Higher Order Structures in Minimalist Derivations

Greg Kobele

TAG+13

Universität Leipzig

Intro

Intro

Grammar formalisms, like programming languages, are useful because

> *they allow us to factor our explanation of linguistic behaviour into a statement of abstract regularities (the grammar), and a description of how these are computed online (the parser/parser-generator)*

Intro

Current MG parsing algorithms

- needlessly explode state space (making beam search implausible)
- are based on (exponentially less succinct) MCFGs
- have only extrema on GLC lattice (inherited from MCFG)

Intro

We exploit the structure of MGs
to define a MG-specific TD parsing strategy

- structures search space by 'sharing' infinite classes of items
- bringing us closer to LC

This gives a formal (very literal) reconstruction of popular psycholinguistic ideas about the human sentence processing mechanism

MGs

Overview

a formalization of Chomsky's "minimalist program"

- I think they are an *exact* formalization
- I am interested in them because they are a bridge between linguistics and computer science

Properties

MGs belong to family of MCS grammar formalisms

- TAG is Monadic CFTG, and MG is (contained in) MRTG
  - Share the regularity of derivation trees
- TALs are all *well-nested* MCFLs, but MLs are the non-well-nested MCFLs
  separation: (Kanazawa & Salvati, 2010)

$$\{w\#w \mid w \in L,\ L \text{ is in } CFL - EDT0L\}$$

well-nested MCFLs can have crossing dependencies, but not between syntactically complicated objects

Minimalist Grammars

- To specify a grammar, we need to specify two things:
  1. The features
     (which features we will use in our grammar)
  2. The lexicon
     (which syntactic feature sequences are assigned to which words)

Features

Features come in pairs

- `=x` and `x`
- `+y` and `-y`

Like in CG, categories are *structured*

- list of features

tradition calls categories: *feature bundles*

`=n.d.-k`

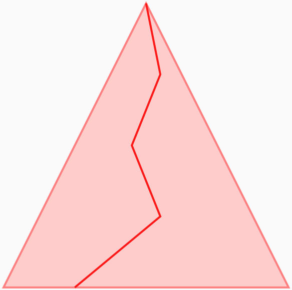
Data structure

Binary branching trees

- internal node labels: > and <
- leaf labels: (w, δ) and `t`

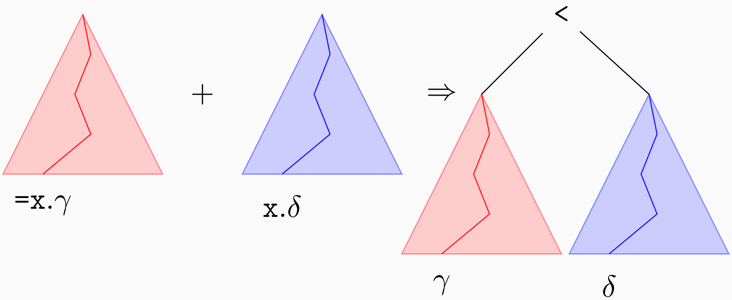
Headed trees

```
head( >(u,v) ) = head( v )
head( <(u,v) ) = head( u )
head( l ) = l
```

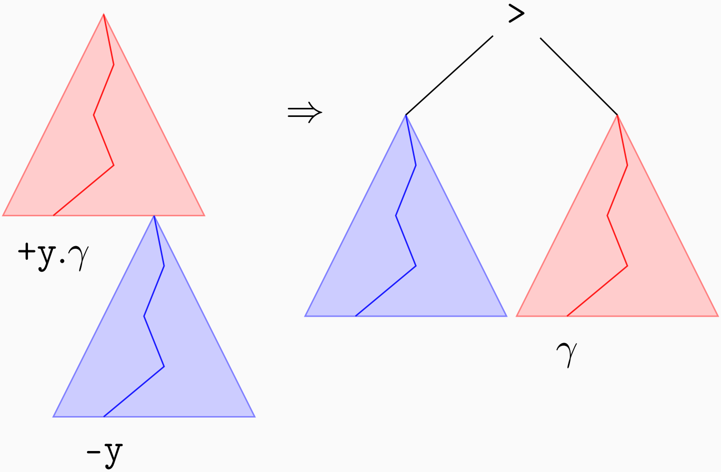


δ

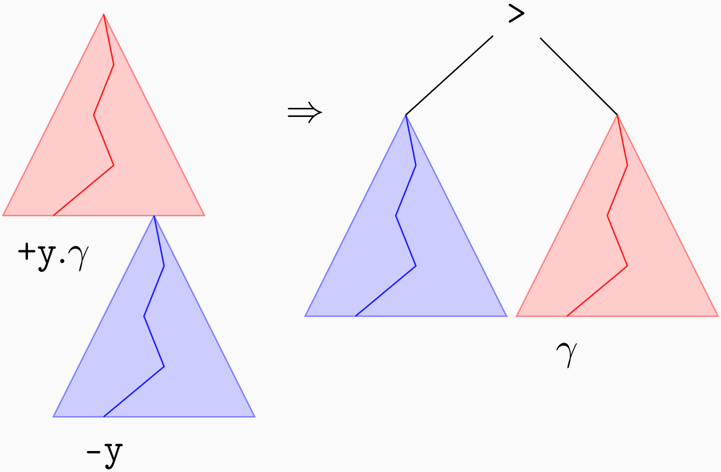
Merge



Move



Move



SMC
No other possible mover

A working example

boy	`n`
every	`=n.d.-k`
laugh	`=d.v`
will	`=v.+k.s`

Representing derivations

Representing derivations

1. `select` *every*

every

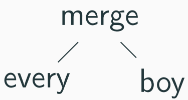
Representing derivations

1. `select` *every*
2. `select` *boy*

every boy

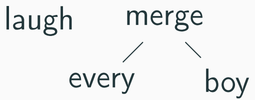
Representing derivations

1. `select` *every*
2. `select` *boy*
3. `merge` *1* and *2*
   $[_{DP}$ every $[_{NP}$ boy]]



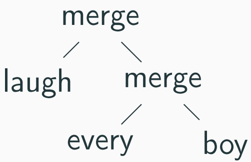
Representing derivations

1. `select` *every*
2. `select` *boy*
3. `merge` *1* and *2*
   $[_{DP}\ \text{every}\ [_{NP}\ \text{boy}\ ]]$
4. `select` *laugh*



