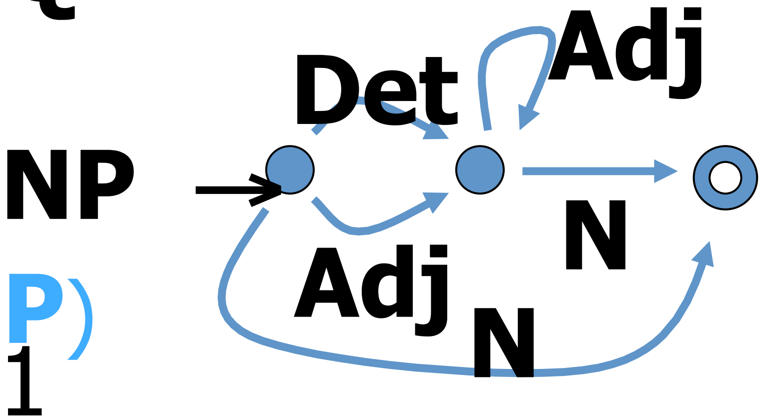
Merging Right-Hand Sides

X → (A | B) G H (P | Q)

NP → (Det | ε) Adj* N

- These are regular expressions!
- Build their minimal DFAs:





- Automaton states replace dotted rules (**X → A G . H P**)

Merging Right-Hand Sides

Indeed, *all* **NP** → rules can be unioned into a single DFA!

NP → ADJP ADJP JJ JJ NN NNS
NP → ADJP DT NN
NP → ADJP JJ NN
NP → ADJP JJ NN NNS
NP → ADJP JJ NNS
NP → ADJP NN
NP → ADJP NN NN
NP → ADJP NN NNS
NP → ADJP NNS
NP → ADJP NPR
NP → ADJP NPRS
NP → DT
NP → DT ADJP
NP → DT ADJP , JJ NN
NP → DT ADJP ADJP NN
NP → DT ADJP JJ JJ NN
NP → DT ADJP JJ NN
NP → DT ADJP JJ NN NN

