Representations in Syntax

Greg Kobele

Thomas Müntzer Lecture

Universität Leipzig

Introduction

Gereon **(disappointedly)**
everyone is moving to representations

Introduction

Gereon **(disappointedly)**
everyone is moving to representations

What are representations?

- how should we think of them?
- what are the questions that we should ask?
- what is the trade-off with derivations?

What are derivations?
ibid.

We should focus on
what information we need to support the interface maps

Representations of Derivations

A derivation

A derivation

1. `select` *every*

   every

A derivation

1. `select` *every*

   every

2. `select` *boy*

   boy

A derivation

1. `select` *every*

   every

2. `select` *boy*

   boy

3. `merge` *1* and *2*

   $[_{DP}$ every $[_{NP}$ boy $]]$

A derivation

1. `select` *every*

   every

2. `select` *boy*

   boy

3. `merge` *1* and *2*

   $[_{DP}$ every $[_{NP}$ boy $]]$

4. `select` *laugh*

   laugh

A derivation

1. `select` *every*

   every

2. `select` *boy*

   boy

3. `merge` *1* and *2*

   $[_{DP}$ every $[_{NP}$ boy $]]$

4. `select` *laugh*

   laugh

5. `merge` *4* and *3*

   $[_{VP}$ laugh $[_{DP}$ every boy $]]$

A derivation

1. `select` *every*

   every

2. `select` *boy*

   boy

3. `merge` *1* and *2*

   $[_{DP}$ every $[_{NP}$ boy $]]$

4. `select` *laugh*

   laugh

5. `merge` *4* and *3*

   $[_{VP}$ laugh $[_{DP}$ every boy $]]$

6. `select` *will*

   will

A derivation

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

A derivation

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

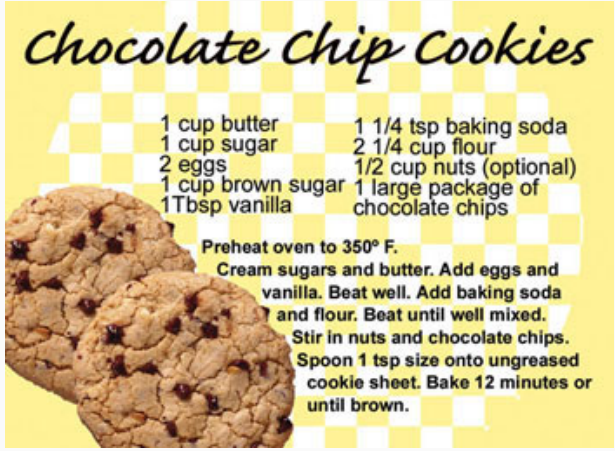
Derivations are processes

A derivation is the process
of constructing an expression

- derivations are important
- important things need to be thought about!
- it is helpful to be able to represent important things

Recipes are representations of processes

- lexical items are ingredients
- merge and move instead of bake, broil, whip, . . .



Derivations as recipes

1. `select` *every*
2. `select` *boy*
3. `merge` *1* and *2*
4. `select` *laugh*
5. `merge` *4* and *3*
6. `select` *will*
7. `merge` *6* and *5*
8. `move` *3* in *7*

Derivations are structured

Order is important

- Some things must happen before others
- Sometimes, it doesn't matter

- `merge` det and noun
- *before* you `merge` the verb

- `cream` sugar and butter
- *before* you `add` the flour

Represent *before*-ness as dominance:
if A must happen *before* B, then B should be higher than A