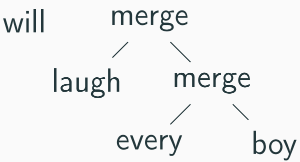
Representing derivations

1. `select` *every*
2. `select` *boy*
3. `merge` *1* and *2*
   $[_{DP}$ every $[_{NP}$ boy $]]$
4. `select` *laugh*
5. `merge` *4* and *3*
   $[_{VP}$ laugh $[_{DP}$ every boy $]]$



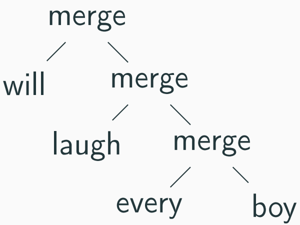
Representing derivations

1. `select` *every*
2. `select` *boy*
3. `merge` *1* and *2*
   $[_{DP}\ \text{every}\ [_{NP}\ \text{boy}\ ]]$
4. `select` *laugh*
5. `merge` *4* and *3*
   $[_{VP}\ \text{laugh}\ [_{DP}\ \text{every boy}\ ]]$
6. `select` *will*



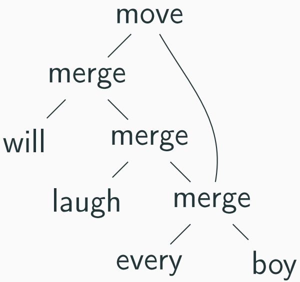
Representing derivations

1. `select` *every*
2. `select` *boy*
3. `merge` *1* and *2*
   $[_{DP}$ every $[_{NP}$ boy $]]$
4. `select` *laugh*
5. `merge` *4* and *3*
   $[_{VP}$ laugh $[_{DP}$ every boy $]]$
6. `select` *will*
7. `merge` *6* and *5*
   $[_{IP}$ will $[_{VP}$ laugh $[_{DP}$ every boy $]]]$

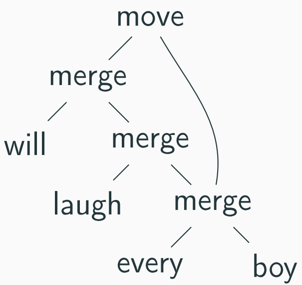


Representing derivations

1. `select` *every*
2. `select` *boy*
3. `merge` *1* and *2*
   $[_{DP}$ every $[_{NP}$ boy $]]$
4. `select` *laugh*
5. `merge` *4* and *3*
   $[_{VP}$ laugh $[_{DP}$ every boy $]]$
6. `select` *will*
7. `merge` *6* and *5*
   $[_{IP}$ will $[_{VP}$ laugh $[_{DP}$ every boy $]]]$
8. `move` *every boy*
   $[_{IP}[_{DP}$ every boy $][_{I'}$ will $[_{VP}$ laugh $t]]]$



The determinacy of movement



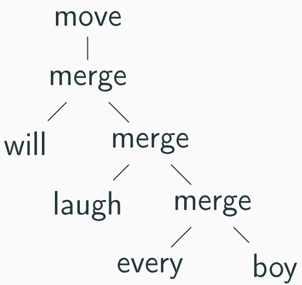
Attract Closest

Minimal Link

Shortest Move

SMC
can only be 1 thing moving for a particular reason at any time

The determinacy of movement



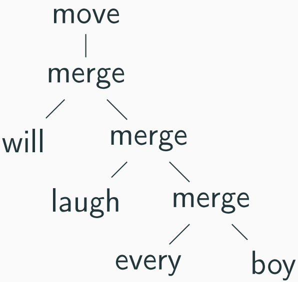
Attract Closest

Minimal Link

Shortest Move

SMC
can only be 1 thing moving for a particular reason at any time

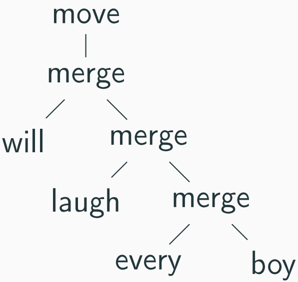
The determinacy of movement



The proof objects of minimalism

- are *first order* (i.e. trees)

The determinacy of movement



The proof objects of minimalism

- are *first order* (i.e. trees)
- the proofs of any proposition (e.g. S) form a *regular tree language*

Towards MCFGs (I.)

- a *categorized string* is a pair $\phi = (u, \delta)$, where

  $\mathbf{u}$ is a string

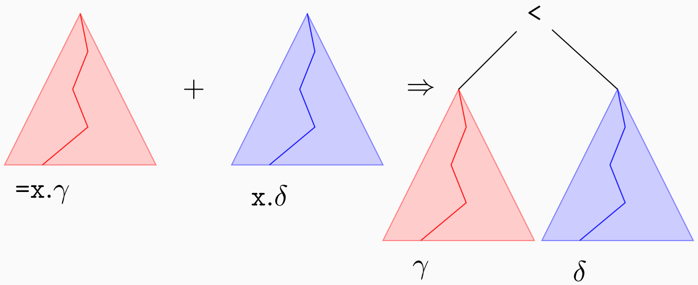
  δ is a feature bundle

- an *expression* is a finite sequence of categorized strings

$$\phi_0, \ldots, \phi_n$$

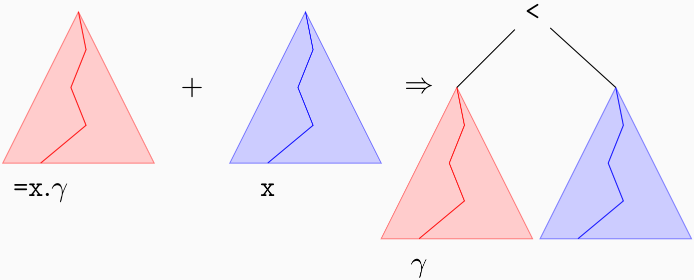
  - each $\phi_i, 1 \leq i \leq n$ represents a moving subtree
  - ϕ_0 represents the rest of the tree

Towards MCFGs (II.)



