Sharpening the empirical claims of generative syntax through formalization

Tim Hunter

University of Minnesota, Twin Cities

ESSLLI, August 2015

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

Sharpening the empirical claims of generative syntax
through formalization

Tim Hunter — ESSLLI, August 2015

Part 3

MGs and MCFGs

Where we're up to

We've seen:

- MGs with operations defined that manipulated trees
- that the structure that "really matters" (e.g. for recursion) can be boiled down to funny-looking "derivation trees" (with things like $\langle \texttt{t}, \texttt{-k} \rangle$ at the non-leaf nodes)

Now:

- A way to think of how these derivation trees relate to surface strings (without going via trees)
- In some ways not totally necessary for the rest of the course, but helpful

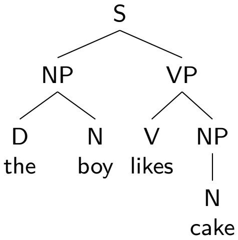
Later:

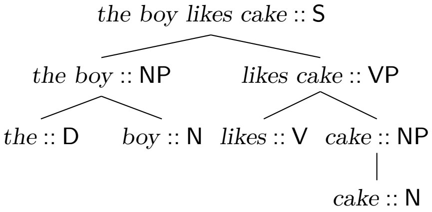
- Adding probabilities to MGs: in a way that sort of works, and does some good stuff, but doesn't do everything we'd want
- Adding probabilities to MGs: in an even better way

Outline

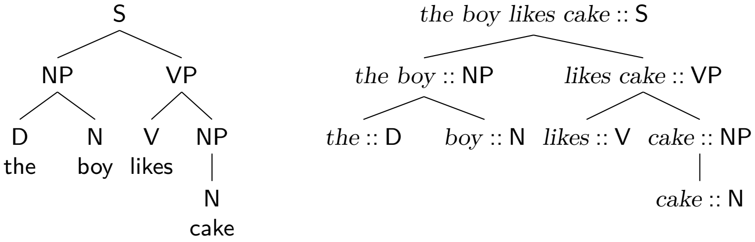
Outline

Trees


S
NP
VP
D
the
N
boy
V
likes
NP
N
cake


the boy likes cake :: S
the boy :: NP
likes cake :: VP
the :: D
boy :: N
likes :: V
cake :: NP
cake :: N

Trees



How to think of a tree:

- less as a picture of a string
- more as a graphical representation of how a string was constructed, with the string "at" the top node

Two sides of a CFG rule

A rule like ‘S $\rightarrow$ NP VP’ says two things:

- What combines with what:
  An NP and a VP can combine to form an S
- How to produce a string of the new category:
  Put the NP-string to the left of the VP-string

More explicitly:

$$st :: \mathrm{S} \quad \rightarrow \quad s :: \mathrm{NP} \quad t :: \mathrm{VP}$$

Example: X-bar theory

Japanese

XP $\rightarrow$ Spec X$'$
X$'$ $\rightarrow$ Comp X

English

XP $\rightarrow$ Spec X$'$
X$'$ $\rightarrow$ X Comp

Example: X-bar theory

Japanese

$\text{XP} \rightarrow \text{Spec X}'$

$\text{X}' \rightarrow \text{Comp X}$

English

$\text{XP} \rightarrow \text{Spec X}'$

$\text{X}' \rightarrow \text{X Comp}$

Japanese

$$st :: \text{XP} \quad \rightarrow \quad s :: \text{Spec} \quad t :: \text{X}'$$

$$st :: \text{X}' \quad \rightarrow \quad s :: \text{Comp} \quad t :: \text{X}$$

English

$$st :: \text{XP} \quad \rightarrow \quad s :: \text{Spec} \quad t :: \text{X}'$$

$$ts :: \text{X}' \quad \rightarrow \quad s :: \text{Comp} \quad t :: \text{X}$$

Example: X-bar theory

Japanese

XP → Spec X′
X′ → Comp X

English

XP → Spec X′
X′ → X Comp

Japanese

$st :: \text{XP} \rightarrow s :: \text{Spec} \quad t :: \text{X}'$

$st :: \text{X}' \rightarrow s :: \text{Comp} \quad t :: \text{X}$

English

$st :: \text{XP} \rightarrow s :: \text{Spec} \quad t :: \text{X}'$

$ts :: \text{X}' \rightarrow s :: \text{Comp} \quad t :: \text{X}$

John-ga Mary-o mita :: VP
John-ga :: Spec
Mary-o mita :: V′
Mary-o :: Comp
mita :: V

