Sharpening the empirical claims of generative syntax through formalization

Tim Hunter

University of Minnesota, Twin Cities

ESSLLI, August 2015

Part 2

Minimalist Grammars

Outline

Outline

Wait a minute!

"I thought the whole point was deciding between candidate sets of primitive derivational operations! Isn't it begging the question to set everything in stone at the beginning like this?"

Wait a minute!

> “I thought the whole point was deciding between candidate sets of primitive derivational operations! Isn’t it begging the question to set everything in stone at the beginning like this?”

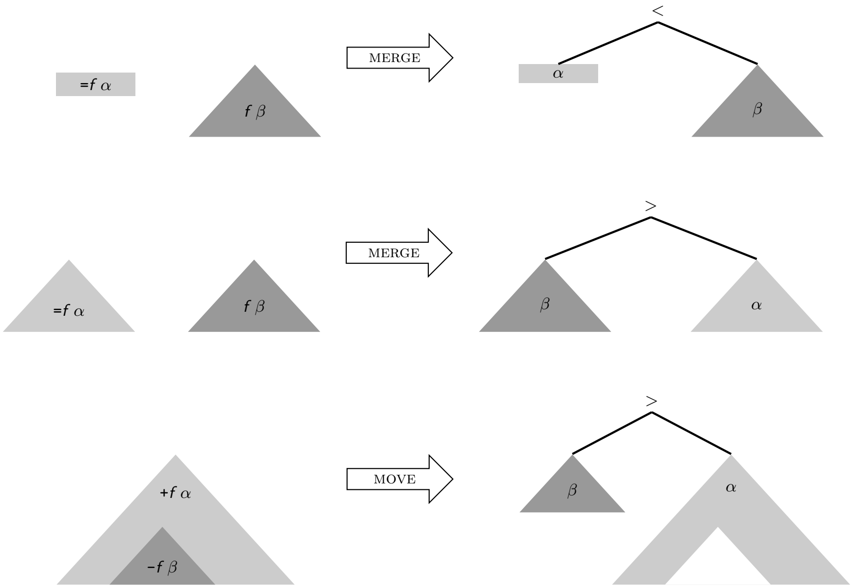
- We’re not setting this in stone — we will look at alternatives.
- But we need a concrete starting point so that we can make the differences concrete.
- What’s coming up is meant as a relatively neutral/“mainstream” starting point.

Minimalist Grammars

Defining a grammar in the MG formalism is defining a set *Lex* of lexical items

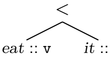
- A lexical item is a string with a sequence of features.
  e.g. *like* :: `=d =d v`, *mary* :: `d`, *who* :: `d -wh`
- Generates the closure of the $Lex \subset Expr$ under two derivational operations:
  - MERGE : $Expr \times Expr \xrightarrow{\text{partial}} Expr$
  - MOVE : $Expr \xrightarrow{\text{partial}} Expr$
- Each feature encodes a requirement that must be met by applying a particular derivational operation.
  - MERGE checks `=f` and `f`
  - MOVE checks `+f` and `-f`
- A derived expression is complete when it has only a single feature remaining unchecked.

Merge and move

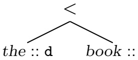


Examples

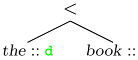
$\textsc{merge}\,(\textit{eat} :: \texttt{=d v}, \textit{it} :: \texttt{d}) =$



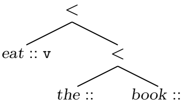
$\textsc{merge}\,(\textit{the} :: \texttt{=n d}, \textit{book} :: \texttt{n}) =$



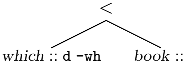
$\textsc{merge}\,(\textit{eat} :: \texttt{=d v},$



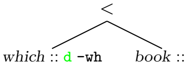
$) =$



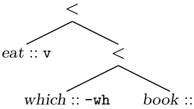
$\textsc{merge}\,(\textit{which} :: \texttt{=n d -wh}, \textit{book} :: \texttt{n}) =$



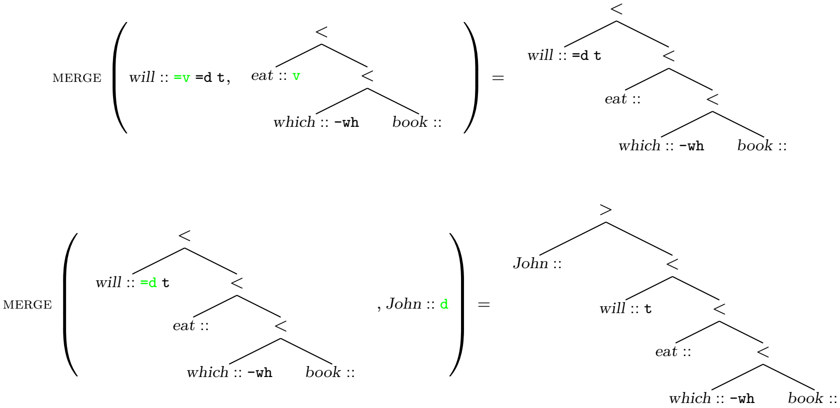
$\textsc{merge}\,(\textit{eat} :: \texttt{=d v},$



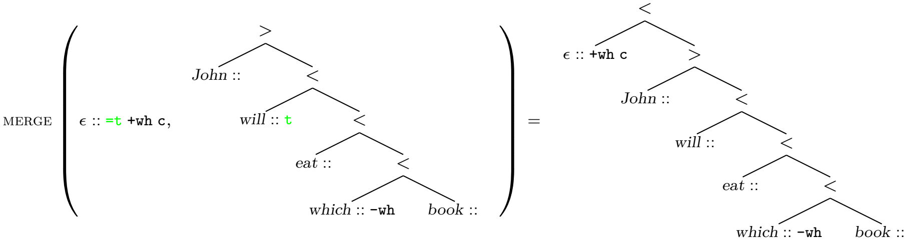
$) =$

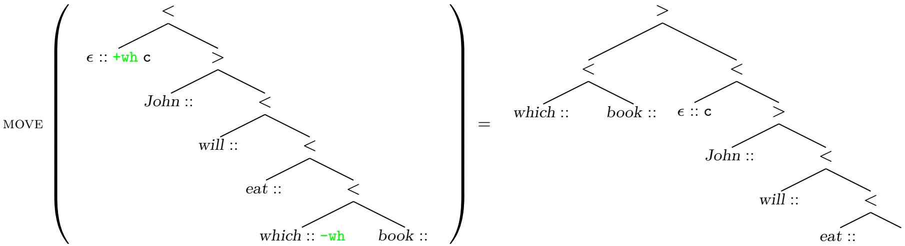


Examples

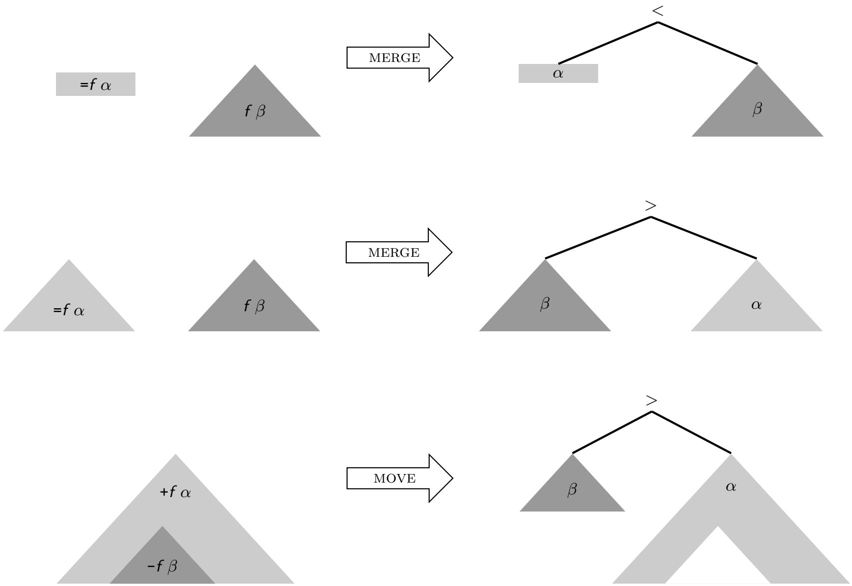

MERGE
will :: =v =d t,
<
eat :: v
<
which :: -wh
book ::
=
<
will :: =d t
<
eat ::
<
which :: -wh
book ::
MERGE
<
will :: =d t
<
eat ::
<
which :: -wh
book ::
, John :: d
=
>
John ::
<
will :: t
<
eat ::
<
which :: -wh
book ::

Examples


MERGE
ϵ :: =t +wh c,
>
John ::
<
will :: t
<
eat ::
<
which :: -wh
book ::
=
<
ϵ :: +wh c
>
John ::
<
will ::
<
eat ::
<
which :: -wh
book ::


MOVE
<
ϵ :: +wh c
>
John ::
<
will ::
<
eat ::
<
which :: -wh
book ::
=
>
<
which ::
book ::
<
ϵ :: c
>
John ::
<
will ::
<
eat ::

Merge and move



Definitions

$$\textsc{merge}\Big(e_1[\text{=}f\ \alpha], e_2[f\ \beta]\Big) = \begin{cases} [_< \ e_1[\alpha]\ e_2[\beta]] & \text{if } e_1[\text{=}f\ \alpha] \in Lex \\ [_> \ e_2[\beta]\ e_1[\alpha]] & \text{otherwise} \end{cases}$$

$$\textsc{move}\Big(e_1[\text{+}f\ \alpha]\Big) = [_> \ e_2[\beta]\ e_1'[\alpha]]$$

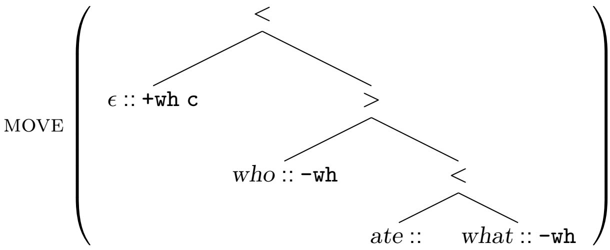
where $e_2[\text{-}f\ \beta]$ is a unique subtree of $e_1[\text{+}f\ \alpha]$

and e_1' is like e_1 but with $e_2[\text{-}f\ \beta]$ replaced by an empty leaf node

Shortest Move Constraint

How do we know which subtree should be displaced when we apply MOVE?

By stipulation, there can only ever be one candidate. This is the Shortest Move Constraint (SMC).



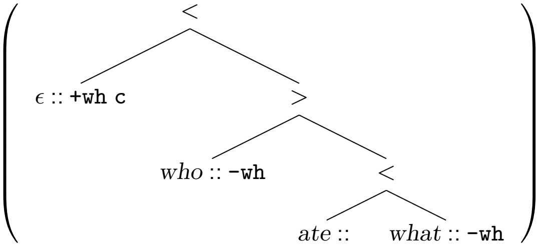
is **undefined**

Shortest Move Constraint

How do we know which subtree should be displaced when we apply MOVE?

By stipulation, there can only ever be one candidate. This is the Shortest Move Constraint (SMC).

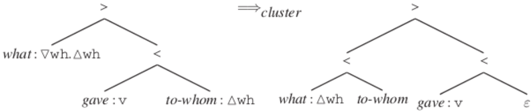
MOVE



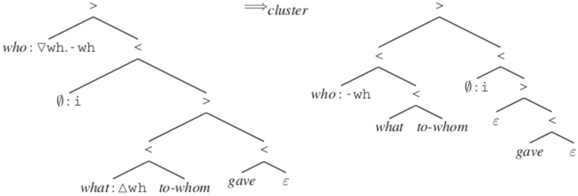
is **undefined**

Q: Multiple wh-movement?
A: Clustering!

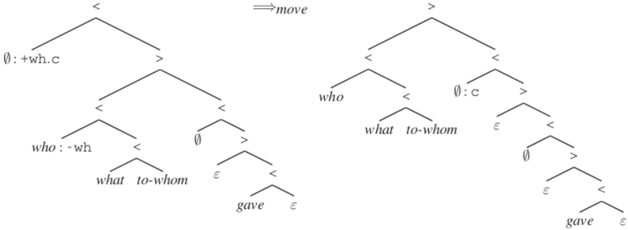
(7) a.



b.



c.



(Gärtner and Michaelis 2010)

Notation

=d v or =dp vp?

Notation

`=d v` or `=dp vp`?

Categorial grammar:

- Primitive symbols for "complete" things, e.g. S, NP
- Derived symbols for "incomplete" things, e.g. S\NP
- Lexical category can specify "what's missing"

Notation

`=d v` or `=dp vp`?