Representing derivations

Representing derivations

1. `select` *every*

every

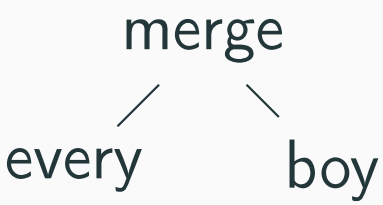
Representing derivations

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

every boy

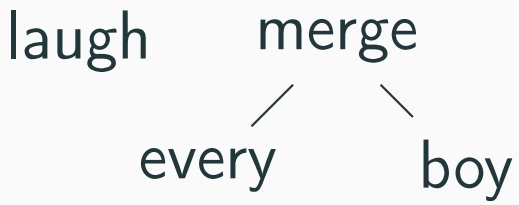
Representing derivations

1. `select` *every*
2. `select` *boy*
3. `merge` *1* and *2*



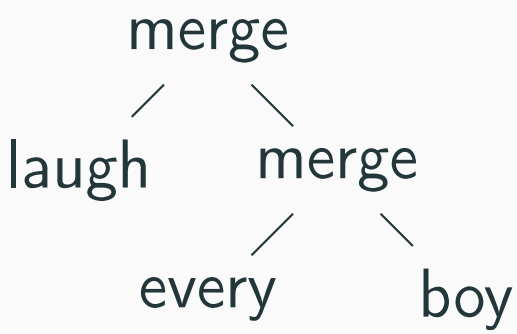
Representing derivations

1. `select` *every*
2. `select` *boy*
3. `merge` *1* and *2*
4. `select` *laugh*



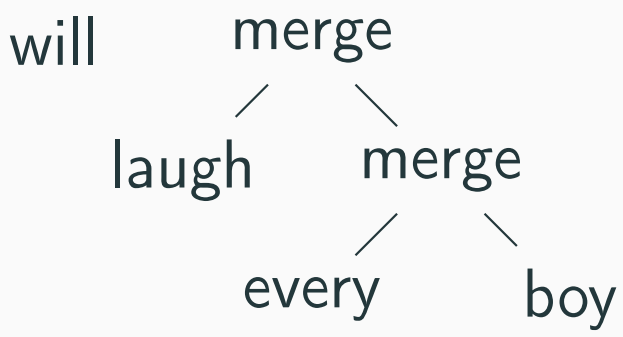
Representing derivations

1. `select` *every*
2. `select` *boy*
3. `merge` *1* and *2*
4. `select` *laugh*
5. `merge` *4* and *3*



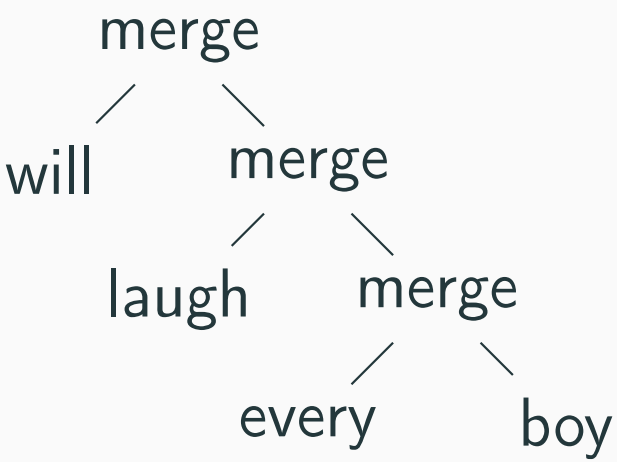
Representing derivations

1. `select` *every*
2. `select` *boy*
3. `merge` *1* and *2*
4. `select` *laugh*
5. `merge` *4* and *3*
6. `select` *will*



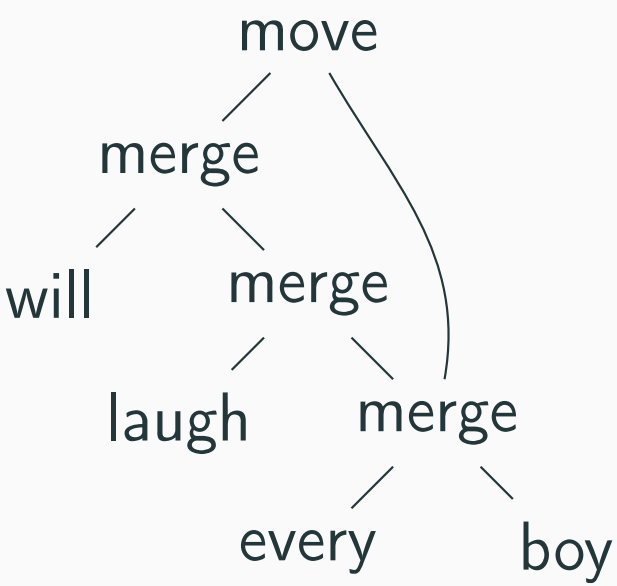
Representing derivations

1. `select` *every*
2. `select` *boy*
3. `merge` *1* and *2*
4. `select` *laugh*
5. `merge` *4* and *3*
6. `select` *will*
7. `merge` *6* and *5*