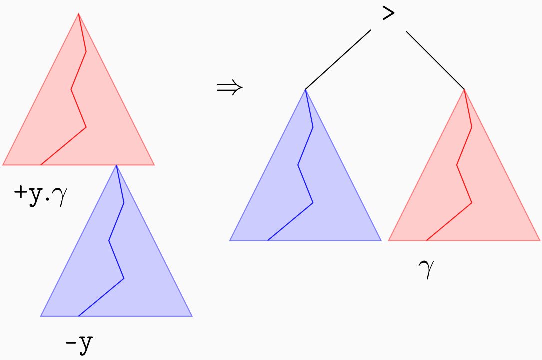
$$\frac{(u, \texttt{=x}.\gamma), \phi_1, \ldots, \phi_m \qquad (v, \texttt{x}.\delta), \psi_1, \ldots, \psi_n}{(u, \gamma), \phi_1, \ldots, \phi_m, (v, \delta), \psi_1, \ldots, \psi_n}$$

Towards MCFGs (II.)



$$\frac{(u, \texttt{=x}.\gamma), \phi_1, \ldots, \phi_m \qquad (v, \texttt{x}), \psi_1, \ldots, \psi_n}{(uv, \gamma), \phi_1, \ldots, \phi_m, \psi_1, \ldots, \psi_n}$$

Towards MCFGs (III.)



$$\frac{(u, \texttt{+y}.\gamma), \phi_1, \ldots, \phi_{j-1}, (v, \texttt{-y}), \phi_{j+1}, \ldots, \phi_m}{(vu, \gamma), \phi_1, \ldots, \phi_{j-1}, \phi_{j+1}, \ldots, \phi_m}$$

Automata

An rule like:

$$\frac{(u, \texttt{+y}.\gamma), \phi_1, \ldots, \phi_{j-1}, (v, \texttt{-y}), \phi_{j+1}, \ldots, \phi_m}{(vu, \gamma), \phi_1, \ldots, \phi_{j-1}, \phi_{j+1}, \ldots, \phi_m}$$

gives us an *ldmbutts (tree-to-string)* production:

$$\begin{aligned} &\mathrm{move}(q(u, v_1, \ldots, v_{j-1}, v, v_{j+1}, \ldots, v_m)) \\ &\qquad \to q'(vu, v_1, \ldots, v_{j-1}, v_{j+1}, \ldots, v_m) \end{aligned}$$

where

$$\begin{aligned} q &= \langle \texttt{+y}.\gamma, \delta_1, \ldots, \delta_{j-1}, \texttt{-y}, \delta_j, \ldots, \delta_m \rangle \\ q' &= \langle \gamma, \delta_1, \ldots, \delta_{j-1}, \delta_j, \ldots, \delta_m \rangle \end{aligned}$$

An example

An example

every

$\big(\text{every}, = \texttt{n.d.-k}\big)$

An example

every boy

(every, $=\mathtt{n}.\mathtt{d}.\mathtt{-k}$) (boy, $\mathtt{n}$)

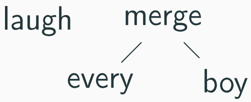
An example

merge

every boy

$$\frac{(\text{every}, \mathtt{=n.d.-k}) \qquad (\text{boy}, \mathtt{n})}{(\text{every boy}, \mathtt{d.-k})}$$

An example



$$(\text{laugh}, =\mathtt{d}.\mathtt{v}) \qquad \frac{(\text{every}, =\mathtt{n}.\mathtt{d}.\text{-}\mathtt{k}) \qquad (\text{boy}, \mathtt{n})}{(\text{every boy}, \mathtt{d}.\text{-}\mathtt{k})}$$

An example

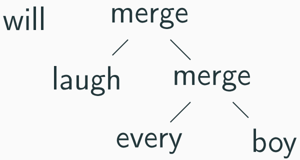
merge

laugh merge

every boy

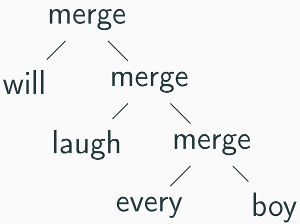
$$\dfrac{(\text{laugh}, \texttt{=d.v}) \qquad \dfrac{(\text{every}, \texttt{=n.d.-k}) \qquad (\text{boy}, \texttt{n})}{(\text{every boy}, \texttt{d.-k})}}{(\text{laugh}, \texttt{v}), (\text{every boy}, \texttt{-k})}$$

An example



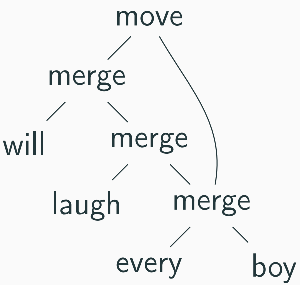
$$(\text{will}, \texttt{=v.+k.s}) \qquad \dfrac{(\text{laugh}, \texttt{=d.v}) \qquad \dfrac{(\text{every}, \texttt{=n.d.-k}) \qquad (\text{boy}, \texttt{n})}{(\text{every boy}, \texttt{d.-k})}}{(\text{laugh}, \texttt{v}), (\text{every boy}, \texttt{-k})}$$

An example



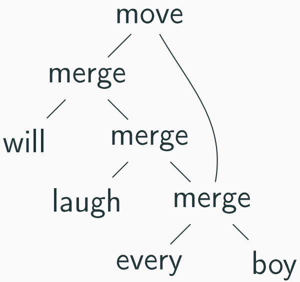
$$\dfrac{(\text{will}, \texttt{=v.+k.s}) \qquad \dfrac{(\text{laugh}, \texttt{=d.v}) \qquad \dfrac{(\text{every}, \texttt{=n.d.-k}) \qquad (\text{boy}, \texttt{n})}{(\text{every boy}, \texttt{d.-k})}}{(\text{laugh}, \texttt{v}), (\text{every boy}, \texttt{-k})}}{(\text{will laugh}, \texttt{+k.s}), (\text{every boy}, \texttt{-k})}$$

An example



$$\dfrac{\dfrac{(\text{will}, \texttt{=v.+k.s}) \qquad \dfrac{(\text{laugh}, \texttt{=d.v}) \qquad \dfrac{(\text{every}, \texttt{=n.d.-k}) \qquad (\text{boy}, \texttt{n})}{(\text{every boy}, \texttt{d.-k})}}{(\text{laugh}, \texttt{v}), (\text{every boy}, \texttt{-k})}}{(\text{will laugh}, \texttt{+k.s}), (\text{every boy}, \texttt{-k})}}{(\text{every boy will laugh}, \texttt{s})}$$