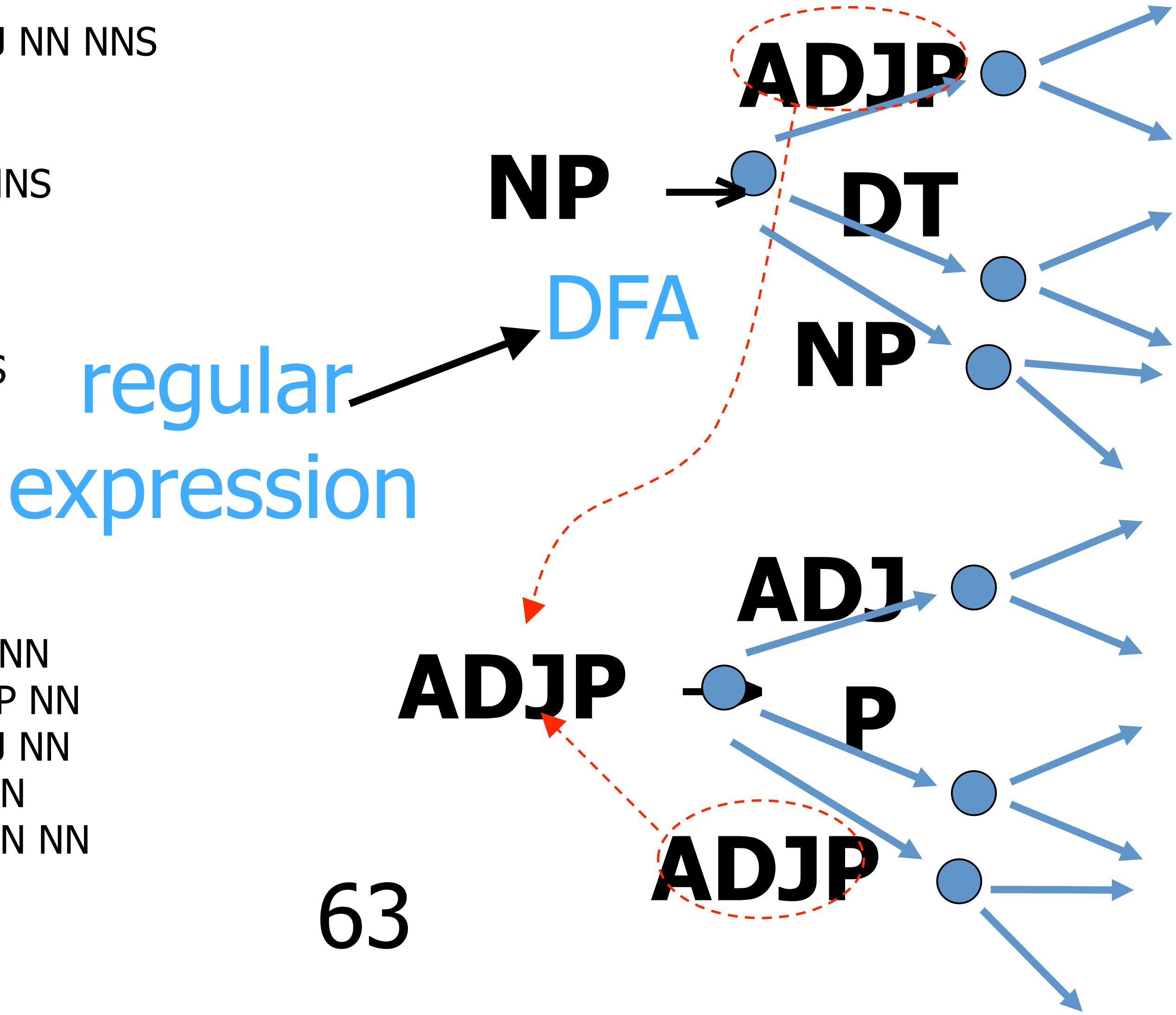
etc.

Merging Right-Hand Sides

Indeed, *all* **NP** → rules can be unioned into a single DFA!

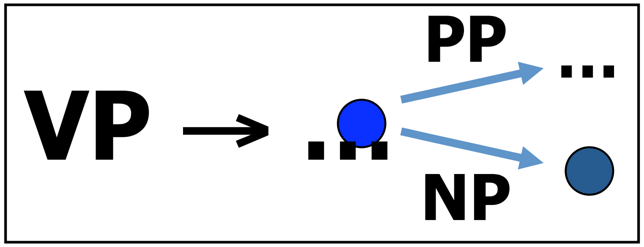
NP → ADJP ADJP JJ JJ NN NNS
| ADJP DT NN
| ADJP JJ NN
| ADJP JJ NN NNS
| ADJP JJ NNS
| ADJP NN
| ADJP NN NN
| ADJP NN NNS
| ADJP NNS
| ADJP NPR
| ADJP NPRS
| DT
| DT ADJP
| DT ADJP , JJ NN
| DT ADJP ADJP NN
| DT ADJP JJ JJ NN
| DT ADJP JJ NN
| DT ADJP JJ NN NN

etc.



Earley's Algorithm on DFAs

- What does Earley's algorithm now look like?