John saw Mary :: VP
John :: Spec
saw Mary :: V′
Mary :: Comp
saw :: V

Outline

Concatenative and non-concatenative operations

Concatenative morphology:

play + ed	$\rightsquigarrow$	played
play + ing	$\rightsquigarrow$	playing
play + s	$\rightsquigarrow$	plays

Non-concatenative morphology:

(k,t,b) + (i,aa)	$\rightsquigarrow$	kitaab	(“book”)
(k,t,b) + (aa,i)	$\rightsquigarrow$	kaatib	(“writer”)
(k,t,b) + (ma,uu)	$\rightsquigarrow$	maktuub	(“written”)
(k,t,b) + (a,i,a)	$\rightsquigarrow$	katiba	(“document”)

Concatenative and non-concatenative operations

Concatenative morphology:

play + ed	$\rightsquigarrow$	played
play + ing	$\rightsquigarrow$	playing
play + s	$\rightsquigarrow$	plays

Non-concatenative morphology:

(k,t,b) + (i,aa)	$\rightsquigarrow$	kitaab	("book")
(k,t,b) + (aa,i)	$\rightsquigarrow$	kaatib	("writer")
(k,t,b) + (ma,uu)	$\rightsquigarrow$	maktuub	("written")
(k,t,b) + (a,i,a)	$\rightsquigarrow$	katiba	("document")

Concatenative syntax:

plays + tennis	$\rightsquigarrow$	plays tennis
plays + soccer	$\rightsquigarrow$	plays soccer
John + plays soccer	$\rightsquigarrow$	John plays soccer
Mary + plays soccer	$\rightsquigarrow$	Mary plays soccer

Concatenative and non-concatenative operations

Concatenative morphology:

play + ed	$\rightsquigarrow$	played
play + ing	$\rightsquigarrow$	playing
play + s	$\rightsquigarrow$	plays

Non-concatenative morphology:

(k,t,b) + (i,aa)	$\rightsquigarrow$	kitaab	(“book”)
(k,t,b) + (aa,i)	$\rightsquigarrow$	kaatib	(“writer”)
(k,t,b) + (ma,uu)	$\rightsquigarrow$	maktuub	(“written”)
(k,t,b) + (a,i,a)	$\rightsquigarrow$	katiba	(“document”)

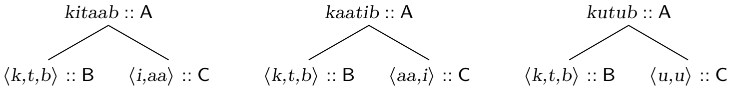
Concatenative syntax:

plays + tennis	$\rightsquigarrow$	plays tennis
plays + soccer	$\rightsquigarrow$	plays soccer
John + plays soccer	$\rightsquigarrow$	John plays soccer
Mary + plays soccer	$\rightsquigarrow$	Mary plays soccer

Non-concatenative syntax:

seems + (John, to be tall)	$\rightsquigarrow$	John seems to be tall
seems + (Mary, to be intelligent)	$\rightsquigarrow$	Mary seems to be intelligent
did + (John see, who)	$\rightsquigarrow$	who did John see
did + (Mary meet, who)	$\rightsquigarrow$	who did Mary meet

Non-concatenative morphology



$$stuvw :: \mathsf{A} \quad \rightarrow \quad \langle s, u, w \rangle :: \mathsf{B} \quad \langle t, v \rangle :: \mathsf{C}$$

Non-concatenative morphology

kitaab :: A
$\langle$*k,t,b*$\rangle$ :: B $\quad$ $\langle$*i,aa*$\rangle$ :: C

kaatib :: A
$\langle$*k,t,b*$\rangle$ :: B $\quad$ $\langle$*aa,i*$\rangle$ :: C

kutub :: A
$\langle$*k,t,b*$\rangle$ :: B $\quad$ $\langle$*u,u*$\rangle$ :: C

$$stuvw :: \mathsf{A} \quad \rightarrow \quad \langle s, u, w\rangle :: \mathsf{B} \quad \langle t, v\rangle :: \mathsf{C}$$

gespielt :: E
spiel :: A $\quad$ $\langle$*ge,t*$\rangle$ :: D

gekauft :: E
kauf :: A $\quad$ $\langle$*ge,t*$\rangle$ :: D

gemacht :: E
mach :: A $\quad$ $\langle$*ge,t*$\rangle$ :: D

$$stu :: \mathsf{E} \quad \rightarrow \quad t :: \mathsf{A} \quad \langle s, u\rangle :: \mathsf{D}$$