An example

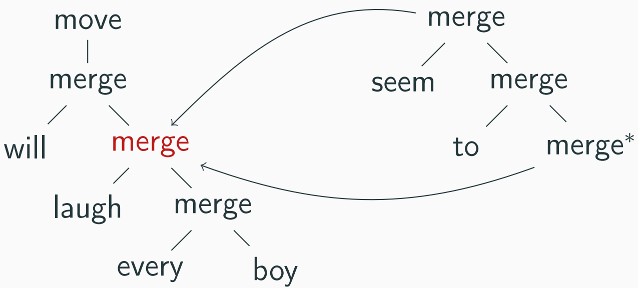


$$\dfrac{(\text{will}, \texttt{=v.+k.s}) \qquad \dfrac{(\text{laugh}, \texttt{=d.v}) \qquad \dfrac{(\text{every}, \texttt{=n.d.-k}) \qquad (\text{boy}, \texttt{n})}{(\text{every boy}, \texttt{d.-k})}}{(\text{laugh}, \texttt{v}), (\text{every boy}, \texttt{-k})}}{\dfrac{(\text{will laugh}, \texttt{+k.s}), (\text{every boy}, \texttt{-k})}{(\text{every boy will laugh}, \texttt{s})}}$$

A slightly larger example

boy	`n`
every	`=n.d.-k`
laugh	`=d.v`
will	`=v.+k.s`
to	`=v.i`
seem	`=i.v`

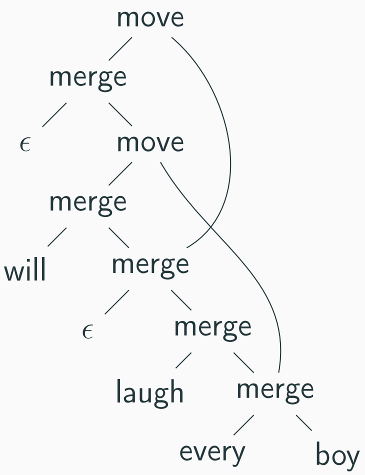
More derivations



Yoda

boy	`n`
every	`=n.d.-k`
laugh	`=d.v`
will	`=v.+k.s`
to	`=v.i`
seem	`=i.v`
ϵ	`=v.v.-top`
ϵ	`=s.+top.c`

Remnant movement



Parsing

Top-down parsing

Items represent cuts of derivation tree

$$\square$$

Top-down parsing

Items represent cuts of derivation tree

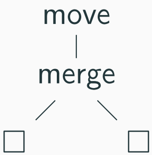
move

|

$\square$

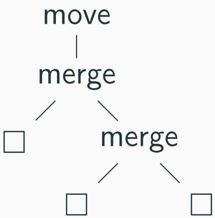
Top-down parsing

Items represent cuts of derivation tree



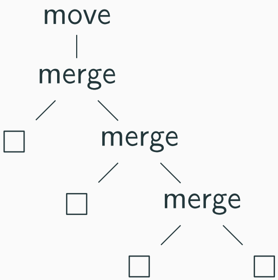
Top-down parsing

Items represent cuts of derivation tree



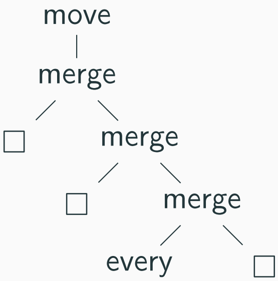
Top-down parsing

Items represent cuts of derivation tree



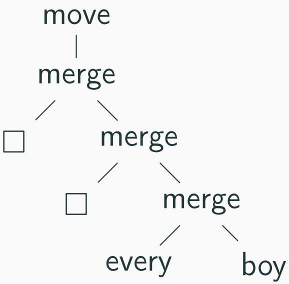
Top-down parsing

Items represent cuts of derivation tree



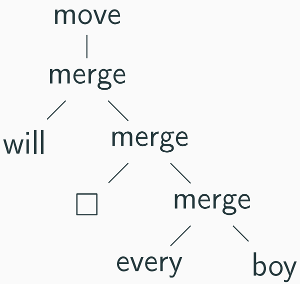
Top-down parsing

Items represent cuts of derivation tree



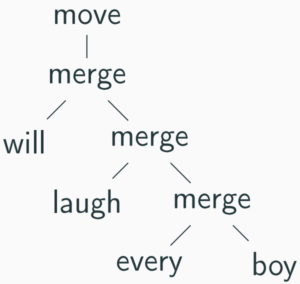
Top-down parsing

Items represent cuts of derivation tree



Top-down parsing

Items represent cuts of derivation tree



Local trees

this exploits:
MG derivation trees form a local set

$\boxed{\mathtt{S}}$

Local trees

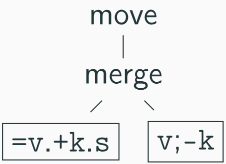
this exploits:
MG derivation trees form a local set

move

$\boxed{\texttt{+k.s;-k}}$

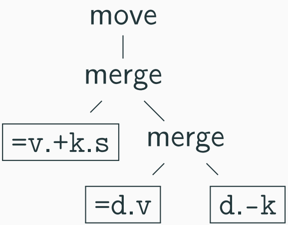
Local trees

this exploits:
MG derivation trees form a local set



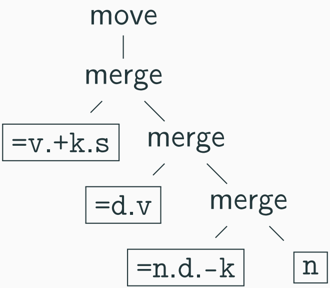
Local trees

this exploits:
MG derivation trees form a local set



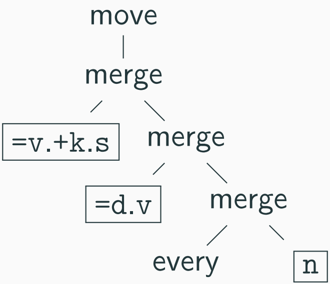
Local trees

this exploits:
MG derivation trees form a local set



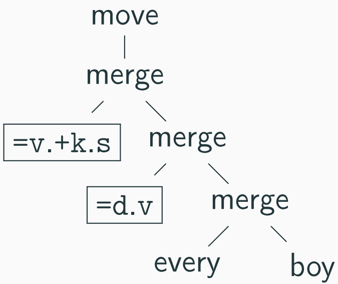
Local trees

this exploits:
MG derivation trees form a local set



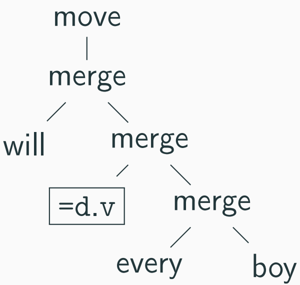
Local trees

this exploits:
MG derivation trees form a local set



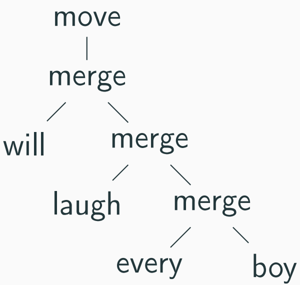
Local trees

this exploits:
MG derivation trees form a local set



Local trees

this exploits:
MG derivation trees form a local set



Undoing movement

- When we hypothesize a move node:

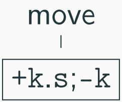
move

|

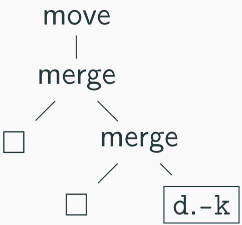
$\boxed{\texttt{+k.s;-k}}$

Undoing movement

- When we hypothesize a move node:

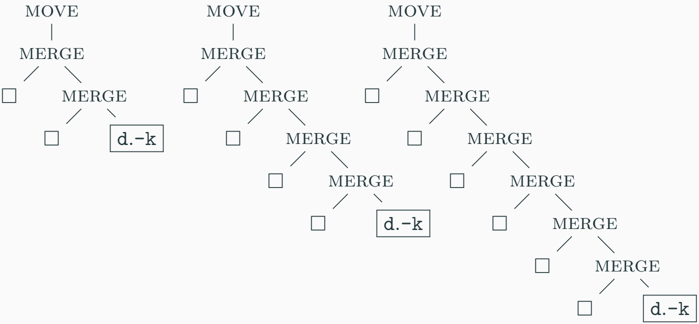


- We next must hypothesize where the mover is:

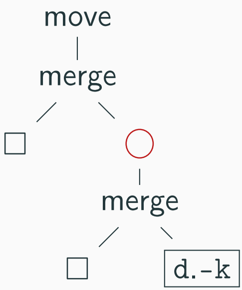


Appearances can be deceiving

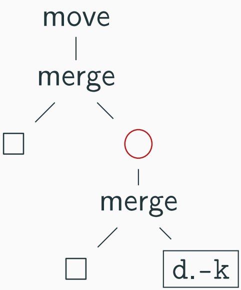
Every boy will (seem to) laugh*



If only. . .



If only. . .



- Might work in this case,
  - but is there a non-analysis specific principle?

Structure in derivations

MG derivations are *subregular* **(Graf)**
(Tier-based) strictly local

strict locality conjunction of negative literals
(with immediate successor)

tier-based relativized successors
($\lhd_T$, where $T \subseteq \Sigma$)

Example (strings)

Primary stress
$\lhd := \lhd_{\acute{\sigma}}$

Have primary stress $\neg(\$ \lhd \$)$

Have at most one stress $\neg(\acute{\sigma} \lhd \acute{\sigma})$

Example (trees)

Movement (Graf)
$\lhd := \lhd_{\texttt{+k},\texttt{-k}}$

Movers gonna move $\neg(\$ \lhd \ell)$

No movement without movement $\neg(\text{move} \lhd \$)$

No competition $\neg(\text{move} \lhd \ell_1, \ell_2)$

Argument structure via *n*-grams

Every lexical item ℓ appears in a derivation with a unique local context

- depends exclusively on positive feature sequence ($=\mathtt{x}$ and $+\mathtt{y}$)

$$(\text{will}, =\mathtt{v}.+\mathtt{k}.\mathtt{s})$$

Argument structure via *n*-grams

Every lexical item ℓ appears in a derivation with a unique local context

- depends exclusively on positive feature sequence (=x and +y)

merge

(will, =v.+k.s) □

Argument structure via *n*-grams

Every lexical item ℓ appears in a derivation with a unique local context

- depends exclusively on positive feature sequence ($=\mathrm{x}$ and $+\mathrm{y}$)

move

merge

(will, $=\mathrm{v}.+\mathrm{k}.\mathrm{s}$) $\square$

Exploiting regularities in derivations

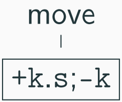
- When we hypothesize a move node:

move

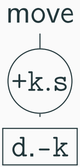
$\boxed{\texttt{+k.s;-k}}$

Exploiting regularities in derivations

- When we hypothesize a move node:



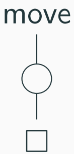
- We know it immediately dominates a mover (on the relevant tier):



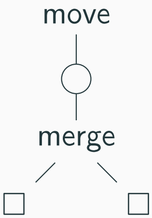
A sketch

□

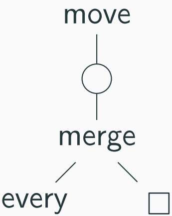
A sketch



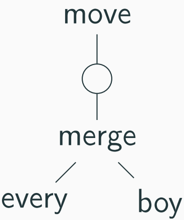
A sketch



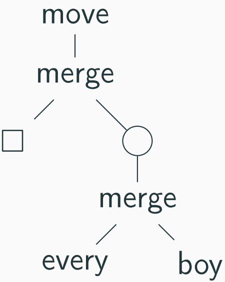
A sketch



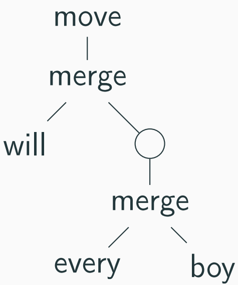
A sketch



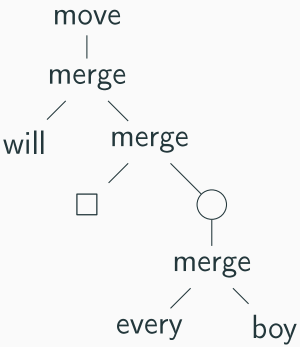
A sketch



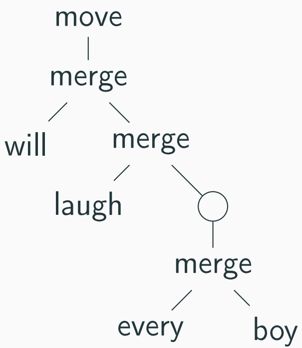
A sketch



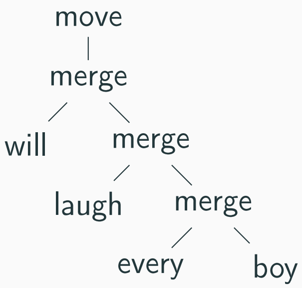
A sketch



A sketch



A sketch



A basic 'hole' data structure

$$\boxed{\alpha}^{g}$$
$$|$$
$$xs$$

- g is a gorn address
  where we are in the derived tree

```
data Hole t b x = Hole t [(b,x)]
```