Categorial grammar:

- Primitive symbols for “complete” things, e.g. S, NP
- Derived symbols for “incomplete” things, e.g. S\NP
- Lexical category can specify “what’s missing”

Traditional X-bar theory:

- Primitive symbols for “incomplete” things, e.g. V, T
- Derived symbols for “complete” things, e.g. VP, TP ($= V'', T''$)
- Separate subcategorization info specifies “what’s missing”

Notation

`=d v` or `=dp vp`?

Categorial grammar:

- Primitive symbols for "complete" things, e.g. S, NP
- Derived symbols for "incomplete" things, e.g. S\NP
- Lexical category can specify "what's missing"

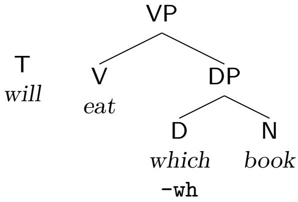
Traditional X-bar theory:

- Primitive symbols for "incomplete" things, e.g. V, T
- Derived symbols for "complete" things, e.g. VP, TP ($=$ V$''$, T$''$)
- Separate subcategorization info specifies "what's missing"

MGs:

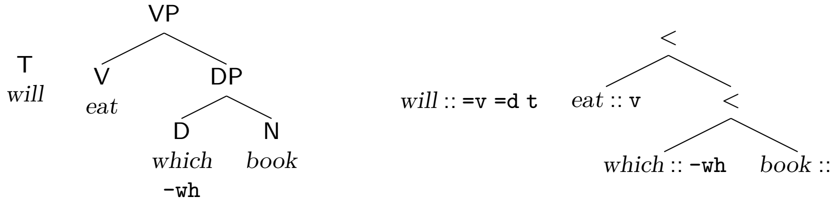
- Primitive symbols for "complete" things, like CG
- So `t` means "a complete projection of T", not "a T head"

Notation comparison



	Conventional notation
‘eat which book’ is a VP	VP label on root
‘which book’ must move	`-wh` on ‘which’
‘will’ combines with a VP	implicit

Notation comparison



	Conventional notation	MG notation
'eat which book' is a VP	VP label on root	`v` on 'eat'
'which book' must move	`-wh` on 'which'	`-wh` on 'which'
'will' combines with a VP	implicit	`=v` on 'will'

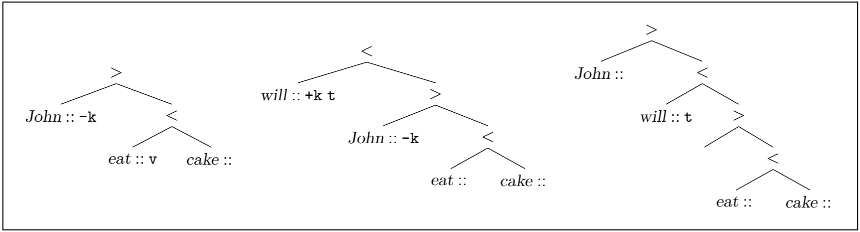
Outline

A Minimalist Grammar

cake :: `d`	*what* :: `d -wh`
John :: `d -k`	*who* :: `d -k -wh`
eat :: `=d =d v`	ϵ :: `=t +wh c`
will :: `=v +k t`	ϵ :: `=t c`

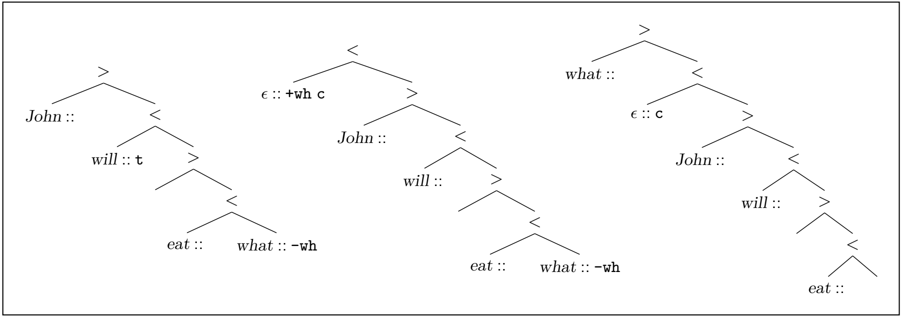
A Minimalist Grammar

cake :: `d`
John :: `d -k`
eat :: `=d =d v`
will :: `=v +k t`
what :: `d -wh`
who :: `d -k -wh`
ϵ :: `=t +wh c`
ϵ :: `=t c`



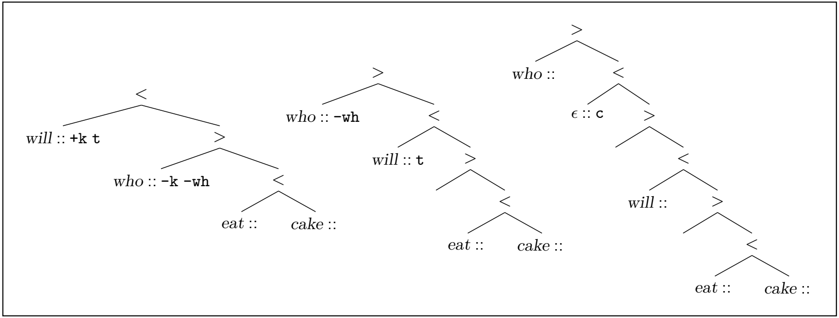
A Minimalist Grammar

cake :: d	*what* :: d -wh
John :: d -k	*who* :: d -k -wh
eat :: =d =d v	ϵ :: =t +wh c
will :: =v +k t	ϵ :: =t c



A Minimalist Grammar

$\mathit{cake} :: \mathtt{d}$	$\mathit{what} :: \mathtt{d}\ \mathtt{-wh}$
$\mathit{John} :: \mathtt{d}\ \mathtt{-k}$	$\mathit{who} :: \mathtt{d}\ \mathtt{-k}\ \mathtt{-wh}$
$\mathit{eat} :: \mathtt{=d}\ \mathtt{=d}\ \mathtt{v}$	$\epsilon :: \mathtt{=t}\ \mathtt{+wh}\ \mathtt{c}$
$\mathit{will} :: \mathtt{=v}\ \mathtt{+k}\ \mathtt{t}$	$\epsilon :: \mathtt{=t}\ \mathtt{c}$

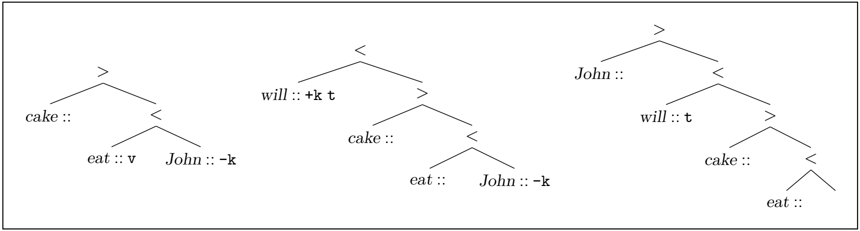


A Minimalist Grammar . . . which overgenerates

cake :: `d`	*what* :: `d -wh`
John :: `d -k`	*who* :: `d -k -wh`
eat :: `=d =d v`	ϵ :: `=t +wh c`
will :: `=v +k t`	ϵ :: `=t c`

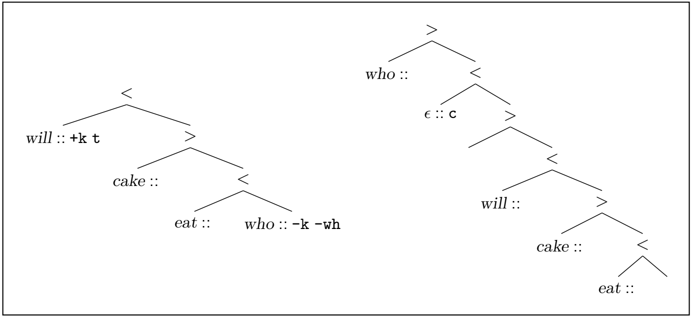
A Minimalist Grammar . . . which overgenerates

cake :: `d`	*what* :: `d -wh`
John :: `d -k`	*who* :: `d -k -wh`
eat :: `=d =d v`	ϵ :: `=t +wh c`
will :: `=v +k t`	ϵ :: `=t c`



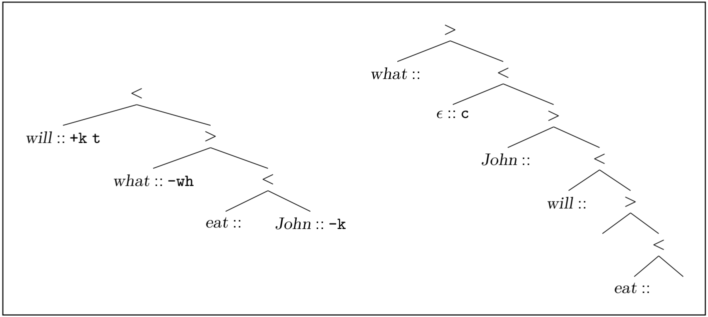
A Minimalist Grammar . . . which overgenerates

cake :: d	*what* :: d -wh
John :: d -k	*who* :: d -k -wh
eat :: =d =d v	ϵ :: =t +wh c
will :: =v +k t	ϵ :: =t c



A Minimalist Grammar . . . which overgenerates

$cake$ :: d	$what$ :: d -wh
$John$ :: d -k	who :: d -k -wh
eat :: =d =d v	ϵ :: =t +wh c
$will$ :: =v +k t	ϵ :: =t c



A Minimalist Grammar . . . which overgenerates

cake :: `d`	*what* :: `d -wh`
John :: `d -k`	*who* :: `d -k -wh`
eat :: `=d =d v`	ϵ :: `=t +wh c`
will :: `=v +k t`	ϵ :: `=t c`

John will eat cake	*John will cake eat*
what John will eat	*what John will eat*
who will eat cake	*who will cake eat*

A Minimalist Grammar . . . which overgenerates

cake :: `d`	*what* :: `d -wh`
John :: `d -k`	*who* :: `d -k -wh`
eat :: `=d =d v`	ϵ :: `=t +wh c`
will :: `=v +k t`	ϵ :: `=t c`

John will eat cake	*John will cake eat*
what John will eat	*what John will eat*
who will eat cake	*who will cake eat*

S	$\rightarrow$	NP VP	VP	$\rightarrow$	V NP
NP	$\rightarrow$	*John*	VP	$\rightarrow$	*runs*
NP	$\rightarrow$	*Mary*	VP	$\rightarrow$	*walks*
			V	$\rightarrow$	*loves*

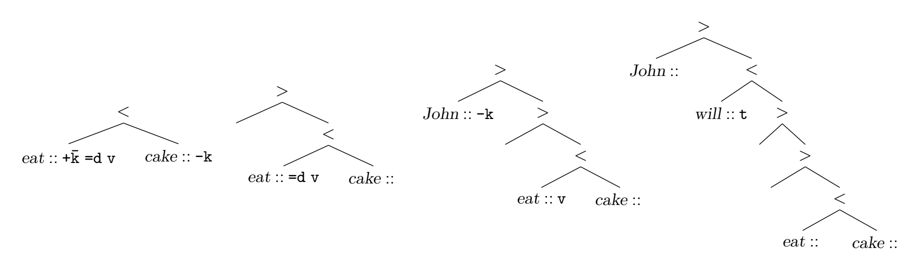
John runs	*Mary runs*
John walks	*Mary walks*
John loves John	*Mary loves John*
John loves Mary	*Mary loves Mary*

First solution: covert movement/agree

cake :: `d -k`	*what* :: `d -k -wh`
John :: `d -k`	*who* :: `d -k -wh`
eat :: `=d +k̄ =d v`	ϵ :: `=t +wh c`
will :: `=v +k t`	ϵ :: `=t c`

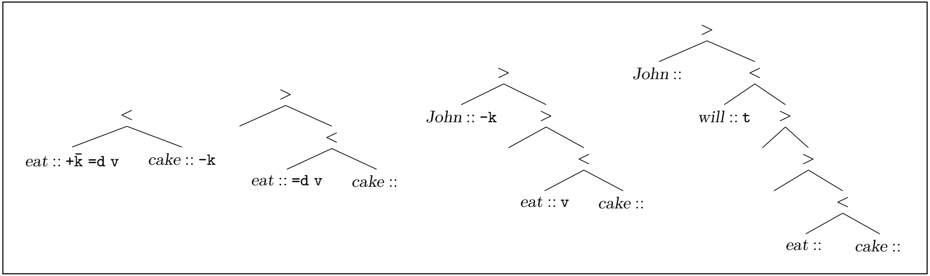
First solution: covert movement/agree

cake :: d -k	*what* :: d -k -wh
John :: d -k	*who* :: d -k -wh
eat :: =d +$\bar{\text{k}}$ =d v	ϵ :: =t +wh c
will :: =v +k t	ϵ :: =t c



First solution: covert movement/agree

cake :: d -k	*what* :: d -k -wh
John :: d -k	*who* :: d -k -wh
eat :: =d +$\bar{\text{k}}$ =d v	ϵ :: =t +wh c
will :: =v +k t	ϵ :: =t c



Note order of features on *eat*!