Non-concatenative morphology

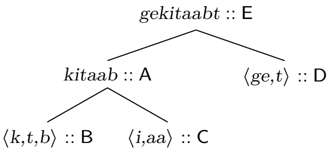
$$stuvw :: \mathsf{A} \quad \rightarrow \quad \langle s, u, w \rangle :: \mathsf{B} \quad \langle t, v \rangle :: \mathsf{C}$$

$$stu :: \mathsf{E} \quad \rightarrow \quad t :: \mathsf{A} \quad \langle s, u \rangle :: \mathsf{D}$$

Non-concatenative morphology

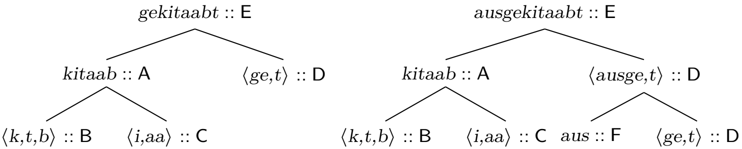
$$stuvw :: \mathsf{A} \quad \to \quad \langle s, u, w \rangle :: \mathsf{B} \quad \langle t, v \rangle :: \mathsf{C}$$

$$stu :: \mathsf{E} \quad \to \quad t :: \mathsf{A} \quad \langle s, u \rangle :: \mathsf{D}$$



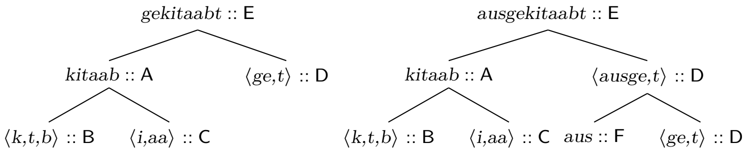
Non-concatenative morphology

$$\begin{aligned} stuvw :: \mathsf{A} &\quad\to\quad \langle s, u, w\rangle :: \mathsf{B} \quad \langle t, v\rangle :: \mathsf{C} \\ stu :: \mathsf{E} &\quad\to\quad t :: \mathsf{A} \quad \langle s, u\rangle :: \mathsf{D} \\ \langle ts, u\rangle :: \mathsf{D} &\quad\to\quad t :: \mathsf{F} \quad \langle s, u\rangle :: \mathsf{D} \end{aligned}$$



Non-concatenative morphology

$$
\begin{array}{rcl}
stuvw :: \mathsf{A} & \rightarrow & \langle s, u, w \rangle :: \mathsf{B} \quad \langle t, v \rangle :: \mathsf{C} \\
stu :: \mathsf{E} & \rightarrow & t :: \mathsf{A} \quad \langle s, u \rangle :: \mathsf{D} \\
\langle ts, u \rangle :: \mathsf{D} & \rightarrow & t :: \mathsf{F} \quad \langle s, u \rangle :: \mathsf{D}
\end{array}
$$



If our goal is to characterize the array of well-formed/derivable objects — not to pronounce them — then all we care about is "what's built out of what":

$$
\begin{array}{rcl}
\mathsf{A} & \rightarrow & \mathsf{B} \quad \mathsf{C} \\
\mathsf{E} & \rightarrow & \mathsf{A} \quad \mathsf{D} \\
\mathsf{D} & \rightarrow & \mathsf{F} \quad \mathsf{D}
\end{array}
$$

Outline

Multiple Context-Free Grammars (MCFGs)

$$st :: \mathrm{S} \quad \rightarrow \quad s :: \mathrm{NP} \quad t :: \mathrm{VP}$$

An MCFG generalises to allow yields to be *tuples of strings*.

$$t_2 s t_1 :: \mathrm{Q} \quad \rightarrow \quad s :: \mathrm{NP} \quad \langle t_1, t_2 \rangle :: \mathrm{VPWH}$$

This rule says two things:

- We can combine an NP with a VPWH to make a Q.
- The yield of the Q is $t_2 s t_1$,
  where s is the yield of the NP and $\langle t_1, t_2 \rangle$ is the yield of the VPWH.

Multiple Context-Free Grammars (MCFGs)

$$st :: \mathsf{S} \quad \rightarrow \quad s :: \mathsf{NP} \quad t :: \mathsf{VP}$$

An MCFG generalises to allow yields to be *tuples of strings*.

$$t_2 s t_1 :: \mathsf{Q} \quad \rightarrow \quad s :: \mathsf{NP} \quad \langle t_1, t_2 \rangle :: \mathsf{VPWH}$$

This rule says two things:

- We can combine an NP with a VPWH to make a Q.
- The yield of the Q is $t_2 s t_1$,
  where s is the yield of the NP and $\langle t_1, t_2 \rangle$ is the yield of the VPWH.

$$\textit{which girl the boy says is tall} :: \mathsf{Q} \quad \rightarrow$$

$$\textit{the boy} :: \mathsf{NP} \quad \langle \textit{says is tall}, \textit{which girl} \rangle :: \mathsf{VPWH}$$

Some technical details

- Each nonterminal has a rank n, and yields only n-tuples of strings.