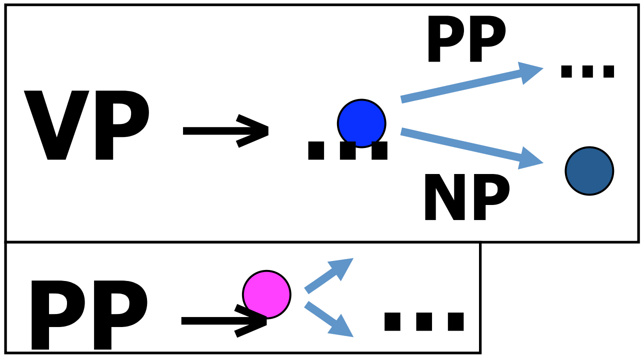


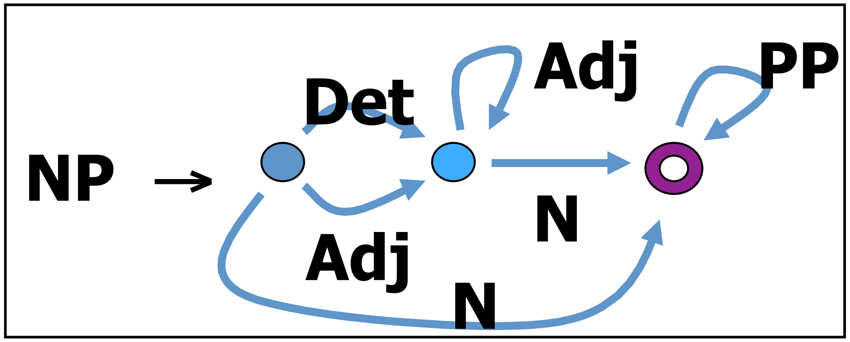
Column 4
...
(2, ●)

predict

Earley's Algorithm on DFAs

- What does Earley's algorithm now look like?

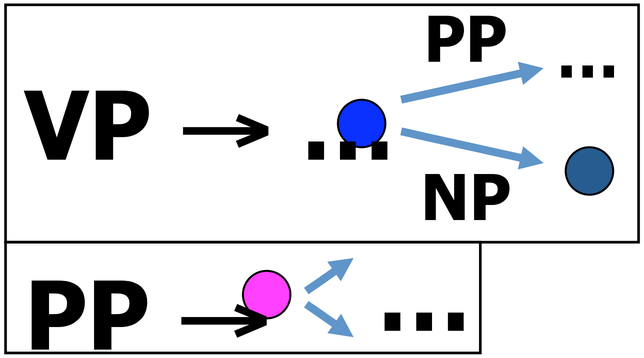


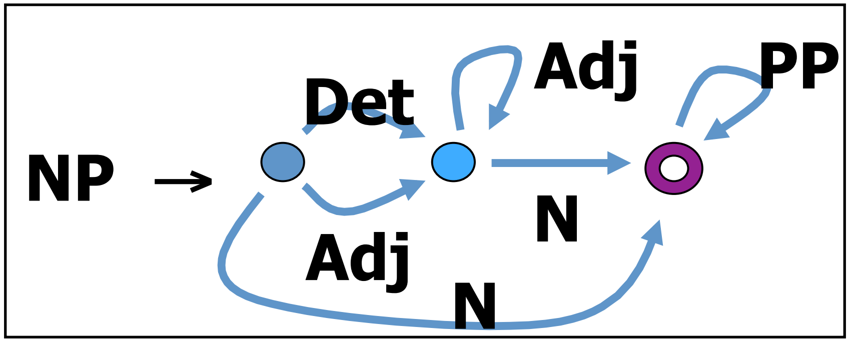


Column 4
...
(2, ●) predict
(4, ●)
(4, ●)

Earley’s Algorithm on DFAs

- What does Earley’s algorithm now look like?