Second solution

Separate `d` into `subj` and `obj`

cake :: `obj`	*what* :: `obj -wh`
John :: `subj -k`	*who* :: `subj -k -wh`
eat :: `=obj =subj v`	ϵ :: `=t +wh c`
will :: `=v +k t`	ϵ :: `=t c`

Problem "solved":

John will eat cake
what John will eat
who will eat cake

Outline

Adding embedded clauses

cake :: `obj`	*what* :: `obj -wh`	*think* :: `=c =subj v`
John :: `subj -k`	*who* :: `subj -k -wh`	*ask* :: `=q =subj v`
eat :: `=obj =subj v`	ϵ :: `=t +wh q`	*Mary* :: `subj -k`
will :: `=v +k t`	ϵ :: `=t c`	

Adding embedded clauses

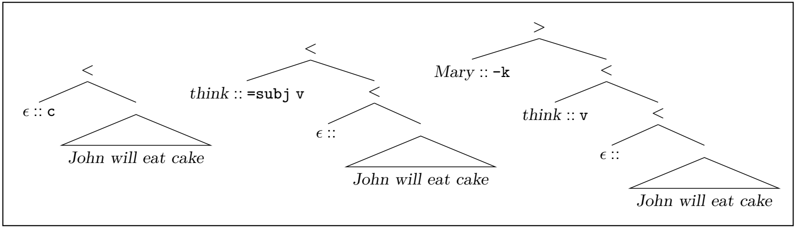
cake :: `obj`	*what* :: `obj -wh`	*think* :: `=c =subj v`
John :: `subj -k`	*who* :: `subj -k -wh`	*ask* :: `=q =subj v`
eat :: `=obj =subj v`	ϵ :: `=t +wh q`	*Mary* :: `subj -k`
will :: `=v +k t`	ϵ :: `=t c`	

John will eat cake	*Mary will think John will eat cake*	...
what John will eat	*what Mary will think John will eat*	...
who will eat cake	*who Mary will think will eat cake*	...

Adding embedded clauses

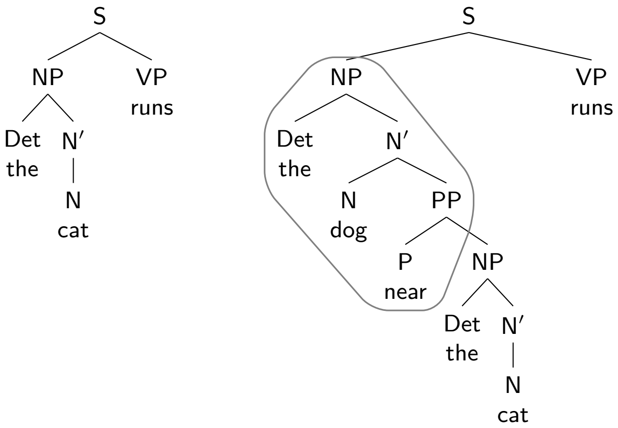
cake :: obj	*what* :: obj -wh	*think* :: =c =subj v
John :: subj -k	*who* :: subj -k -wh	*ask* :: =q =subj v
eat :: =obj =subj v	ϵ :: =t +wh q	*Mary* :: subj -k
will :: =v +k t	ϵ :: =t c	

John will eat cake	*Mary will think John will eat cake*	. . .
what John will eat	*what Mary will think John will eat*	. . .
who will eat cake	*who Mary will think will eat cake*	. . .



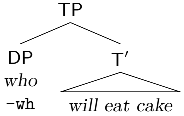
Reminder: "Loops" in a CFG

S	$\rightarrow$	NP VP	VP	$\rightarrow$	*runs*
NP	$\rightarrow$	Det N′	Det	$\rightarrow$	*the*
N′	$\rightarrow$	N	N	$\rightarrow$	*dog*
N′	$\rightarrow$	N PP	N	$\rightarrow$	*cat*
PP	$\rightarrow$	P NP	P	$\rightarrow$	*near*

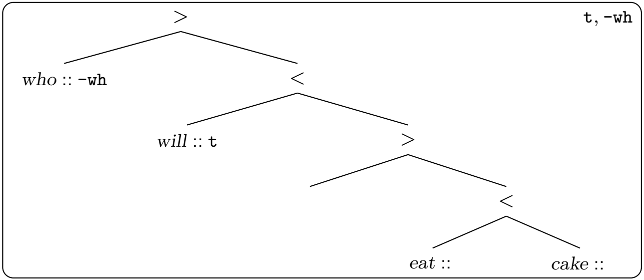


Which extensions create “loops”?

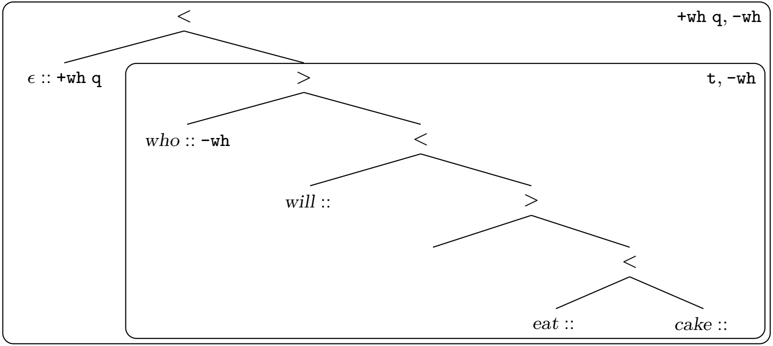
Starting point:



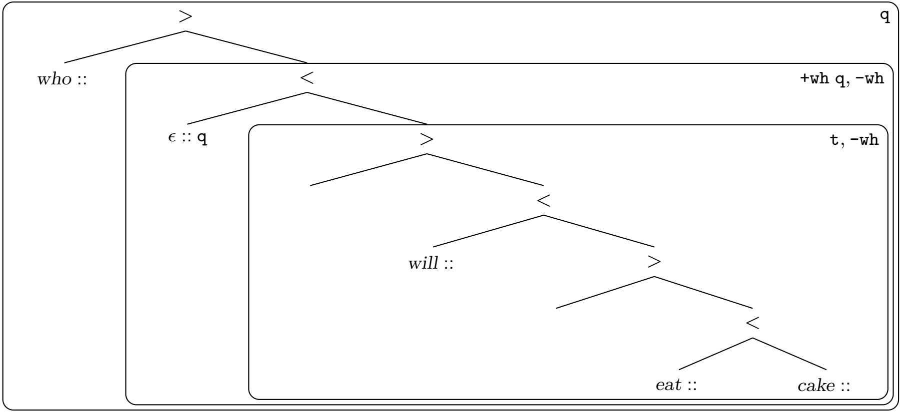
A simple, non-looping completion



A simple, non-looping completion

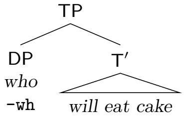


A simple, non-looping completion



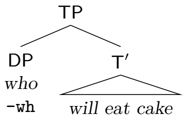
Which extensions create "loops"?

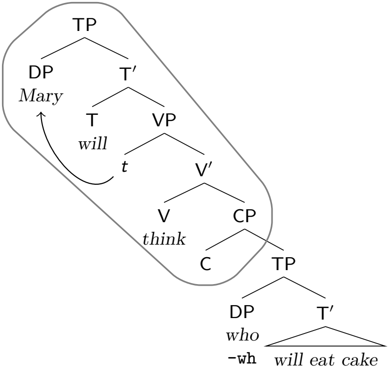
Starting point:



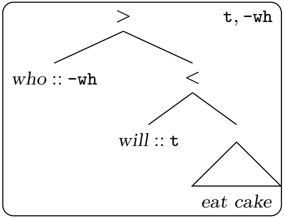
Which extensions create "loops"?

Starting point:

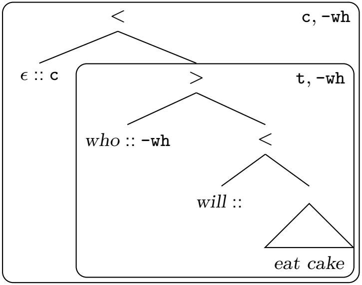




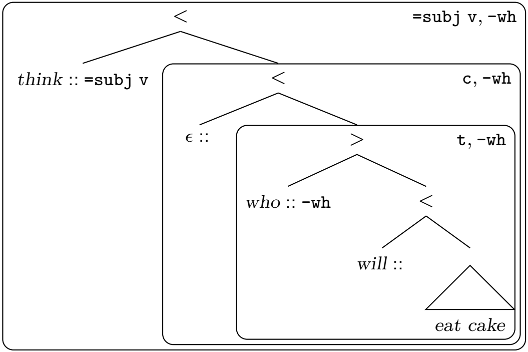
Extending with *Mary will think . . .*



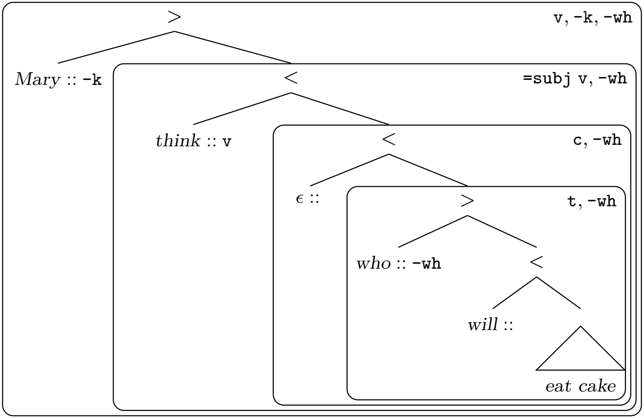
Extending with *Mary will think* . . .



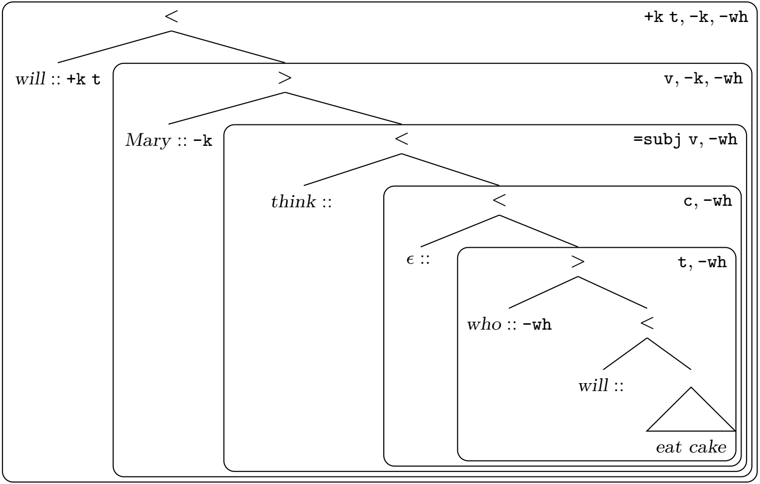
Extending with *Mary will think …*



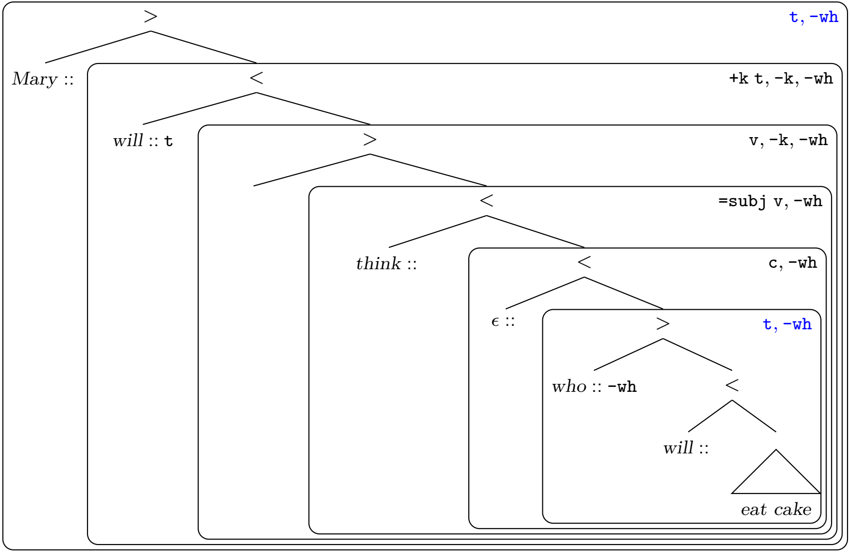
Extending with *Mary will think . . .*



Extending with *Mary will think* . . .