  So given this rule:

  $$t_2 s t_1 :: \mathrm{Q} \quad \rightarrow \quad s :: \mathrm{NP} \quad \langle t_1, t_2 \rangle :: \mathrm{VPWH}$$

  we know that anything producing a VPWH must produce a 2-tuple.

  $$\langle \ldots, \ldots \rangle :: \mathrm{VPWH} \quad \rightarrow \quad \ldots$$

  and that anything producing an NP must produce a 1-tuple:

  $$\ldots :: \mathrm{NP} \quad \rightarrow \quad \ldots$$

(Seki et al. 1991, Weir 1988, Vijay-Shanker et al. 1987)

Some technical details

- Each nonterminal has a rank n, and yields only n-tuples of strings.

  So given this rule:

  $$t_2 s t_1 :: \mathrm{Q} \quad \rightarrow \quad s :: \mathrm{NP} \quad \langle t_1, t_2 \rangle :: \mathrm{VPWH}$$

  we know that anything producing a VPWH must produce a 2-tuple.

  $$\langle \dots, \dots \rangle :: \mathrm{VPWH} \quad \rightarrow \quad \dots$$

  and that anything producing an NP must produce a 1-tuple:

  $$\dots :: \mathrm{NP} \quad \rightarrow \quad \dots$$

- The string-composition functions cannot copy pieces of their arguments.

$$\begin{array}{rrcll}
\text{OK} & s\,t :: \mathrm{VP} & \rightarrow & s :: \mathrm{V} & t :: \mathrm{NP} \\
\text{OK} & t\,s\,\mathit{himself} :: \mathrm{S} & \rightarrow & s :: \mathrm{V} & t :: \mathrm{NP} \\
\text{Not OK} & t\,s\,t :: \mathrm{S} & \rightarrow & s :: \mathrm{V} & t :: \mathrm{NP}
\end{array}$$

(Seki et al. 1991, Weir 1988, Vijay-Shanker et al. 1987)

Some technical details

- Each nonterminal has a rank n, and yields only n-tuples of strings.

  So given this rule:

  $$t_2 s t_1 :: \mathrm{Q} \quad \rightarrow \quad s :: \mathrm{NP} \quad \langle t_1, t_2 \rangle :: \mathrm{VPWH}$$

  we know that anything producing a VPWH must produce a 2-tuple.

  $$\langle \ldots, \ldots \rangle :: \mathrm{VPWH} \quad \rightarrow \quad \ldots$$

  and that anything producing an NP must produce a 1-tuple:

  $$\ldots :: \mathrm{NP} \quad \rightarrow \quad \ldots$$

- The string-composition functions cannot copy pieces of their arguments.

OK	$s\,t :: \mathrm{VP}$	$\rightarrow$	$s :: \mathrm{V}$	$t :: \mathrm{NP}$
OK	$t\,s\,\mathit{himself} :: \mathrm{S}$	$\rightarrow$	$s :: \mathrm{V}$	$t :: \mathrm{NP}$
Not OK	$t\,s\,t :: \mathrm{S}$	$\rightarrow$	$s :: \mathrm{V}$	$t :: \mathrm{NP}$

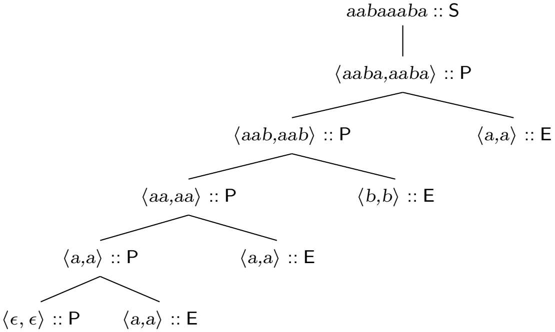
- Essentially equivalent to linear context-free rewriting systems (LCFRSs).