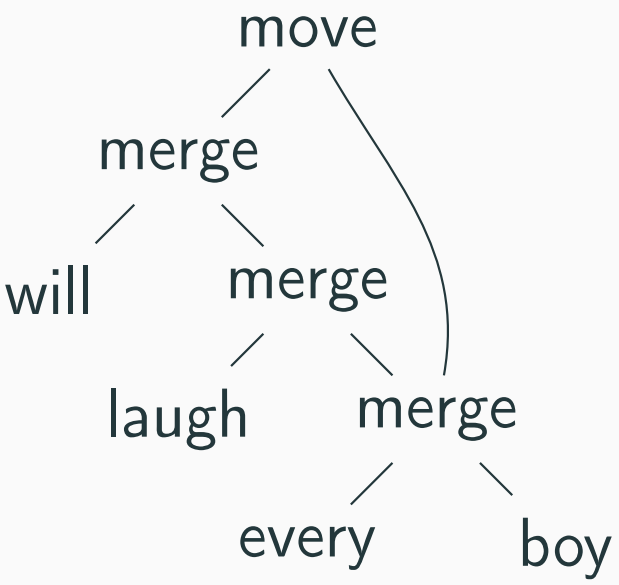


Representing derivations

1. `select` *every*
2. `select` *boy*
3. `merge` *1* and *2*
4. `select` *laugh*
5. `merge` *4* and *3*
6. `select` *will*
7. `merge` *6* and *5*
8. `move` *every boy*