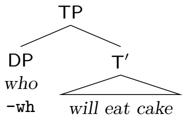


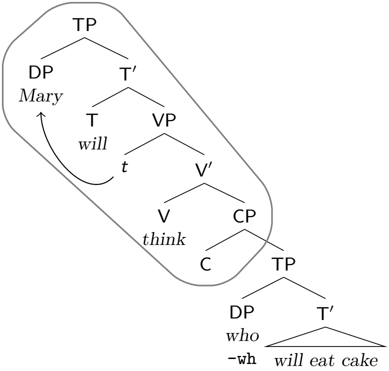
Extending with *Mary will think …*



Which extensions create "loops"?

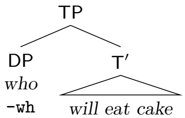
Starting point:

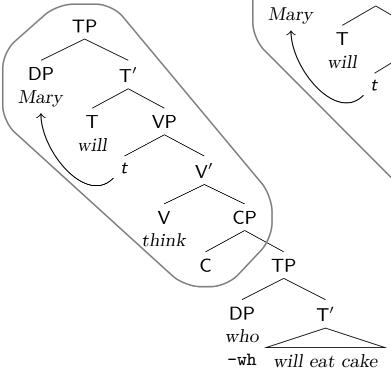


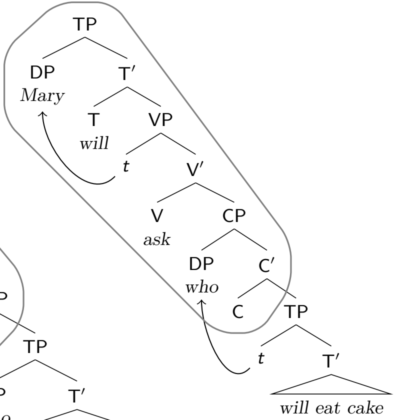


Which extensions create "loops"?

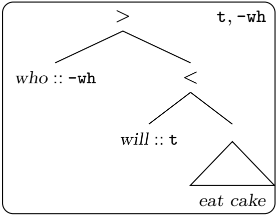
Starting point:



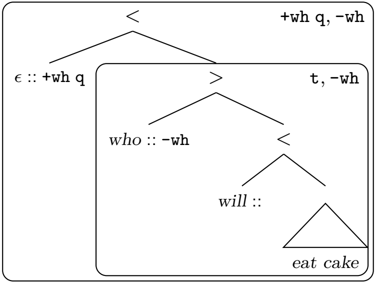




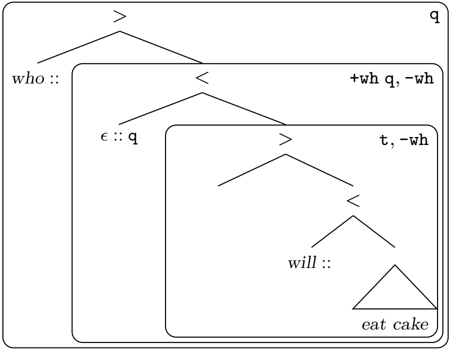
Extending with *Mary will ask …*



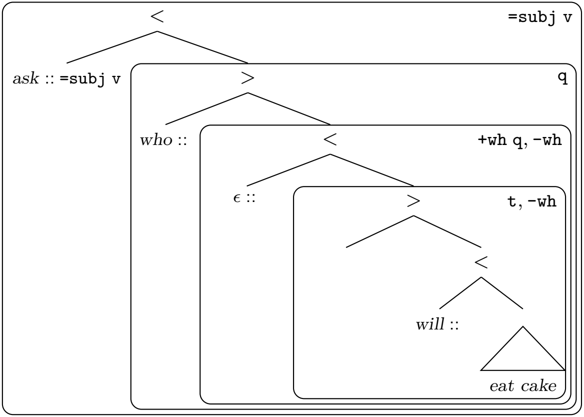
Extending with *Mary will ask . . .*



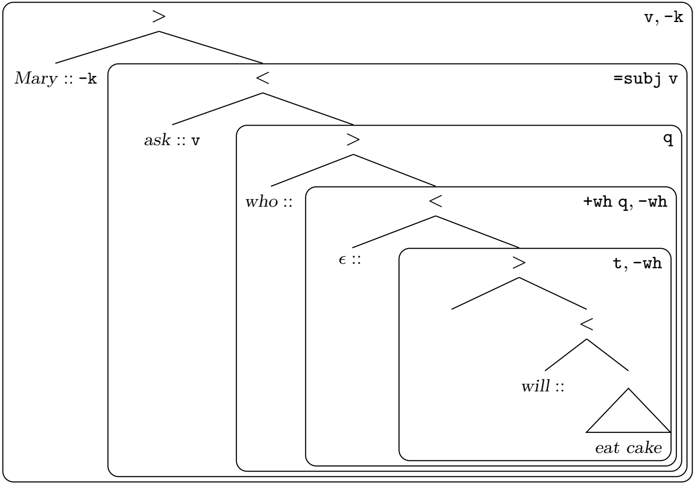
Extending with *Mary will ask …*



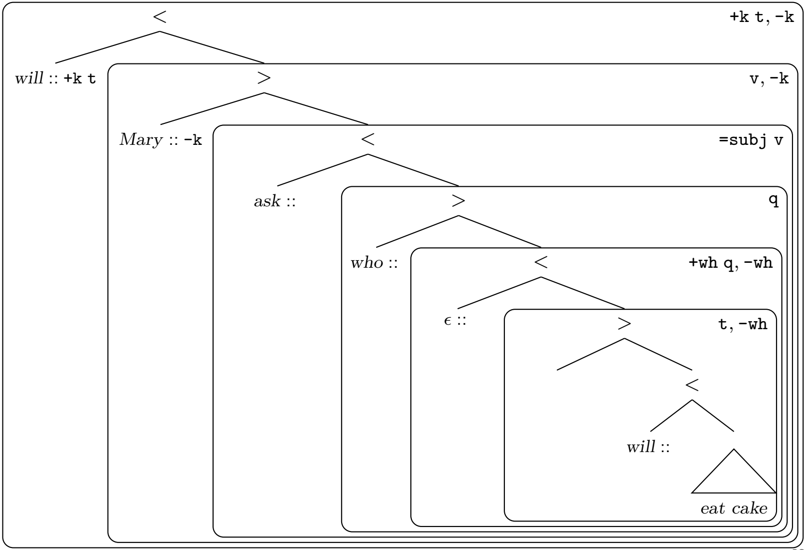
Extending with *Mary will ask* . . .



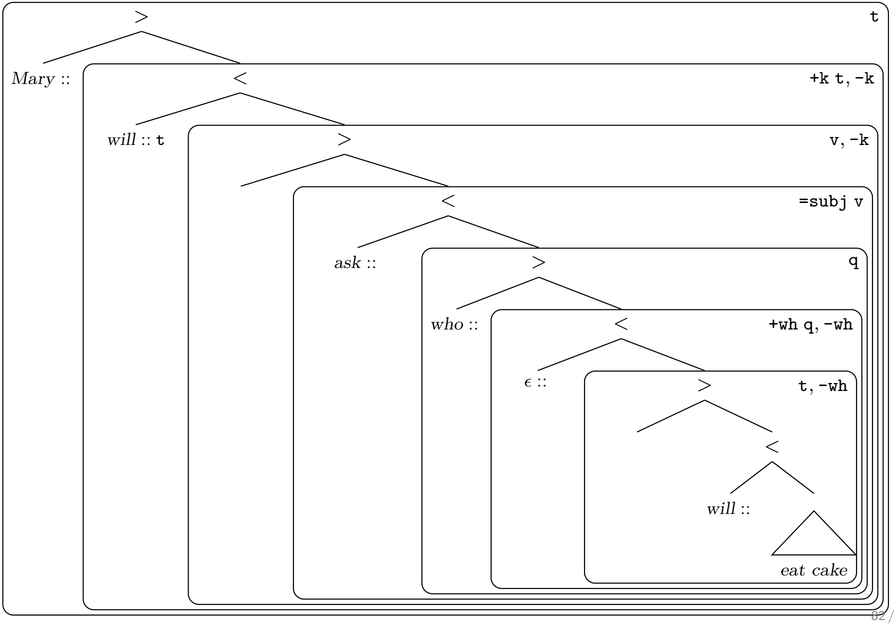
Extending with *Mary will ask ...*



Extending with *Mary will ask . . .*

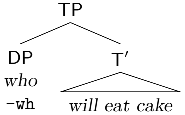


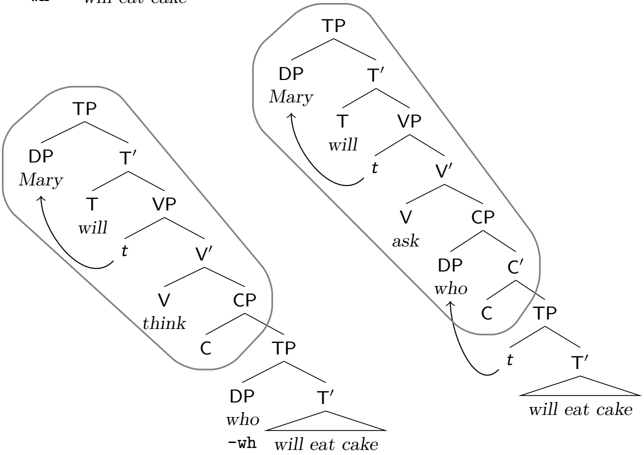
Extending with *Mary will ask ...*



Which extensions create "loops"?

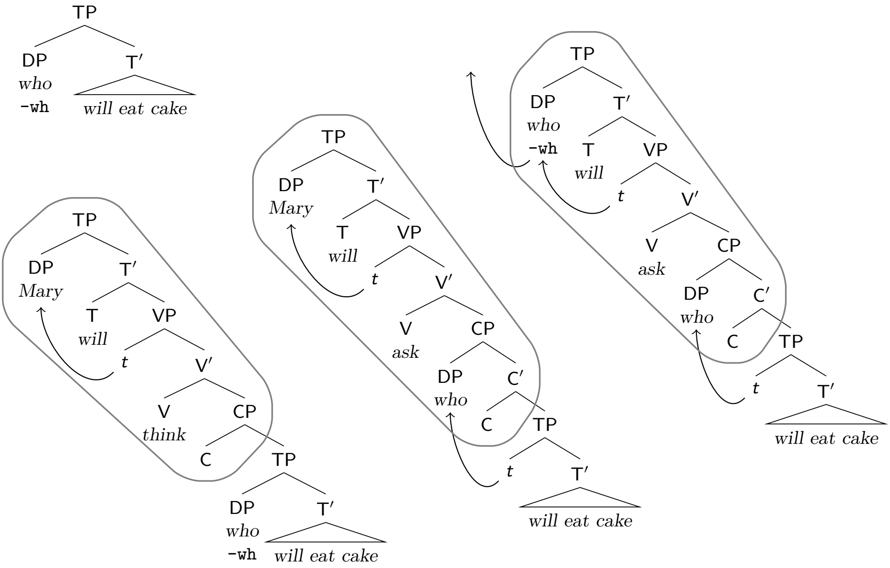
Starting point:



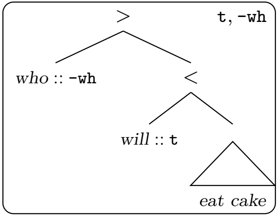


Which extensions create “loops”?