(Seki et al. 1991, Weir 1988, Vijay-Shanker et al. 1987)

Beyond context-free

$$
\begin{array}{rcl}
t_1 t_2 :: \mathsf{S} & \rightarrow & \langle t_1, t_2 \rangle :: \mathsf{P} \\
\langle t_1 u_1, t_2 u_2 \rangle :: \mathsf{P} & \rightarrow & \langle t_1, t_2 \rangle :: \mathsf{P} \quad \langle u_1, u_2 \rangle :: \mathsf{E} \\
\langle \epsilon, \epsilon \rangle :: \mathsf{P} & & \\
\langle a, a \rangle :: \mathsf{E} & & \\
\langle b, b \rangle :: \mathsf{E} & &
\end{array}
$$

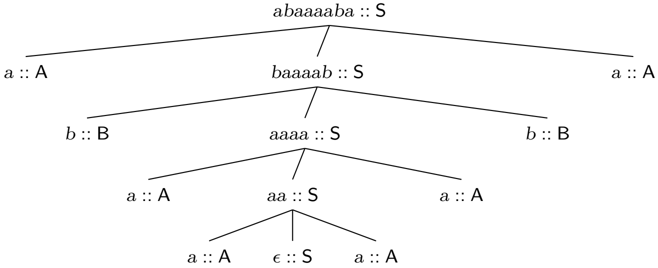
$$\left\{ ww \mid w \in \{a, b\}^* \right\}$$



Unlike in a CFG, we can ensure that the two "halves" are extended in the same ways without concatenating them together.

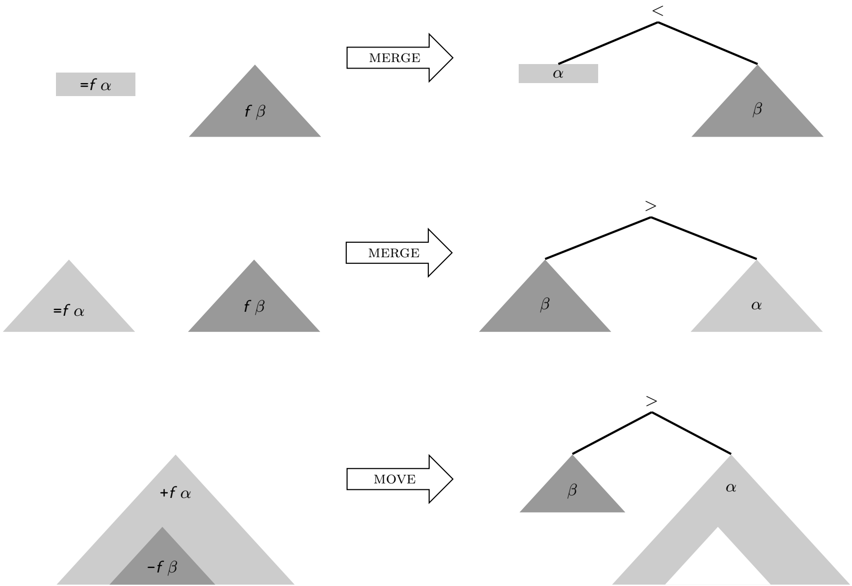
For comparison

$$
\begin{array}{rcl}
t_1 s t_2 :: \mathsf{S} & \rightarrow & t_1 :: \mathsf{A} \quad s :: \mathsf{S} \quad t_2 :: \mathsf{A} \\
t_1 s t_2 :: \mathsf{S} & \rightarrow & t_1 :: \mathsf{B} \quad s :: \mathsf{S} \quad t_2 :: \mathsf{B} \\
\epsilon :: \mathsf{S} & & \\
a :: \mathsf{A} & & \\
b :: \mathsf{B} & &
\end{array}
$$



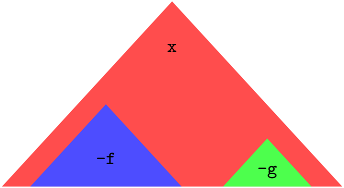
Outline

Merge and move



What matters in a (derived) tree

This tree:



becomes a tuple of categorized strings:

$$\left\langle \; \boxed{s :: \mathtt{x}} \;,\; \boxed{t :: \mathtt{-f}} \;,\; \boxed{u :: \mathtt{-g}} \; \right\rangle_0$$

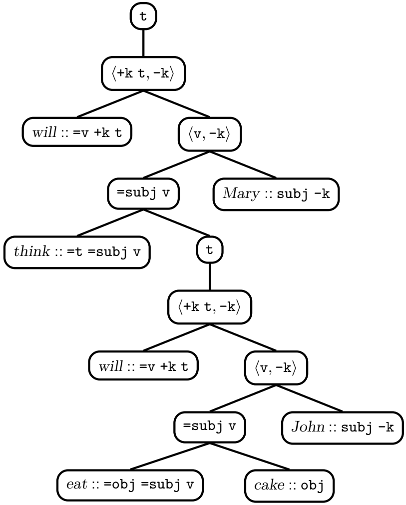
or, equivalently, a tuple-of-strings, categorized by a tuple-of-categories:

$$\langle s, t, u \rangle :: \langle \mathtt{x}, \mathtt{-f}, \mathtt{-g} \rangle_0$$

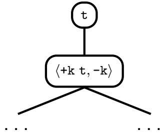
(Michaelis 2001, Stabler and Keenan 2003)

Remember MG derivation trees?