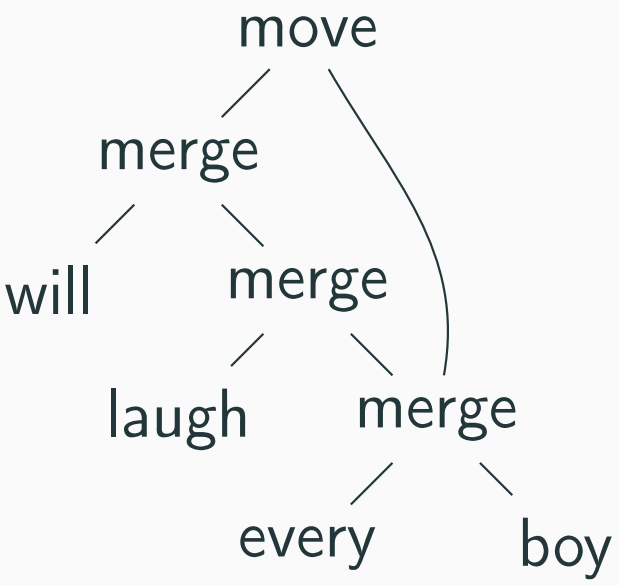


The structure of derivations



subtrees: describe how to construct something

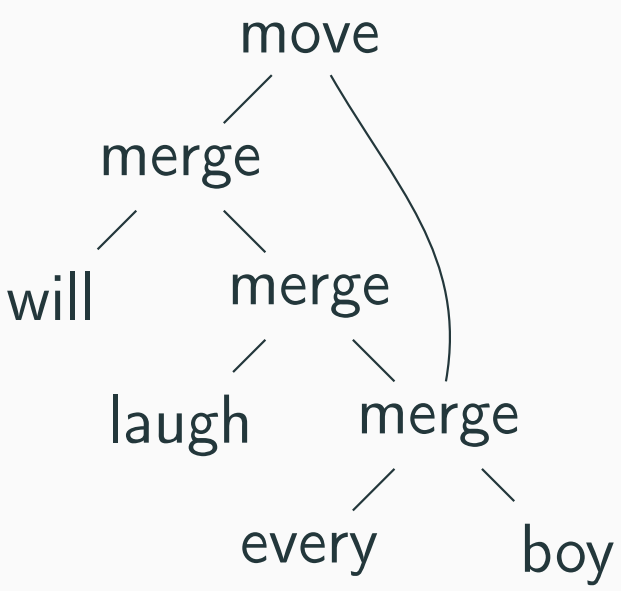
The structure of derivations



subtrees: describe how to construct something

x dominates y: to build *x*, you first have to build *y*

The structure of derivations

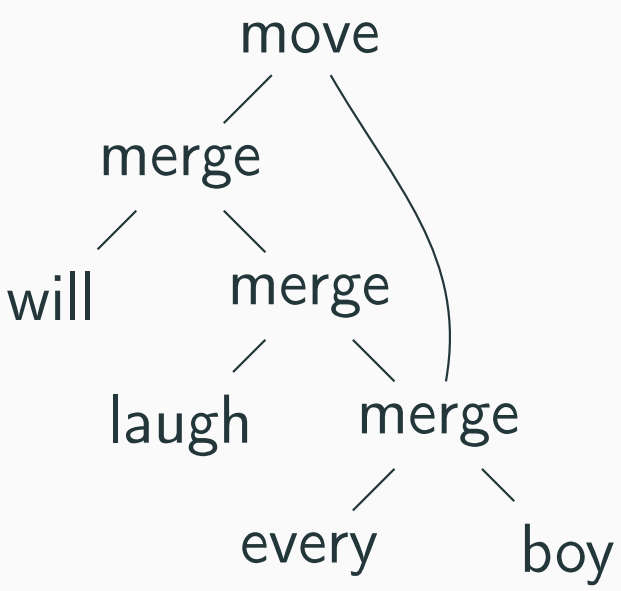


subtrees: describe how to construct something

x dominates y: to build *x*, you first have to build *y*

x c-commands y: before *x* can be used, you first have to build *y*

The structure of derivations



subtrees: describe how to construct something

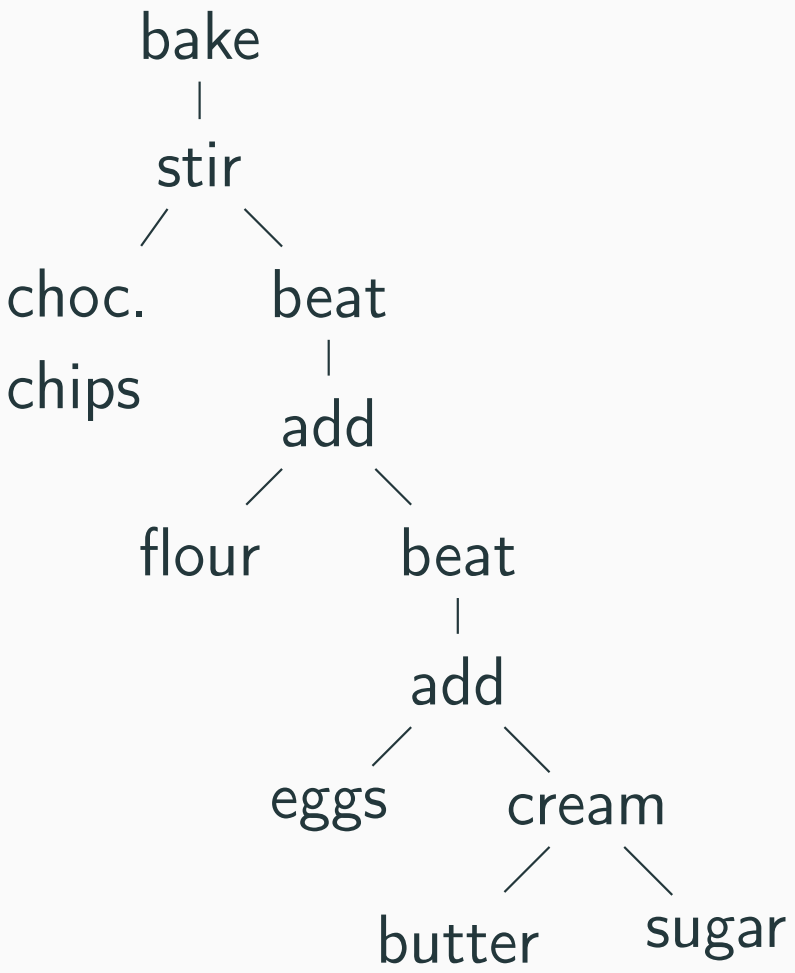
x dominates y: to build *x*, you first have to build *y*

x c-commands y: before *x* can be used, you first have to build *y*

x and y are independent: they can be built in any order

For comparison

1. `cream` *sugar* and *butter*
2. `add` *eggs* to *1*
3. `beat` *2*
4. `add` *flour* to *3*
5. `beat` *4*
6. `stir` *chocolate chips* into *5*
7. `bake` *6*



Infinite regress?