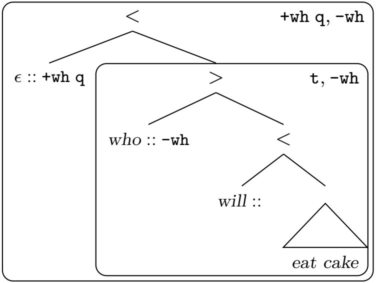
Starting point:



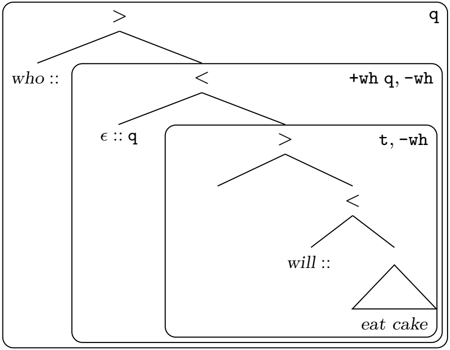
Extending with *who will ask . . .*



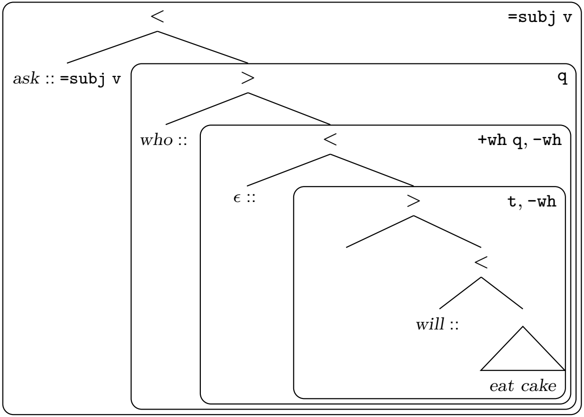
Extending with *who will ask* . . .



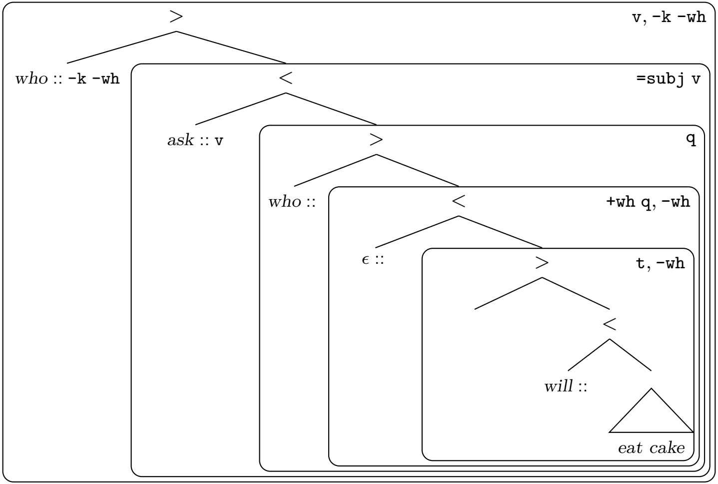
Extending with *who will ask ...*



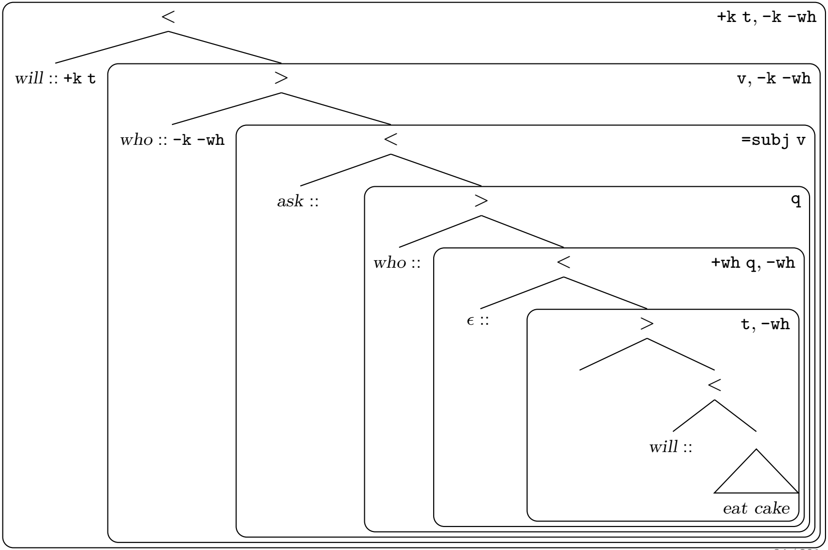
Extending with *who will ask ...*



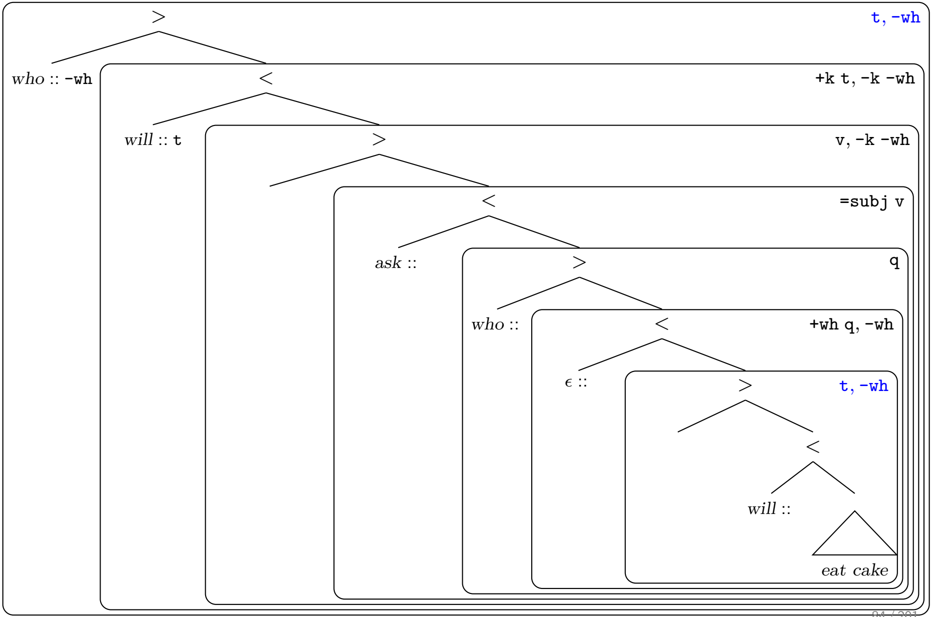
Extending with *who will ask …*



Extending with *who will ask . . .*

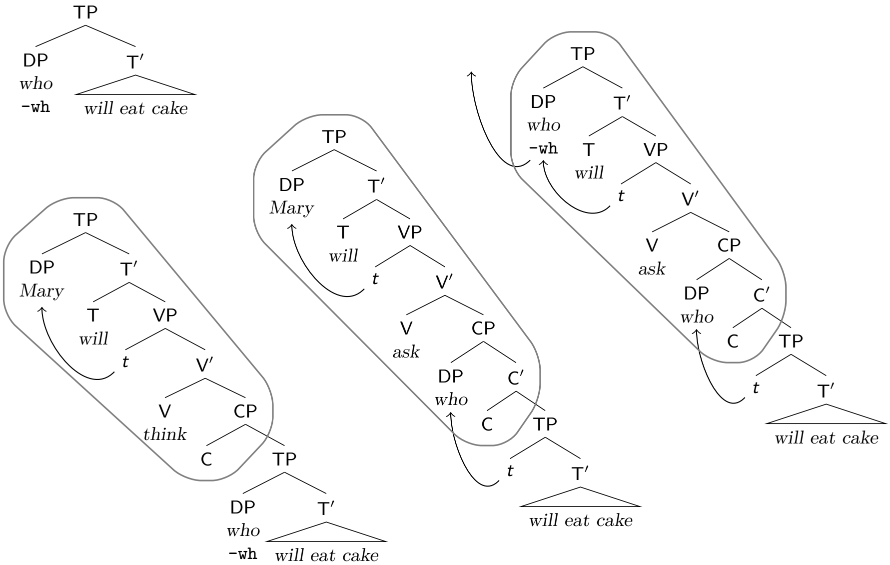


Extending with *who will ask* ...



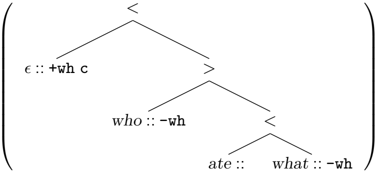
Which extensions create “loops”?

Starting point:



Importance of the SMC

The SMC ensures that there is a finite number of types (that we care about).

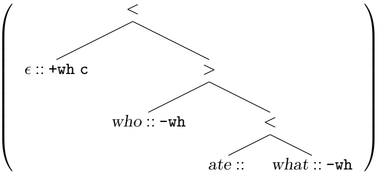
Recall: MOVE 

is **undefined**

(Michaelis 2001)

Importance of the SMC

The SMC ensures that there is a finite number of types (that we care about).

Recall: MOVE 

is **undefined**

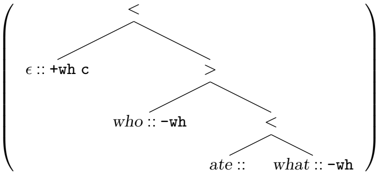
- So MOVE cannot be applied to expressions of type $\langle \texttt{+wh c}, \texttt{-wh}, \texttt{-wh} \rangle$.

(Michaelis 2001)

Importance of the SMC

The SMC ensures that there is a finite number of types (that we care about).

Recall: MOVE



is **undefined**

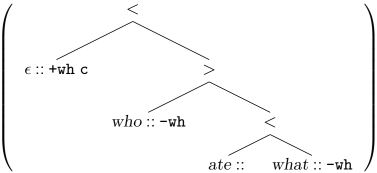
- So MOVE cannot be applied to expressions of type ⟨+wh c, -wh, -wh⟩.
- Nor to expressions of type ⟨+wh c, -wh -k, -wh⟩.
- These are “dead end” types.

(Michaelis 2001)

Importance of the SMC

The SMC ensures that there is a finite number of types (that we care about).

Recall: MOVE



is **undefined**

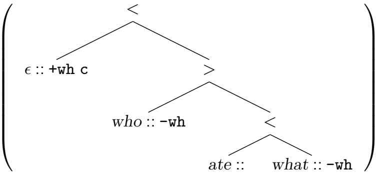
- So MOVE cannot be applied to expressions of type ⟨+wh c, -wh, -wh⟩.
- Nor to expressions of type ⟨+wh c, -wh -k, -wh⟩.
- These are "dead end" types.
- An expression of type ⟨t, -wh -k, -wh⟩ can be the input to MERGE.

(Michaelis 2001)

Importance of the SMC

The SMC ensures that there is a finite number of types (that we care about).

Recall: MOVE



is **undefined**

- So MOVE cannot be applied to expressions of type $\langle$+wh c, -wh, -wh$\rangle$.
- Nor to expressions of type $\langle$+wh c, -wh -k, -wh$\rangle$.
- These are "dead end" types.

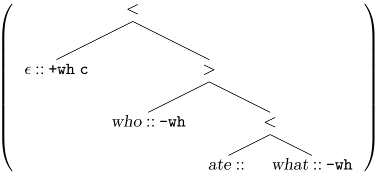
- An expression of type $\langle$t, -wh -k, -wh$\rangle$ can be the input to MERGE.
- But such types are also bound to lead to dead ends.

(Michaelis 2001)

Importance of the SMC

The SMC ensures that there is a finite number of types (that we care about).

Recall: MOVE



is **undefined**

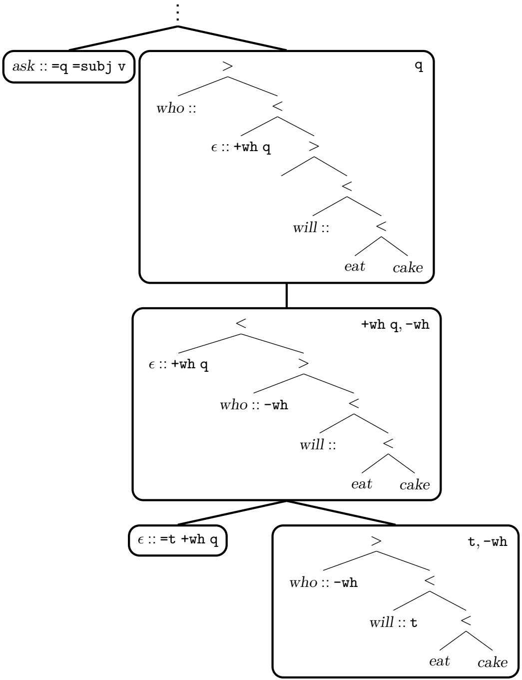
- So MOVE cannot be applied to expressions of type $\langle$+wh c, -wh, -wh$\rangle$.
- Nor to expressions of type $\langle$+wh c, -wh -k, -wh$\rangle$.
- These are "dead end" types.