A basic 'hole' data structure

$$\begin{array}{c} \boxed{\alpha}^{g} \\ | \\ xs \end{array}$$

- *g* is a gorn address
  where we are in the derived tree
- *xs* is a (finite) list of

```
data Hole t b x = Hole t [(b,x)]
```

A basic 'hole' data structure

$$\begin{array}{c}\boxed{\alpha}^{g}\\ | \\ \mathit{xs}\end{array}$$

- *g* is a gorn address
  where we are in the derived tree
- *xs* is a (finite) list of
  - derivations with holes
    elements in separate tiers

```
data Hole t b x = Hole t [(b,x)]
```

A basic 'hole' data structure

$$\boxed{\alpha}^{g}$$
$$|$$
$$xs$$

- g is a gorn address
  where we are in the derived tree
- xs is a (finite) list of
  - derivations with holes
    elements in separate tiers
  - ... paired with feature bundles
    information about the occupied tier

```
data Hole t b x = Hole t [(b,x)]
```

Unmerge1

- Given

$$\begin{array}{c}\boxed{\alpha : g} \\ | \\ xs\end{array}$$

Unmerge1

- Given

$$\boxed{\alpha : g}$$

$$|$$

$$\mathit{xs}$$

- merge could have applied

merge

$$/ \quad \backslash$$

$$\boxed{=\mathrm{x}.\alpha : g0} \quad \boxed{\mathrm{x} : g1}$$

$$| \qquad\qquad |$$

$$\mathit{us} \qquad\qquad \mathit{vs}$$

Unmerge1

- Given

$$\begin{array}{c} \boxed{\alpha : g} \\ | \\ \textit{xs} \end{array}$$

- merge could have applied

$$\begin{array}{ccc} & \text{merge} & \\ / & & \backslash \\ \boxed{=\text{x}.\alpha : g0} & & \boxed{\text{x} : g1} \\ | & & | \\ \textit{us} & & \textit{vs} \end{array}$$

Unmerge1

- Given

$$\boxed{\alpha : g}$$

xs

- merge could have applied

merge

$$\boxed{=\mathrm{x}.\alpha : g0} \quad \boxed{\mathrm{x} : g1}$$

us *vs*

```
xs = sort (us ++ vs)
```

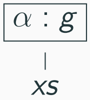
Unmove

- Given

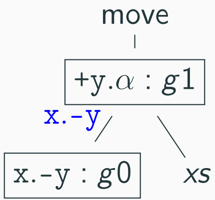
$$\begin{array}{c}\boxed{\alpha : g} \\ | \\ xs\end{array}$$

Unmove

- Given



- move could have applied



Unmerge2

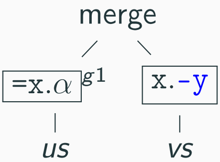
- Given

$$\begin{array}{c} \boxed{\alpha}^{g} \\ | \\ xs \end{array}$$

Unmerge2

- Given

- merge could have applied to a mover



Unmerge2

- Given

$$\boxed{\alpha}^{g}$$
$$|$$
$$xs$$

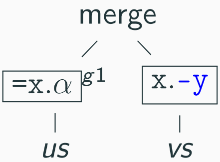
- merge could have applied to a mover

$$\text{merge}$$
$$\swarrow \qquad \searrow$$
$$\boxed{\texttt{=x.}\alpha}^{g1} \qquad \boxed{\texttt{x.-y}}$$
$$| \qquad\qquad |$$
$$us \qquad\qquad vs$$

Unmerge2

- Given

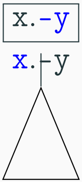
- merge could have applied to a mover




```
xs = sort (us ++ vs)
```

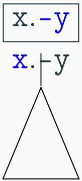
Completion (I)

- Given

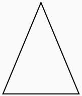


Completion (I)

- Given



- this is the tree you're looking for

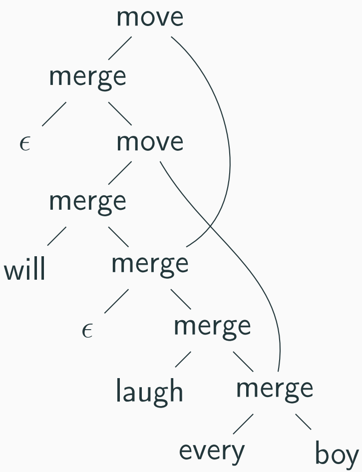


ATNs and filling gaps

- Psycholinguists
  - you process moved items (fillers)
  - and then you try to find where they moved from (gap)
- TD MG parsing

  to process filler, first find gap!
- Here
  - *unmove* constructs a filler
  - *unmerge2* constructs a gap
  - *complete* fills the gap