Do we have to build derivation trees?

NO!!!

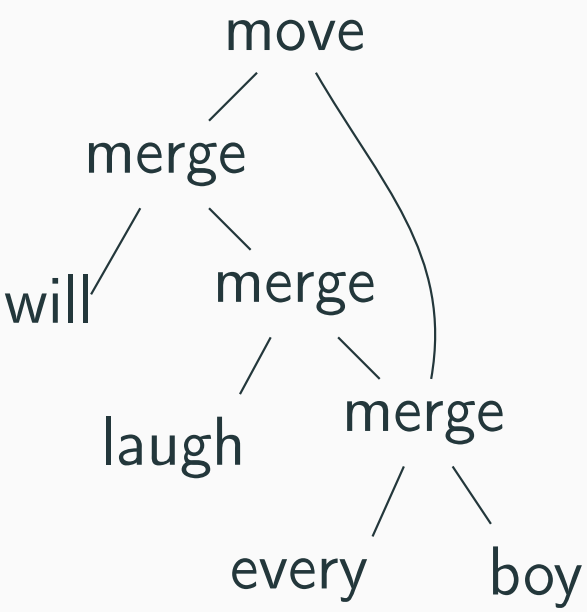
- a recipe is a description of the process, not the process itself
- a recipe is helpful to <u>think about</u> what you did/will do

You can make a cookie without writing down what you did/are doing/will do

Properties of Derivations

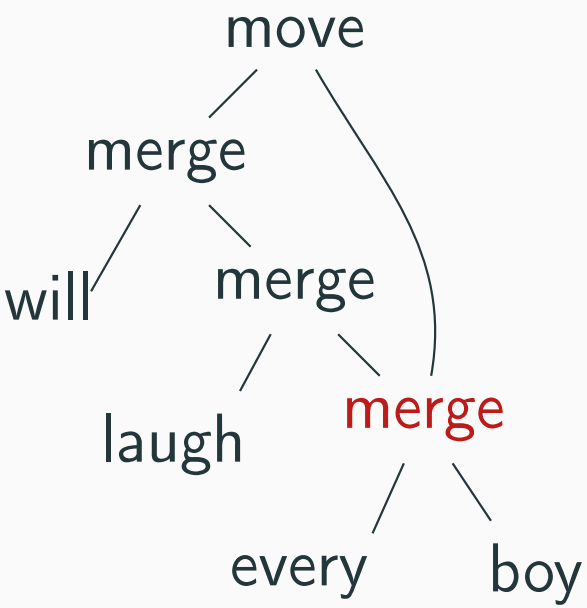
Why do derivations look the way they do?

Why?



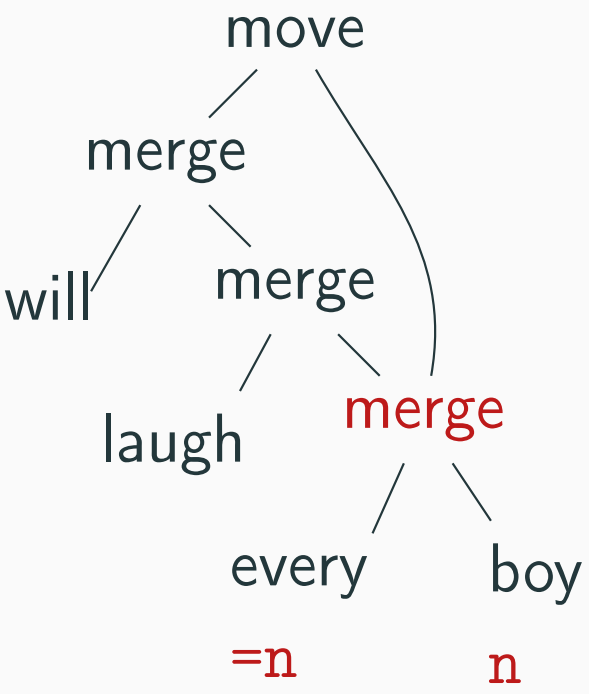
Why do derivations look the way they do?