- An expression of type $\langle$t, -wh -k, -wh$\rangle$ can be the input to MERGE.
- But such types are also bound to lead to dead ends.

So any type of the form $\langle \alpha, \ldots, \texttt{-f}\alpha_i, \ldots, \texttt{-f}\alpha_j, \ldots \rangle$ is not **useful**.
Thus there are only a finite number of useful types.

(Michaelis 2001)

Outline

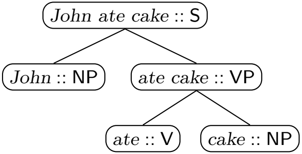

⋮
ask :: =q =subj v
>
q
who ::
<
ϵ :: +wh q
>
<
will ::
<
eat
cake
<
+wh q, -wh
ϵ :: +wh q
>
who :: -wh
<
will ::
<
eat
cake
ϵ :: =t +wh q
>
t, -wh
who :: -wh
<
will :: t
<
eat
cake

A possible concern

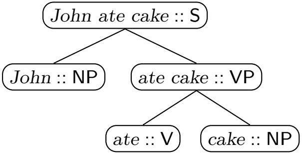
Question

“But hasn’t our eventual derived expression lost the information that ‘cake’ is a DP?”

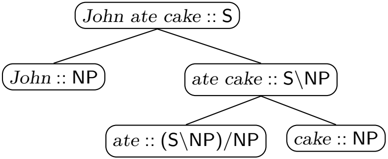
Derivations



Derivations



$$\dfrac{\mathit{John} :: \mathsf{NP} \qquad \dfrac{\mathit{ate} :: (\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP} \qquad \mathit{cake} :: \mathsf{NP}}{\mathit{ate\ cake} :: \mathsf{S}\backslash\mathsf{NP}}}{\mathit{John\ ate\ cake} :: \mathsf{S}}$$



A possible concern

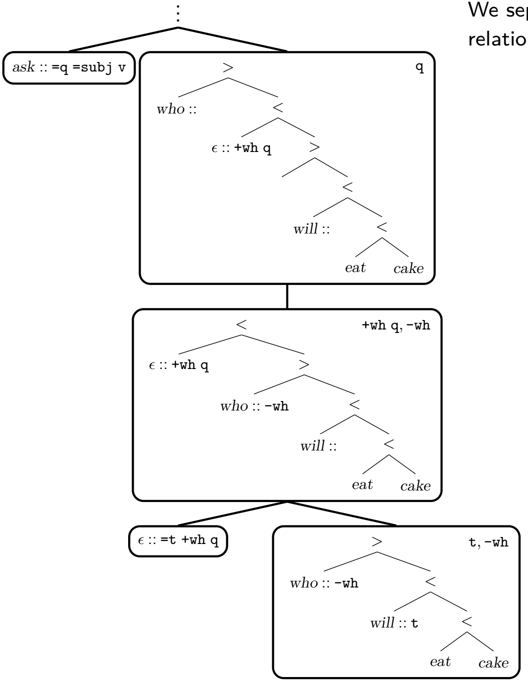
Question

"But hasn't our eventual derived expression lost the information that 'cake' is a DP?"

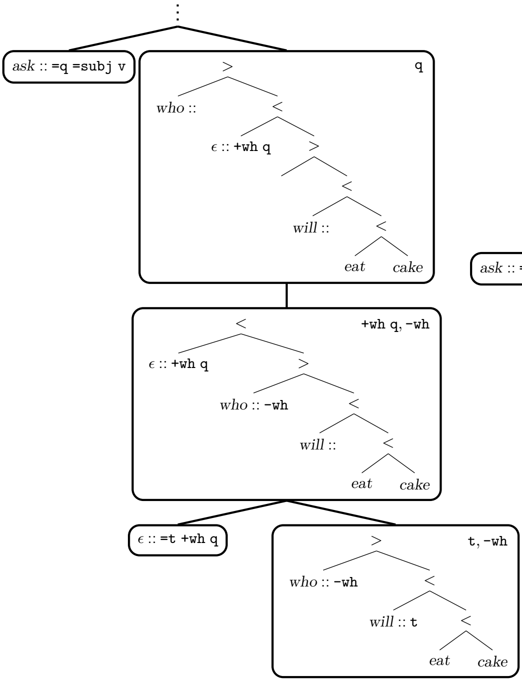
Answer

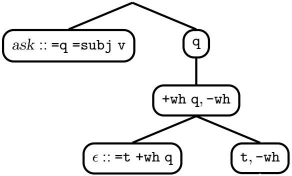
Yes, but only in the same way that $\mathit{John\ ate\ cake}$ :: S has also lost this information.

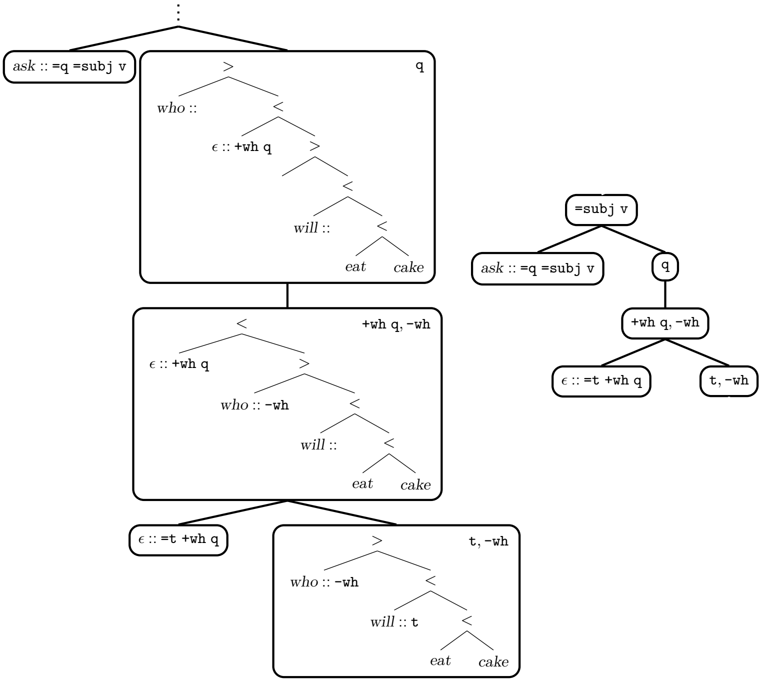
The point is not that we can look at the whole derivation to retrieve that information, the point is that the information has already done its job.

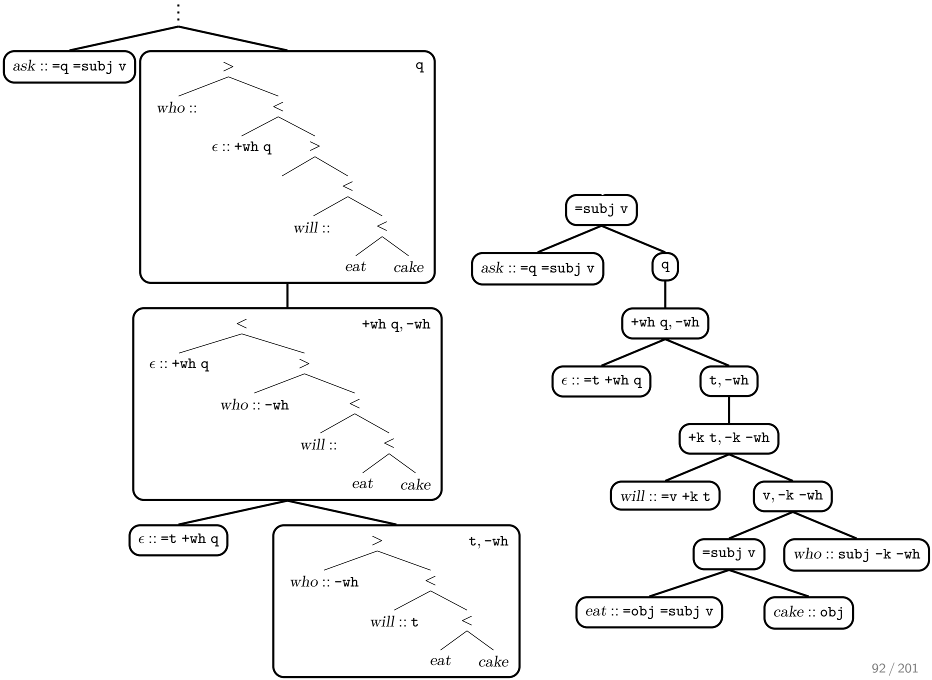


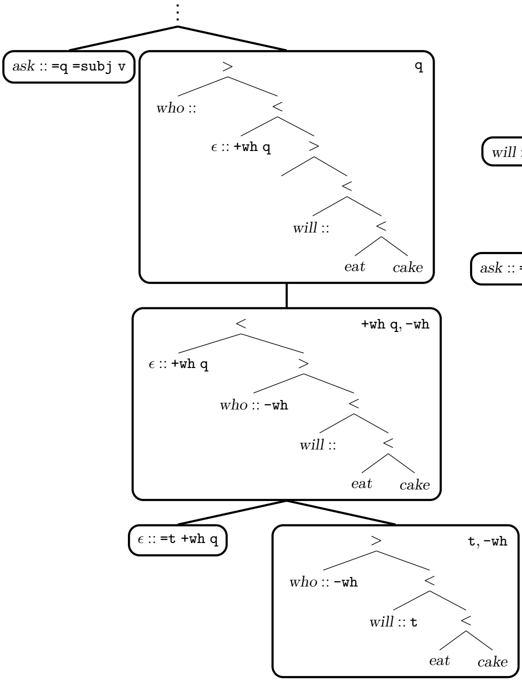
We separate the derivational precedence relation from the part-whole relation


ask :: =q =subj v
>
q
who ::
<
ϵ :: +wh q
>
<
will ::
<
eat
cake
<
+wh q, -wh
ϵ :: +wh q
>
who :: -wh
<
will ::
<
eat
cake
ϵ :: =t +wh q
>
t, -wh
who :: -wh
<
will :: t
<
eat
cake

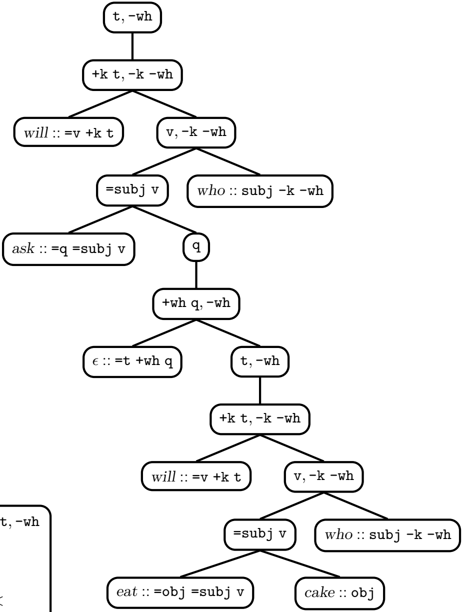

ask :: =q =subj v
q
+wh q, -wh
ϵ :: =t +wh q
t, -wh


ask :: =q =subj v
q
who ::
ϵ :: +wh q
will ::
eat
cake
+wh q, -wh
ϵ :: +wh q
who :: -wh
will ::
eat
cake
ϵ :: =t +wh q
t, -wh
who :: -wh
will :: t
eat
cake
=subj v
ask :: =q =subj v
q
+wh q, -wh
ϵ :: =t +wh q
t, -wh