Remnant movement



boy	`n`
every	`=n.d.-k`
laugh	`=d.v`
will	`=v.+k.s`
ϵ	`=v.v.-top`
ϵ	`=s.+top.s`

Remnant movement

$\boxed{\text{s}}^{\epsilon}$

Remnant movement

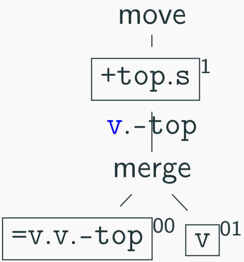
move

$\boxed{\texttt{+top.s}}^{1}$

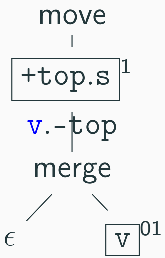
v.-top

$\boxed{\texttt{v.-top}}^{0}$

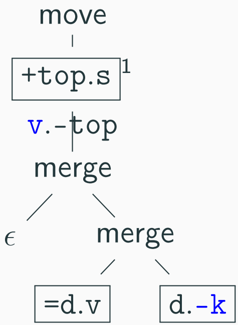
Remnant movement



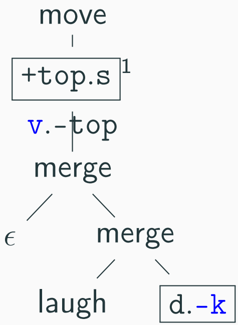
Remnant movement



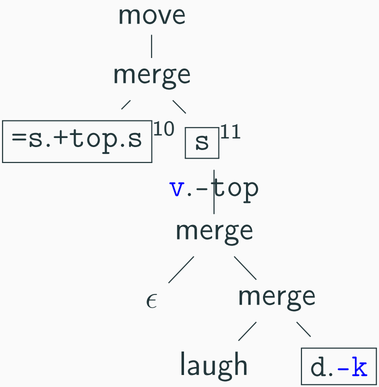
Remnant movement



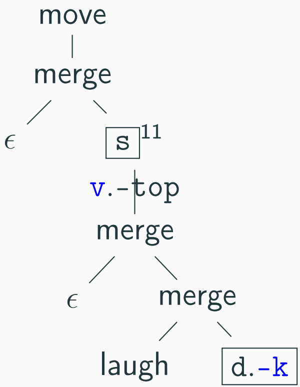
Remnant movement



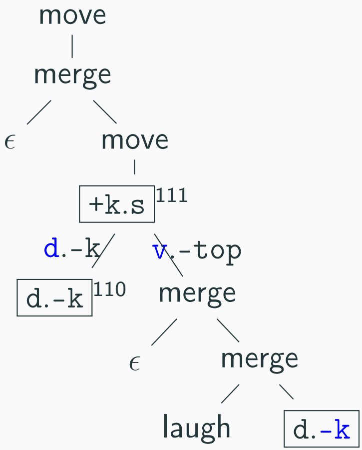
Remnant movement



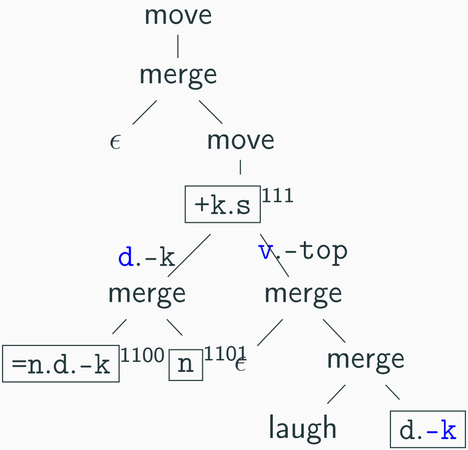
Remnant movement



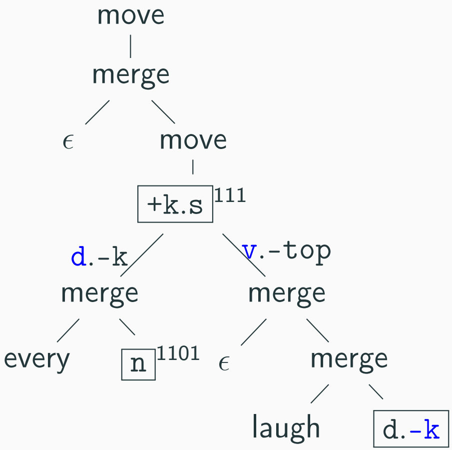
Remnant movement



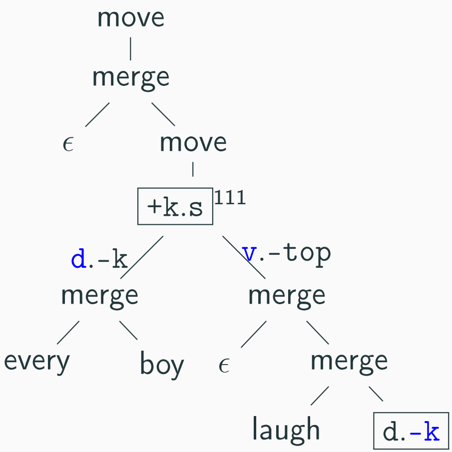
Remnant movement



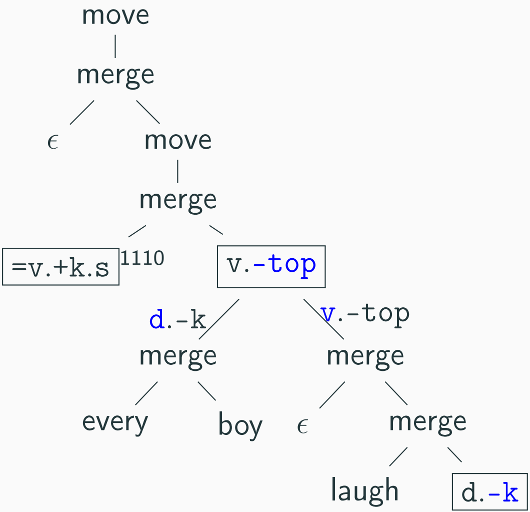
Remnant movement



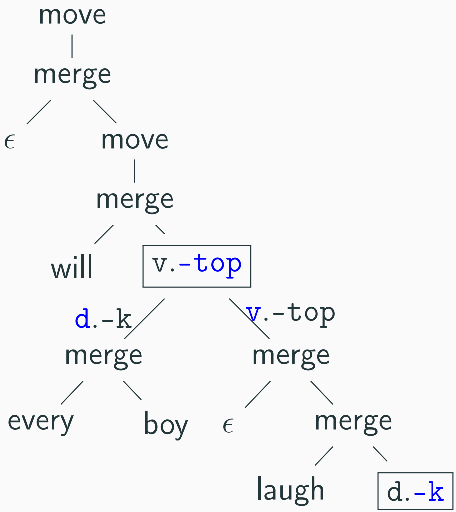
Remnant movement



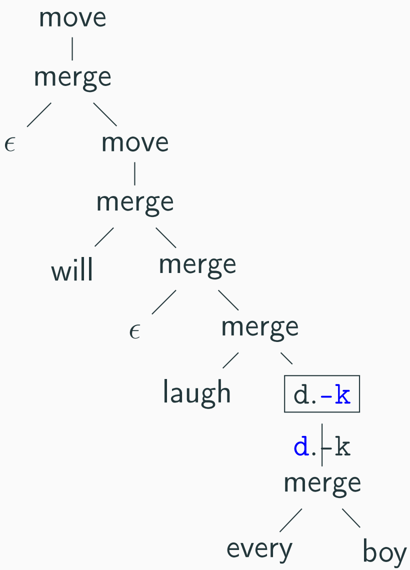
Remnant movement



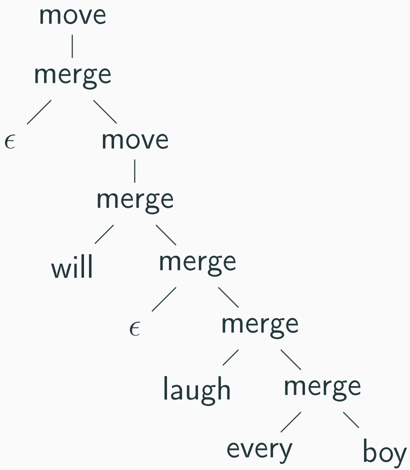
Remnant movement



Remnant movement



Remnant movement



Enforcing the SMC

Recall:

1. Movers gonna move : $\neg(\$ \lhd \ell)$
2. No movement without movement : $\neg(\textbf{move} \lhd \$)$
3. No competition : $\neg(\textbf{move} \lhd \ell_1, \ell_2)$

How are these being enforced?

Enforcing the SMC

Recall:

1. Movers gonna move : $\neg(\$ \lhd \ell)$
2. No movement without movement : $\neg(\mathbf{move} \lhd \$)$
3. No competition : $\neg(\mathbf{move} \lhd \ell_1, \ell_2)$

How are these being enforced?

1. Two ways of generating a mover:

Enforcing the SMC

Recall:

1. Movers gonna move : $\neg(\$ \lhd \ell)$
2. No movement without movement : $\neg(\textbf{move} \lhd \$)$
3. No competition : $\neg(\textbf{move} \lhd \ell_1, \ell_2)$

How are these being enforced?

1. Two ways of generating a mover:
   - via unmerge2 (i.e. a gap) must be filled
   - via unmove (i.e. a filler) born dominated

Enforcing the SMC

Recall:

1. Movers gonna move : $\neg(\$ \lhd \ell)$
2. No movement without movement : $\neg(\mathbf{move} \lhd \$)$
3. No competition : $\neg(\mathbf{move} \lhd \ell_1, \ell_2)$

How are these being enforced?

1. Two ways of generating a mover:
2. move nodes and movers postulated simultaneously

Enforcing the SMC

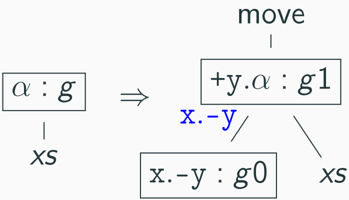
Recall:

1. Movers gonna move : $\neg(\$ \lhd \ell)$
2. No movement without movement : $\neg(\textbf{move} \lhd \$)$
3. No competition : $\neg(\textbf{move} \lhd \ell_1, \ell_2)$

How are these being enforced?

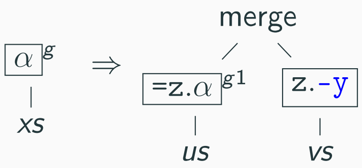
1. Two ways of generating a mover:
2. move nodes and movers postulated simultaneously
3. via restrictions

Restricting Unmove



As long as
nothing in *xs* is on the -y tier

Restricting Unmerge2



If there is something on the $-\mathrm{y}$ tier in *xs*
it must complete this gap

in other words, the $-\mathrm{y}$ tier is hereby blocked!

Completion (II)

A partial proof tree with an *n*-ary hole
is an operation of type

$$(\alpha_1 \to \cdots \to \alpha_n \to t) \to t$$

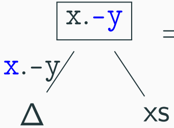
The 'α's are the types of the arguments to the hole

Upper bounds on

1. number of holes
2. their arity

depending on number of $-\mathrm{y}$ feature types in lexicon

Completion (III)



$$\Rightarrow \quad \Delta[us_1, \ldots, us_k]$$

Conditions

```
xs = sort ( us1 ++ ... ++ usk )
```

each substitution path is free for the relevant tier

A Note on Semantic Interpretation

$$[\![\textbf{merge}]\!] \mapsto \lambda m, n.(\!| m \oplus n |\!)$$

$$[\![\textbf{merge}]\!] \mapsto \lambda m, n.(\!| m \oplus \Box n |\!)$$

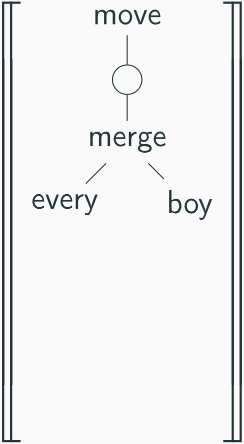
$$[\![\textbf{move}]\!] \mapsto \lambda m.m$$

$$[\![\textbf{move}]\!] \mapsto \lambda m.\langle m \rangle^{k}_{\oplus}$$

$$[\![\ell]\!] = \mathcal{I}(\ell)^{\uparrow}$$

```
(| f m n |) = do
      x <- m
      y <- n
      return (f x y)
```

The meaning of partial parse trees



$$= \lambda f_{\bigcirc}.[\![\mathit{move}]\!](f_{\bigcirc}\ ([\![\mathit{merge}]\!]\ [\![\mathit{every}]\!]\ [\![\mathit{boy}]\!]))$$

$$= \lambda f_{\bigcirc}.\langle f_{\bigcirc}\ ([\![\mathit{every}]\!]\ [\![\mathit{boy}]\!])^{\uparrow}\rangle^{k}_{FA}$$

Conclusion : Exploiting structure

- MGs have more structure in their derivations than is being made use of
  - how can we take advantage of it?
- Simple intersection w/ regular sets:

$$(\mathsf{will}, \texttt{=}^{\mathtt{r}}\mathtt{v}^{\mathtt{s}}.\texttt{+}^{\mathtt{p}}\mathtt{k}^{\mathtt{q}}.^{\mathtt{p}}\mathtt{c}^{\mathtt{s}}), \text{ where } \delta(q, \mathsf{will}) = r$$

  - how to do scheduling to obtain a version of the present algorithm?
- Left-corner parsing (for CFGs) has similar looking partial proof trees
  - can we use these ideas to get a left-corner parser for MGs and solve the problem of left branch movement?