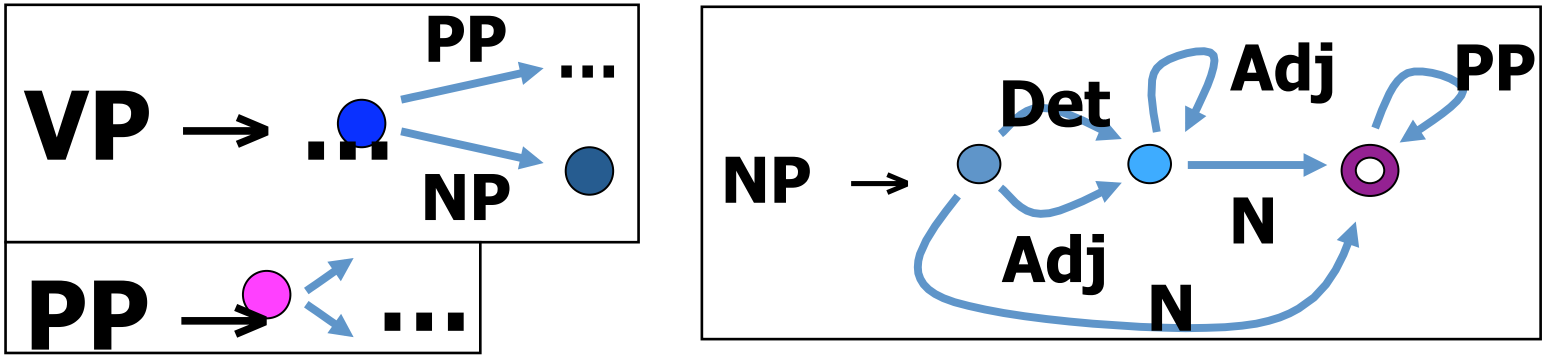


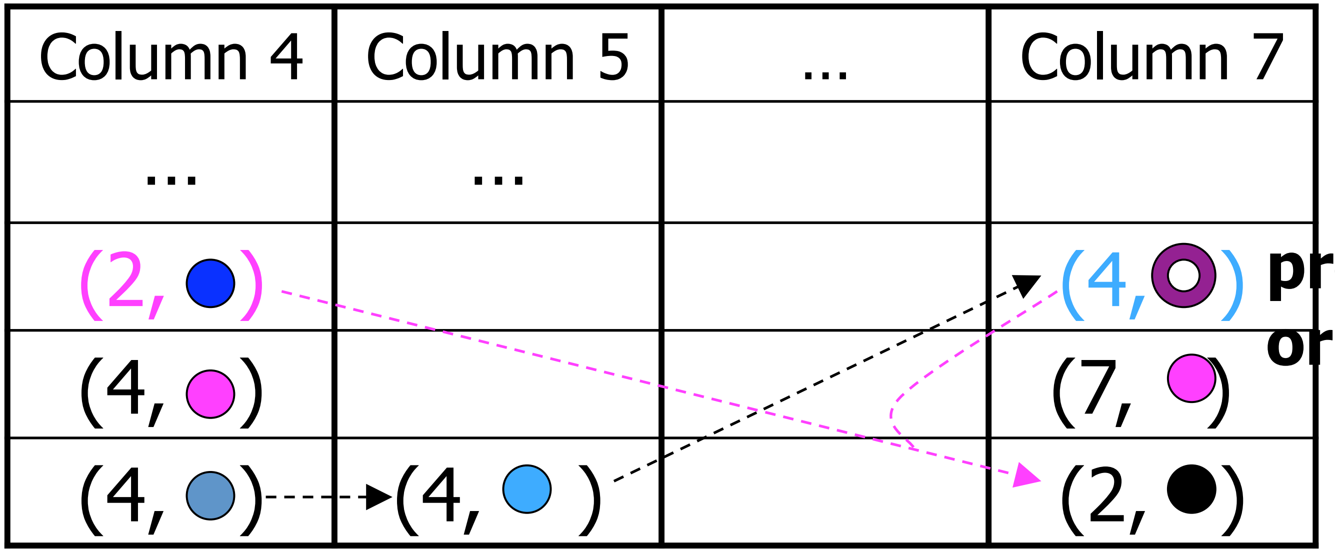
Column 4	Column 5	…	Column 7
…	…		
(2, ●)			(4, ○)
(4, ●)			
(4, ●)	(4, ●)		

predict or attach?

Earley’s Algorithm on DFAs

- What does Earley’s algorithm now look like?