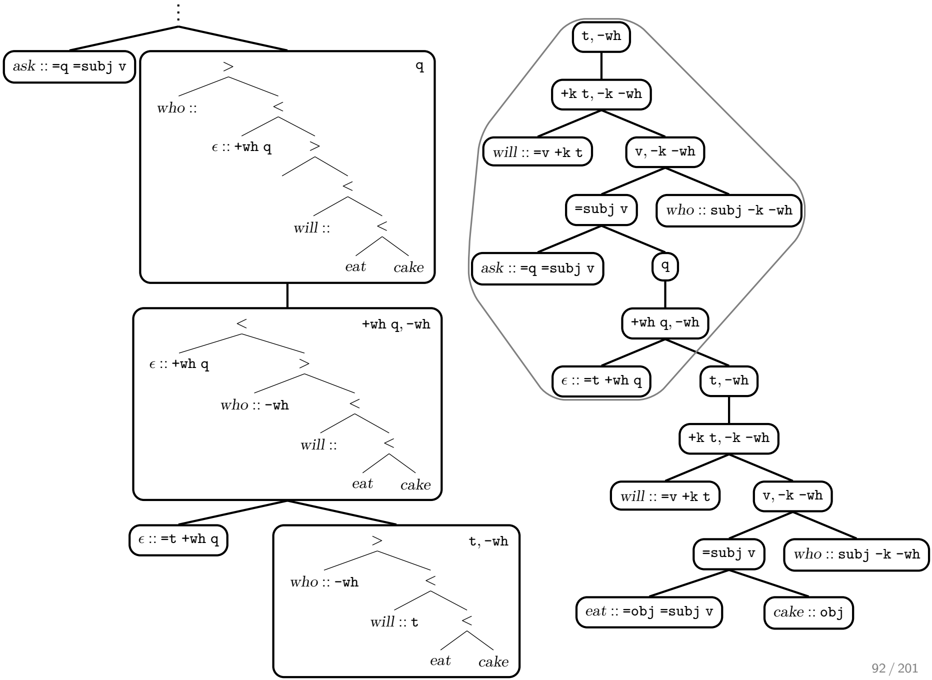

ask :: =q =subj v
>
q
who ::
<
ϵ :: +wh q
>
<
will ::
<
eat
cake
<
+wh q, -wh
ϵ :: +wh q
>
who :: -wh
<
will ::
<
eat
cake
ϵ :: =t +wh q
>
t, -wh
who :: -wh
<
will :: t
<
eat
cake
=subj v
ask :: =q =subj v
q
+wh q, -wh
ϵ :: =t +wh q
t, -wh
+k t, -k -wh
will :: =v +k t
v, -k -wh
=subj v
who :: subj -k -wh
eat :: =obj =subj v
cake :: obj



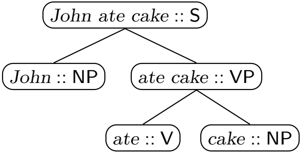
⋮
ask :: =q =subj v
q
>
who ::
<
ϵ :: +wh q
>
<
will ::
<
eat
cake
+wh q, -wh
<
ϵ :: +wh q
>
who :: -wh
<
will ::
<
eat
cake
ϵ :: =t +wh q
t, -wh
>
who :: -wh
<
will :: t
<
eat
cake



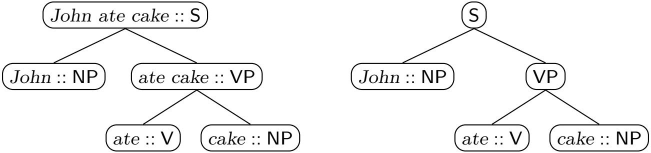
t, -wh
+k t, -k -wh
will :: =v +k t
v, -k -wh
=subj v
who :: subj -k -wh
ask :: =q =subj v
q
+wh q, -wh
ϵ :: =t +wh q
t, -wh
+k t, -k -wh
will :: =v +k t
v, -k -wh
=subj v
who :: subj -k -wh
eat :: =obj =subj v
cake :: obj


ask :: =q =subj v
q
who ::
ϵ :: +wh q
will ::
eat
cake
+wh q, -wh
ϵ :: +wh q
who :: -wh
will ::
eat
cake
ϵ :: =t +wh q
t, -wh
who :: -wh
will :: t
eat
cake
t, -wh
+k t, -k -wh
will :: =v +k t
v, -k -wh
=subj v
who :: subj -k -wh
ask :: =q =subj v
q
+wh q, -wh
ϵ :: =t +wh q
t, -wh
+k t, -k -wh
will :: =v +k t
v, -k -wh
=subj v
who :: subj -k -wh
eat :: =obj =subj v
cake :: obj

Labeling of internal nodes

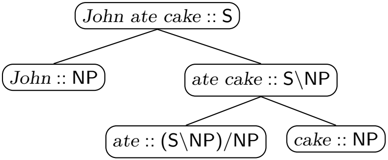


Labeling of internal nodes



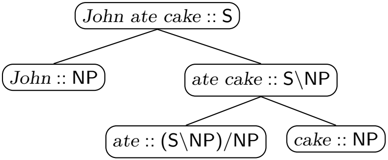
Labeling of internal nodes

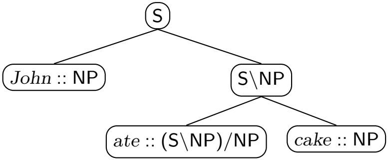
$$\dfrac{\mathit{John} :: \mathsf{NP} \qquad \dfrac{\mathit{ate} :: (\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP} \qquad \mathit{cake} :: \mathsf{NP}}{\mathit{ate\ cake} :: \mathsf{S}\backslash\mathsf{NP}}}{\mathit{John\ ate\ cake} :: \mathsf{S}}$$

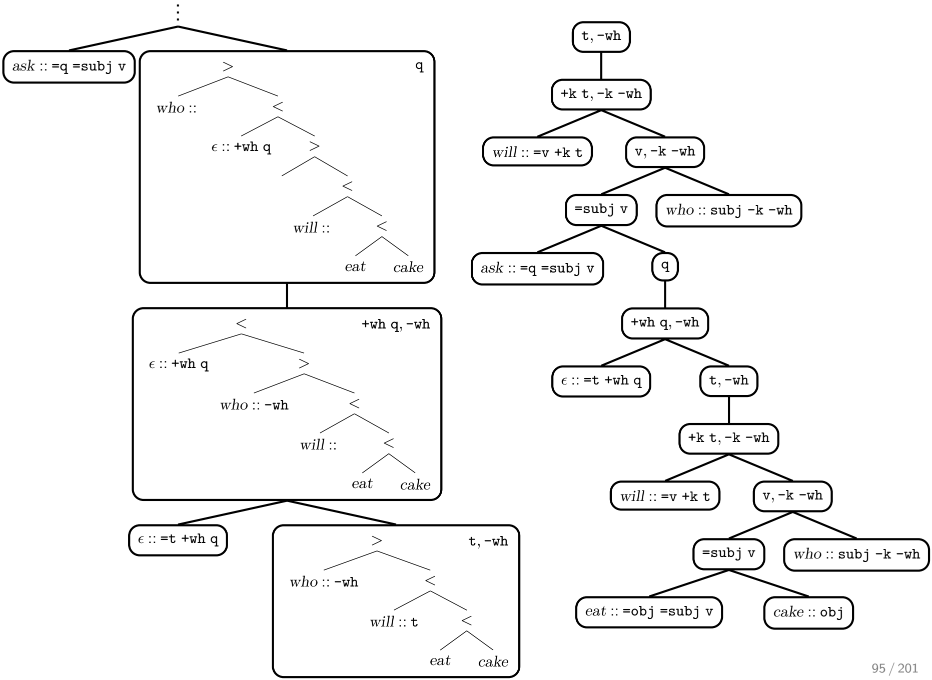


Labeling of internal nodes

$$\dfrac{\mathit{John} :: \mathsf{NP} \qquad \dfrac{\mathit{ate} :: (\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP} \qquad \mathit{cake} :: \mathsf{NP}}{\mathit{ate\ cake} :: \mathsf{S}\backslash\mathsf{NP}}}{\mathit{John\ ate\ cake} :: \mathsf{S}}$$






ask :: =q =subj v
q
who ::
ϵ :: +wh q
will ::
eat
cake
+wh q, -wh
ϵ :: +wh q
who :: -wh
will ::
eat
cake
ϵ :: =t +wh q
t, -wh
who :: -wh
will :: t
eat
cake
t, -wh
+k t, -k -wh
will :: =v +k t
v, -k -wh
=subj v
who :: subj -k -wh
ask :: =q =subj v
q
+wh q, -wh
ϵ :: =t +wh q
t, -wh
+k t, -k -wh
will :: =v +k t
v, -k -wh
=subj v
who :: subj -k -wh
eat :: =obj =subj v
cake :: obj

Context-free structure

$$
\begin{aligned}
\langle \texttt{=subj v} \rangle &\rightarrow \langle \texttt{=q =subj v} \rangle \quad \langle \texttt{q} \rangle \\
\langle \texttt{q} \rangle &\rightarrow \langle \texttt{+wh q}, \texttt{-wh} \rangle \\
\langle \texttt{+wh q}, \texttt{-wh} \rangle &\rightarrow \langle \texttt{=t +wh q} \rangle \quad \langle \texttt{t}, \texttt{-wh} \rangle
\end{aligned}
$$

Context-free structure

$$
\begin{array}{rcll}
\langle \texttt{=subj v} \rangle & \rightarrow & \langle \texttt{=q =subj v} \rangle & \langle \texttt{q} \rangle \\
\langle \texttt{q} \rangle & \rightarrow & \langle \texttt{+wh q}, \texttt{-wh} \rangle & \\
\langle \texttt{+wh q}, \texttt{-wh} \rangle & \rightarrow & \langle \texttt{=t +wh q} \rangle & \langle \texttt{t}, \texttt{-wh} \rangle
\end{array}
$$

General schemas for MERGE steps (approximate):

$$
\begin{array}{rcll}
\langle \gamma, \alpha_1, \dots, \alpha_j, \beta_1, \dots, \beta_k \rangle & \rightarrow & \langle \texttt{=f}\gamma, \alpha_1, \dots, \alpha_j \rangle & \langle \texttt{f}, \beta_1, \dots, \beta_k \rangle \\
\langle \gamma, \alpha_1, \dots, \alpha_j, \delta, \beta_1, \dots, \beta_k \rangle & \rightarrow & \langle \texttt{=f}\gamma, \alpha_1, \dots, \alpha_j \rangle & \langle \texttt{f}\delta, \beta_1, \dots, \beta_k \rangle
\end{array}
$$

General schemas for MOVE steps (approximate):

$$
\begin{array}{rcl}
\langle \gamma, \alpha_1, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_k \rangle & \rightarrow & \langle \texttt{+f}\gamma, \alpha_1, \dots, \alpha_{i-1}, \texttt{-f}, \alpha_{i+1}, \dots, \alpha_k \rangle \\
\langle \gamma, \alpha_1, \dots, \alpha_{i-1}, \delta, \alpha_{i+1}, \dots, \alpha_k \rangle & \rightarrow & \langle \texttt{+f}\gamma, \alpha_1, \dots, \alpha_{i-1}, \texttt{-f}\delta, \alpha_{i+1}, \dots, \alpha_k \rangle
\end{array}
$$

Context-free structure

$$\begin{array}{rcll} \langle \texttt{=subj v} \rangle & \rightarrow & \langle \texttt{=q =subj v} \rangle & \langle \texttt{q} \rangle \\ \langle \texttt{q} \rangle & \rightarrow & \langle \texttt{+wh q}, \texttt{-wh} \rangle & \\ \langle \texttt{+wh q}, \texttt{-wh} \rangle & \rightarrow & \langle \texttt{=t +wh q} \rangle & \langle \texttt{t}, \texttt{-wh} \rangle \end{array}$$

General schemas for MERGE steps (approximate):

$$\begin{array}{rcll} \langle \gamma, \alpha_1, \dots, \alpha_j, \beta_1, \dots, \beta_k \rangle & \rightarrow & \langle \texttt{=f}\gamma, \alpha_1, \dots, \alpha_j \rangle & \langle \texttt{f}, \beta_1, \dots, \beta_k \rangle \\ \langle \gamma, \alpha_1, \dots, \alpha_j, \delta, \beta_1, \dots, \beta_k \rangle & \rightarrow & \langle \texttt{=f}\gamma, \alpha_1, \dots, \alpha_j \rangle & \langle \texttt{f}\delta, \beta_1, \dots, \beta_k \rangle \end{array}$$

General schemas for MOVE steps (approximate):

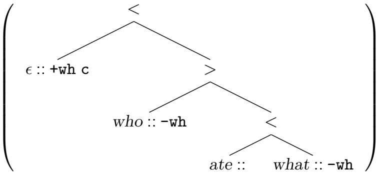
$$\begin{array}{rcl} \langle \gamma, \alpha_1, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_k \rangle & \rightarrow & \langle \texttt{+f}\gamma, \alpha_1, \dots, \alpha_{i-1}, \texttt{-f}, \alpha_{i+1}, \dots, \alpha_k \rangle \\ \langle \gamma, \alpha_1, \dots, \alpha_{i-1}, \delta, \alpha_{i+1}, \dots, \alpha_k \rangle & \rightarrow & \langle \texttt{+f}\gamma, \alpha_1, \dots, \alpha_{i-1}, \texttt{-f}\delta, \alpha_{i+1}, \dots, \alpha_k \rangle \end{array}$$

- MOVE steps **change** something without **combining** it with anything
- Compare with unary CFG rules, or type-raising in CCG, or ...

Importance of the SMC

The SMC ensures that there is a finite number of types (that we care about).