- We can tell that this tree represents a well-formed derivation, by checking the feature-manipulations at each step.
- How can we work out which string it derives?
    - Build up a tree according to merge and move rules, and read off leaves of the tree.
    - But there's a simpler way.



Producing a string from a derivation tree

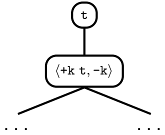


What do we need to have computed at the ⟨`+k t`, `-k`⟩ node, in order to compute the final string

Mary will think John will eat cake

at the `t` node?

Producing a string from a derivation tree

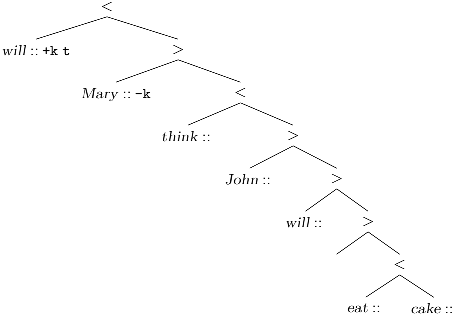


What do we need to have computed at the ⟨`+k t`, `-k`⟩ node, in order to compute the final string

Mary will think John will eat cake

at the `t` node?

This tree would do:

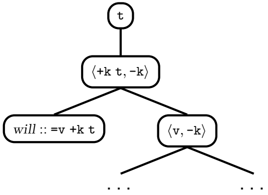


But all we actually need to know is:

- What's the string corresponding to the part that's going to move to check `-k`?
- What's the string corresponding to the leftovers?

These questions are answered by the tuple ⟨*will think John will eat cake*, *Mary*⟩

Producing a string from a derivation tree

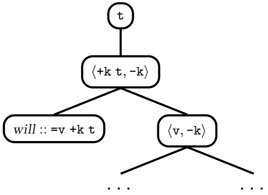


What do we need to have computed at the $\langle \texttt{v}, \texttt{-k} \rangle$ node, in order to compute the desired tuple

$$\langle \textit{will think John will eat cake}, \textit{Mary} \rangle$$

at the $\langle \texttt{+k t}, \texttt{-k} \rangle$ node?

Producing a string from a derivation tree

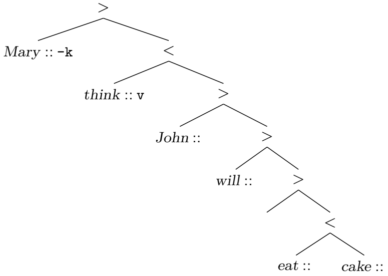


What do we need to have computed at the $\langle \texttt{v}, \texttt{-k} \rangle$ node, in order to compute the desired tuple

$$\langle \textit{will think John will eat cake}, \textit{Mary} \rangle$$

at the $\langle \texttt{+k t}, \texttt{-k} \rangle$ node?

This tree would do:

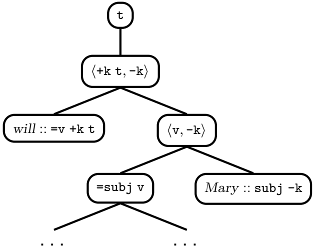


But all we actually need to know is:

- What's the string corresponding to the part that's going to move to check `-k`?
- What's the string corresponding to the leftovers?

These questions are answered by the tuple $\langle \textit{think John will eat cake}, \textit{Mary} \rangle$

Producing a string from a derivation tree

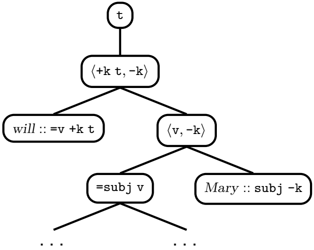


What do we need to have computed at the `=subj v` node, in order to compute the desired tuple

$$\langle \textit{think John will eat cake, Mary} \rangle$$

at the $\langle \texttt{v}, \texttt{-k} \rangle$ node?

Producing a string from a derivation tree

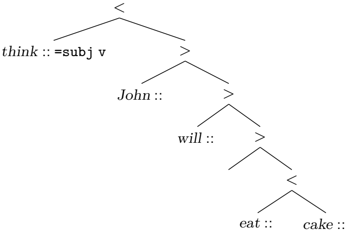


What do we need to have computed at the =subj v node, in order to compute the desired tuple

$$\langle \textit{think John will eat cake, Mary} \rangle$$

at the ⟨v, -k⟩ node?