Column 4	Column 5	…	Column 7
…	…		
(2, ●)			(4, ○)
(4, ●)			(7, ●)
(4, ●)	(4, ●)		(2, ●)

predict or attach?

Both!

Pruning for Speed

- Heuristically throw away constituents that probably won't make it into best complete parse.
- Use probabilities to decide which ones.
  - So probs are useful for speed as well as accuracy!
- Both safe and unsafe methods exist
  - Throw x away if $p(x) < 10^{-200}$
    (and lower this threshold if we don't get a parse)
  - Throw x away if $p(x) < 100 * p(y)$
    for some y that spans the same set of words
  - Throw x away if $p(x)*q(x)$ is small, where $q(x)$ is an estimate of probability of all rules needed to combine x with the other words in the sentence

Agenda ("Best-First") Parsing

- Explore best options first
  - Should get some good parses early on – grab one & go!
- Prioritize constits (and dotted constits)
  - Whenever we build something, give it a priority
    - How likely do we think it is to make it into the highest-prob parse?
  - usually related to log prob. of that constit
  - might also hack in the constit's context, length, etc.
  - if priorities are defined carefully, obtain an A* algorithm
- Put each constit on a priority queue (heap)
- Repeatedly pop and process best constituent.
  - CKY style: combine w/ previously popped neighbors.
  - Earley style: scan/predict/attach as usual. What else?

Preprocessing

- First “tag” the input with parts of speech:
  - Guess the correct preterminal for each word, using faster methods we’ll learn later
  - Now only allow one part of speech per word
  - This eliminates a lot of crazy constituents!
  - But if you tagged wrong you could be hosed
- Raise the stakes:
  - What if tag says not just “verb” but “transitive verb”? Or “verb with a direct object and 2 PPs attached”? (“supertagging”)
- Safer to allow a few possible tags per word, not just one …

Center-Embedding

```
if x
then
    if y
    then
        if a
        then b
        endif
    else b
    endif
else b
endif
```

STATEMENT → if EXPR then
STATEMENT endif

STATEMENT → if EXPR then STATEMENT else STATEMENT endif

But not:
STATEMENT → if EXPR then
STATEMENT

Center-Embedding

- This is the rat that ate the malt.
- This is the malt that the rat ate.

- This is the cat that bit the rat that ate the malt.
- This is the malt that the rat that the cat bit ate.

- This is the dog that chased the cat that bit the rat that ate the malt.
- This is the malt that [the rat that [the cat that [the dog chased] bit] ate].

More Center-Embedding

[What did you disguise
[those handshakes that you greeted
[the people we bought
[the bench
[Billy was read to]
on]
with]
with]
for]?

[Which mantelpiece did you put
[the idol I sacrificed
[the fellow we sold
[the bridge you threw
[the bench
[Billy was read to]
on]
off]
to]
to]
on]?

Take that,
English teachers!

Center Recursion vs. Tail Recursion

[What did you disguise
[those handshakes that you greeted
[the people we bought
[the bench
[Billy was read to]
on]
with]
with]
for]?

[For what did you disguise
[those handshakes with which you greeted
[the people with which we bought
[the bench on which
[Billy was read to]?

"pied piping" –
NP moves leftward,
preposition follows along

Disallow Center-Embedding?

- Center-embedding seems to be in the grammar, but people have trouble processing more than 1 level of it.
- You can limit # levels of center-embedding via features: e.g., S[S_DEPTH=n+1] → A S[S_DEPTH=n] B

- If a CFG limits # levels of embedding, then it can be compiled into a finite-state machine – we don't need a stack at all!
  - Finite-state recognizers run in linear time.
  - However, it's tricky to turn them into parsers for the original CFG from which the recognizer was compiled.

Parsing Algs for non-CFG

- If you're going to make up a new kind of grammar, you should also describe how to parse it.

- Such algorithms exist!
- For example, there are parsing algorithms for TAG (where larger tree fragments can be combined by substitution & adjunction)