Recall: MOVE



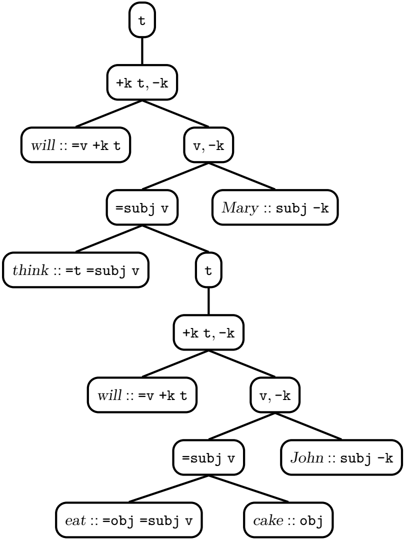
is **undefined**

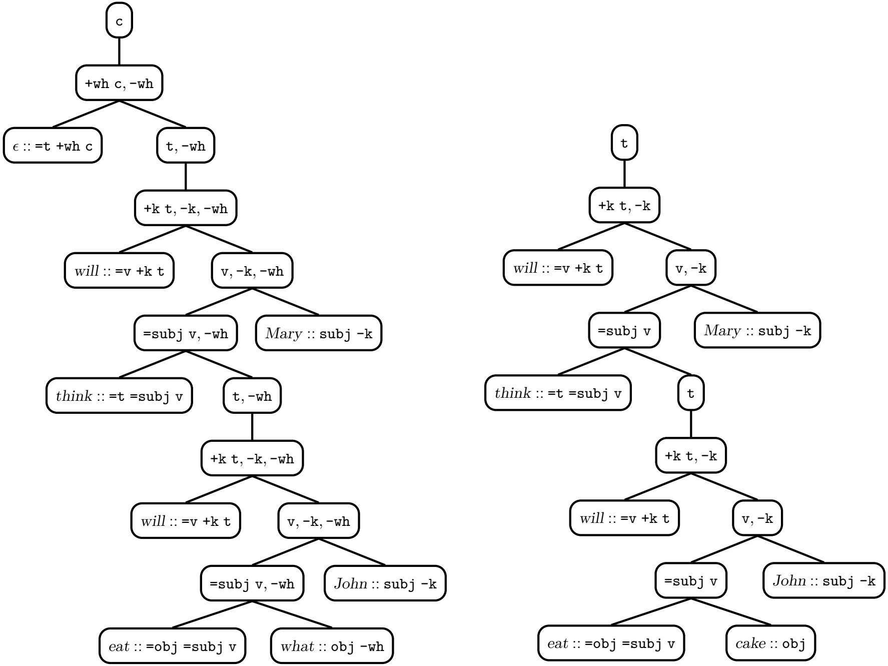
- So MOVE cannot be applied to expressions of type $\langle$+wh c, -wh, -wh$\rangle$.
- Nor to expressions of type $\langle$+wh c, -wh -k, -wh$\rangle$.
- These are "dead end" types.

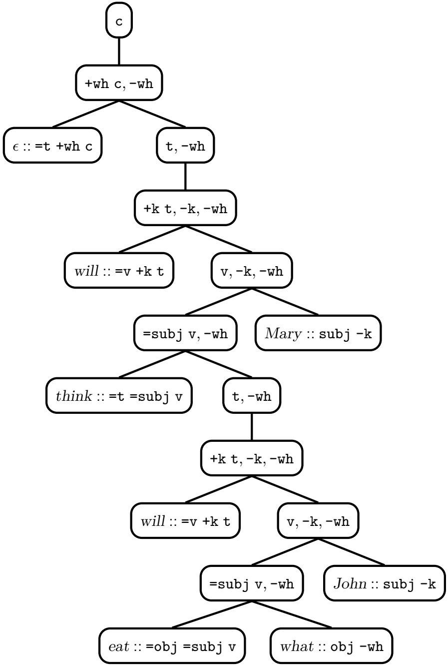
- An expression of type $\langle$t, -wh -k, -wh$\rangle$ can be the input to MERGE.
- But such types are also bound to lead to dead ends.

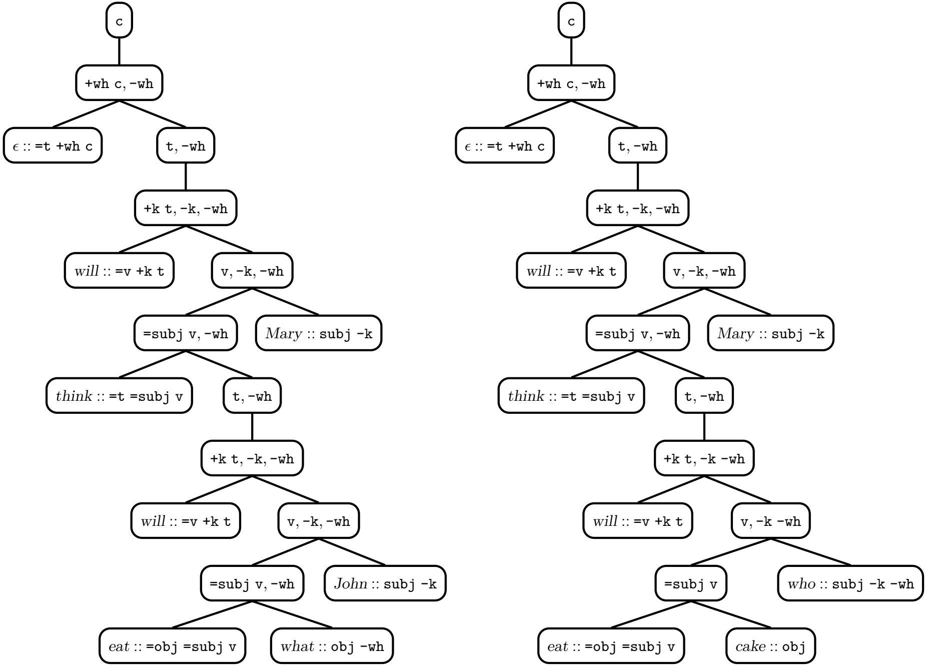
So any type of the form $\langle\alpha, \ldots, \texttt{-f}\alpha_i, \ldots, \texttt{-f}\alpha_j, \ldots\rangle$ is not **useful**.
Thus there are only a finite number of useful types.

(Michaelis 2001)


t
+k t, -k
will :: =v +k t
v, -k
=subj v
Mary :: subj -k
think :: =t =subj v
t
+k t, -k
will :: =v +k t
v, -k
=subj v
John :: subj -k
eat :: =obj =subj v
cake :: obj


c
+wh c, -wh
ϵ :: =t +wh c
t, -wh
+k t, -k, -wh
will :: =v +k t
v, -k, -wh
=subj v, -wh
Mary :: subj -k
think :: =t =subj v
t, -wh
+k t, -k, -wh
will :: =v +k t
v, -k, -wh
=subj v, -wh
John :: subj -k
eat :: =obj =subj v
what :: obj -wh
t
+k t, -k
will :: =v +k t
v, -k
=subj v
Mary :: subj -k
think :: =t =subj v
t
+k t, -k
will :: =v +k t
v, -k
=subj v
John :: subj -k
eat :: =obj =subj v
cake :: obj


c
+wh c, -wh
ϵ :: =t +wh c
t, -wh
+k t, -k, -wh
will :: =v +k t
v, -k, -wh
=subj v, -wh
Mary :: subj -k
think :: =t =subj v
t, -wh
+k t, -k, -wh
will :: =v +k t
v, -k, -wh
=subj v, -wh
John :: subj -k
eat :: =obj =subj v
what :: obj -wh


c
+wh c, -wh
ϵ :: =t +wh c
t, -wh
+k t, -k, -wh
will :: =v +k t
v, -k, -wh
=subj v, -wh
Mary :: subj -k
think :: =t =subj v
t, -wh
+k t, -k, -wh
will :: =v +k t
v, -k, -wh
=subj v, -wh
John :: subj -k
eat :: =obj =subj v
what :: obj -wh
c
+wh c, -wh
ϵ :: =t +wh c
t, -wh
+k t, -k, -wh
will :: =v +k t
v, -k, -wh
=subj v, -wh
Mary :: subj -k
think :: =t =subj v
t, -wh
+k t, -k -wh
will :: =v +k t
v, -k -wh
=subj v
who :: subj -k -wh
eat :: =obj =subj v
cake :: obj

Part 1: Grammars and cognitive hypotheses

- What is a grammar?
- What can grammars do?
- Concrete illustration of a target: Surprisal

Parts 2–4: Assembling the pieces

- Minimalist Grammars (MGs)
- MGs and MCFGs
- Probabilities on MGs

Part 5: Learning and wrap-up

- Something slightly different: Learning model
- Recap and open questions

Billot, S. and Lang, B. (1989). The structure of shared forests in ambiguous parsing. In *Proceedings of the 1989 Meeting of the Association of Computational Linguistics*.

Chomsky, N. (1965). *Aspects of the Theory of Syntax*. MIT Press, Cambridge, MA.

Chomsky, N. (1980). *Rules and Representations*. Columbia University Press, New York.

Ferreira, F. (2005). Psycholinguistics, formal grammars, and cognitive science. *The Linguistic Review*, 22:365–380.

Frazier, L. and Clifton, C. (1996). *Construal*. MIT Press, Cambridge, MA.

Gärtner, H.-M. and Michaelis, J. (2010). On the Treatment of Multiple-Wh Interrogatives in Minimalist Grammars. In Hanneforth, T. and Fanselow, G., editors, *Language and Logos*, pages 339–366. Akademie Verlag, Berlin.

Gibson, E. and Wexler, K. (1994). Triggers. *Linguistic Inquiry*, 25:407–454.

Hale, J. (2006). Uncertainty about the rest of the sentence. *Cognitive Science*, 30:643–Âŋ672.

Hale, J. T. (2001). A probabilistic earley parser as a psycholinguistic model. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics*.

Hunter, T. (2011). Insertion Minimalist Grammars: Eliminating redundancies between merge and move. In Kanazawa, M., Kornai, A., Kracht, M., and Seki, H., editors, *The Mathematics of Language (MOL 12 Proceedings)*, volume 6878 of *LNCS*, pages 90–107, Berlin Heidelberg. Springer.

Hunter, T. and Dyer, C. (2013). Distributions on minimalist grammar derivations. In *Proceedings of the 13th Meeting on the Mathematics of Language*.

Koopman, H. and Szabolcsi, A. (2000). *Verbal Complexes*. MIT Press, Cambridge, MA.

Lang, B. (1988). Parsing incomplete sentences. In *Proceedings of the 12th International Conference on Computational Linguistics*, pages 365–371.

Levy, R. (2008). Expectation-based syntactic comprehension. *Cognition*, 106(3):1126–1177.

Michaelis, J. (2001). Derivational minimalism is mildly context-sensitive. In Moortgat, M., editor, *Logical Aspects of Computational Linguistics*, volume 2014 of *LNCS*, pages 179–198. Springer, Berlin Heidelberg.

Miller, G. A. and Chomsky, N. (1963). Finitary models of language users. In Luce, R. D., Bush, R. R., and Galanter, E., editors, *Handbook of Mathematical Psychology*, volume 2. Wiley and Sons, New York.

Morrill, G. (1994). *Type Logical Grammar: Categorial Logic of Signs*. Kluwer, Dordrecht.

Nederhof, M. J. and Satta, G. (2008). Computing partition functions of pcfgs. *Research on Language and Computation*, 6(2):139–162.

Seki, H., Matsumara, T., Fujii, M., and Kasami, T. (1991). On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229.

Stabler, E. P. (2006). Sidewards without copying. In Wintner, S., editor, *Proceedings of The 11th Conference on Formal Grammar*, pages 157–170, Stanford, CA. CSLI Publications.

Stabler, E. P. (2011). Computational perspectives on minimalism. In Boeckx, C., editor, *The Oxford Handbook of Linguistic Minimalism*. Oxford University Press, Oxford.

Stabler, E. P. and Keenan, E. L. (2003). Structural similarity within and among languages. *Theoretical Computer Science*, 293:345–363.

References III

Vijay-Shanker, K., Weir, D. J., and Joshi, A. K. (1987). Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Meeting of the Association for Computational Linguistics*, pages 104–111.

Weir, D. (1988). *Characterizing mildly context-sensitive grammar formalisms*. PhD thesis, University of Pennsylvania.

Yngve, V. H. (1960). A model and an hypothesis for language structure. In *Proceedings of the American Philosophical Society*, volume 104, pages 444–466.