This tree would do:



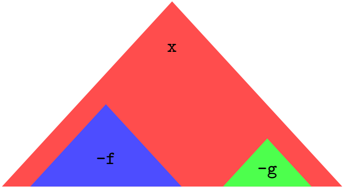
But all we actually need to know is:

- What's the string corresponding to the entire tree? (The "leftovers after no movement".)

This question is answered by the string *think John will eat cake*

What matters in a (derived) tree

This tree:



becomes a tuple of categorized strings:

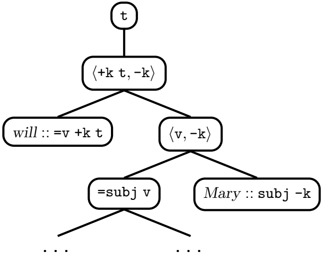
$$\left\langle \boxed{s :: \mathtt{x}} \;,\; \boxed{t :: \mathtt{-f}} \;,\; \boxed{u :: \mathtt{-g}} \right\rangle_0$$

or, equivalently, a tuple-of-strings, categorized by a tuple-of-categories:

$$\langle s, t, u \rangle :: \langle \mathtt{x}, \mathtt{-f}, \mathtt{-g} \rangle_0$$

(Michaelis 2001, Stabler and Keenan 2003)

MCFG rules



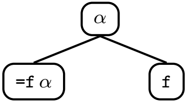
$$
\begin{array}{rcl}
t_2 t_1 :: \texttt{t} & \rightarrow & \langle t_1, t_2 \rangle :: \langle \texttt{+k t}, \texttt{-k} \rangle \\
\textit{Mary will think John will eat cake} :: \texttt{t} & \rightarrow & \langle \textit{will think John will eat cake, Mary} \rangle :: \langle \texttt{+k t}, \texttt{-k} \rangle
\end{array}
$$

$$
\begin{array}{rcl}
\langle s t_1, t_2 \rangle :: \langle \texttt{+k t}, \texttt{-k} \rangle & \rightarrow & s :: \texttt{=v +k t} \quad \langle t_1, t_2 \rangle :: \langle \texttt{v}, \texttt{-k} \rangle \\
\langle \textit{will think John will eat cake, Mary} \rangle :: \langle \texttt{+k t}, \texttt{-k} \rangle & \rightarrow & \textit{will} :: \texttt{=v +k t} \quad \langle \textit{think John will eat cake, Mary} \rangle :: \langle \texttt{v}, \texttt{-k} \rangle
\end{array}
$$

$$
\begin{array}{rcl}
\langle s, t \rangle :: \langle \texttt{v}, \texttt{-k} \rangle & \rightarrow & s :: \texttt{=subj v} \quad t :: \texttt{subj -k} \\
\langle \textit{think John will eat cake, Mary} \rangle :: \langle \texttt{v}, \texttt{-k} \rangle & \rightarrow & \textit{think John will eat cake} :: \texttt{=subj v} \quad \textit{Mary} :: \texttt{subj -k}
\end{array}
$$

One slightly annoying wrinkle

We know that this is a valid derivational step:



What is the corresponding MCFG rule?

Selected thing on the right?

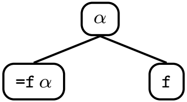
$$st :: \alpha \quad \rightarrow \quad s :: \texttt{=f}\ \alpha \quad t :: \texttt{f}$$

Selected thing on the left?

$$ts :: \alpha \quad \rightarrow \quad s :: \texttt{=f}\ \alpha \quad t :: \texttt{f}$$

One slightly annoying wrinkle

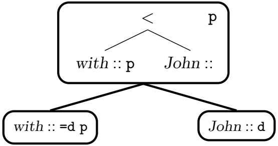
We know that this is a valid derivational step:



What is the corresponding MCFG rule?

Selected thing on the right?

$$st :: \alpha \quad \rightarrow \quad s :: \texttt{=f}\ \alpha \quad t :: \texttt{f}$$

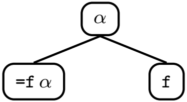


Selected thing on the left?

$$ts :: \alpha \quad \rightarrow \quad s :: \texttt{=f}\ \alpha \quad t :: \texttt{f}$$

One slightly annoying wrinkle

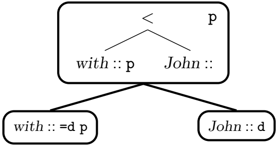
We know that this is a valid derivational step:



What is the corresponding MCFG rule?