Why?
because *every* selects for a *N*, and *boy* is an *N*



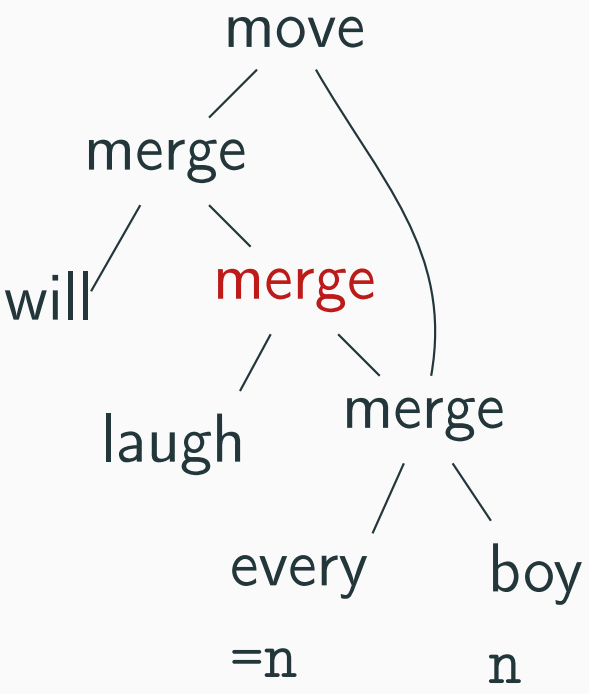
Why do derivations look the way they do?

Why?
because *every* selects for a *N*, and *boy* is an *N*



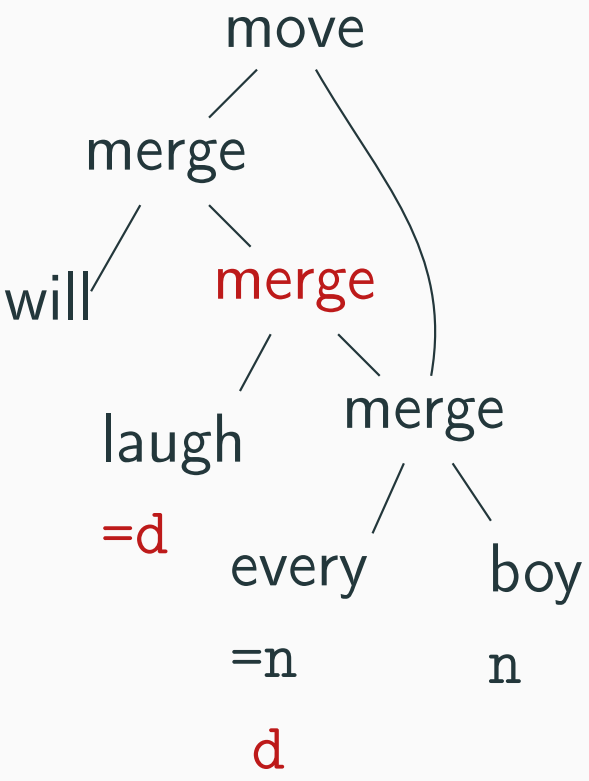
Why do derivations look the way they do?