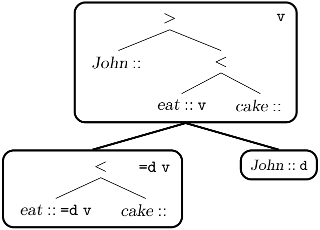
Selected thing on the right?

$$st :: \alpha \quad \rightarrow \quad s :: \texttt{=f}\ \alpha \quad t :: \texttt{f}$$



Selected thing on the left?

$$ts :: \alpha \quad \rightarrow \quad s :: \texttt{=f}\ \alpha \quad t :: \texttt{f}$$



One slightly annoying wrinkle

Each type needs to record not only the unchecked features, but also whether the expression is lexical.

I'll write lexical types as $\langle \ldots \rangle_1$ and non-lexical types as $\langle \ldots \rangle_0$.

So types of the form $\langle \texttt{=f}\ \alpha \rangle_1$ act slightly differently from those of the form $\langle \texttt{=f}\ \alpha \rangle_0$.

$$\begin{array}{rcll} st :: \langle \alpha \rangle_0 & \to & s :: \langle \texttt{=f}\ \alpha \rangle_1 & t :: \langle \texttt{f} \rangle_n \\ \textit{with John} :: \langle \texttt{p} \rangle_0 & \to & \textit{with} :: \langle \texttt{=d p} \rangle_1 & \textit{John} :: \langle \texttt{d} \rangle_1 \end{array}$$

$$\begin{array}{rcll} ts :: \langle \alpha \rangle_0 & \to & s :: \langle \texttt{=f}\ \alpha \rangle_0 & t :: \langle \texttt{f} \rangle_n \\ \textit{John eat cake} :: \langle \texttt{v} \rangle_0 & \to & \textit{eat cake} :: \langle \texttt{=d v} \rangle_0 & \textit{John} :: \langle \texttt{d} \rangle_1 \end{array}$$

Context-free structure

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

Context-free structure

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

General schemas for MERGE steps (approximate):

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

General schemas for MOVE steps (approximate):

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

Context-free structure

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

General schemas for MERGE steps (approximate):

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

General schemas for MOVE steps (approximate):

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

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

Three schemas for MERGE rules:

$$\langle st, t_1, \ldots, t_k \rangle :: \langle \gamma, \alpha_1, \ldots, \alpha_k \rangle_0 \quad \rightarrow \quad s :: \langle \texttt{=f}\gamma \rangle_1 \quad \langle t, t_1, \ldots, t_k \rangle :: \langle \texttt{f}, \alpha_1, \ldots, \alpha_k \rangle_n$$

$$\langle ts, s_1, \ldots, s_j, t_1, \ldots, t_k \rangle :: \langle \gamma, \alpha_1, \ldots, \alpha_j, \beta_1, \ldots, \beta_k \rangle_0 \quad \rightarrow \quad \langle s, s_1, \ldots, s_j \rangle :: \langle \texttt{=f}\gamma, \alpha_1, \ldots, \alpha_j \rangle_0 \quad \langle t, t_1, \ldots, t_k \rangle :: \langle \texttt{f}, \beta_1, \ldots, \beta_k \rangle_n$$

$$\langle s, s_1, \ldots, s_j, t, t_1, \ldots, t_k \rangle :: \langle \gamma, \alpha_1, \ldots, \alpha_j, \delta, \beta_1, \ldots, \beta_k \rangle_0 \quad \rightarrow \quad \langle s, s_1, \ldots, s_j \rangle :: \langle \texttt{=f}\gamma, \alpha_1, \ldots, \alpha_j \rangle_n \quad \langle t, t_1, \ldots, t_k \rangle :: \langle \texttt{f}\delta, \beta_1, \ldots, \beta_k \rangle_{n'}$$

Two schemas for MOVE rules:

$$\langle s_i s, s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_k \rangle :: \langle \gamma, \alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k \rangle_0 \quad \rightarrow \quad \langle s, s_1, \ldots, s_i, \ldots, s_k \rangle :: \langle \texttt{+f}\gamma, \alpha_1, \ldots, \alpha_{i-1}, \texttt{-f}, \alpha_{i+1}, \ldots, \alpha_k \rangle_0$$

$$\langle s, s_1, \ldots, s_i, \ldots, s_k \rangle :: \langle \gamma, \alpha_1, \ldots, \alpha_{i-1}, \delta, \alpha_{i+1}, \ldots, \alpha_k \rangle_0 \quad \rightarrow \quad \langle s, s_1, \ldots, s_i, \ldots, s_k \rangle :: \langle \texttt{+f}\gamma, \alpha_1, \ldots, \alpha_{i-1}, \texttt{-f}\delta, \alpha_{i+1}, \ldots, \alpha_k \rangle_0$$

References I

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.