Why?
because *laugh* selects for an *D*, and *every* is a *D*



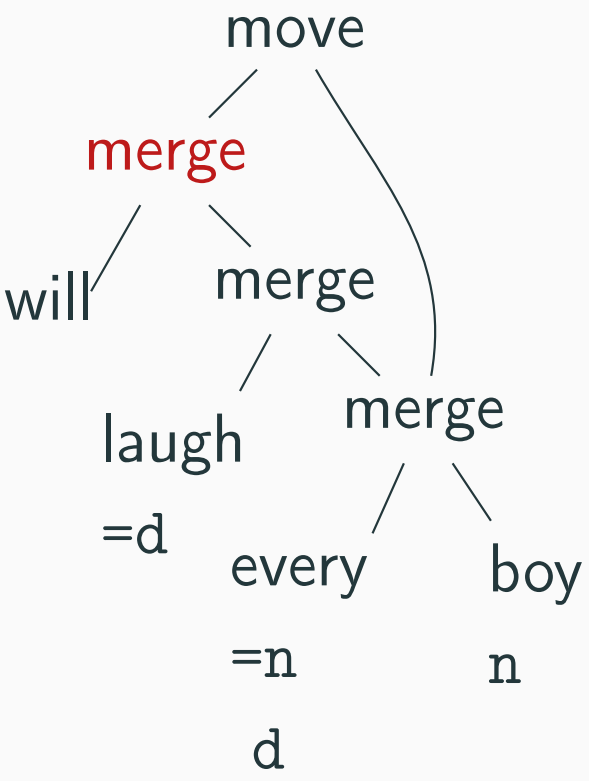
Why do derivations look the way they do?

Why?
because *laugh* selects for an *D*, and *every* is a *D*



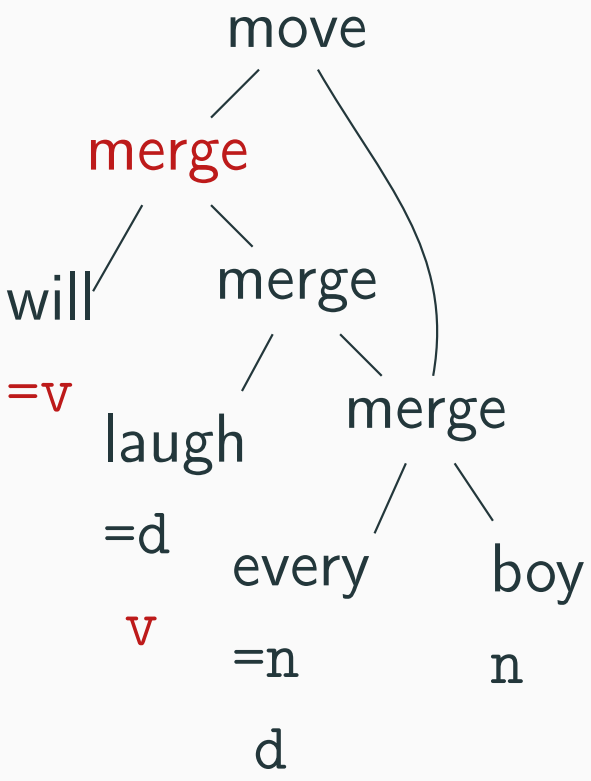
Why do derivations look the way they do?

Why?
because *will* selects for a *V*, and *laugh* is a *V*



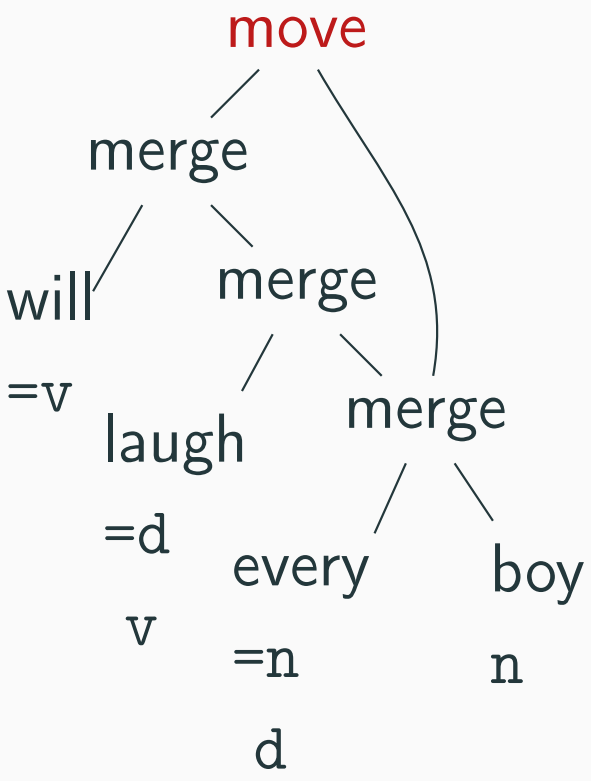
Why do derivations look the way they do?

Why?
because *will* selects for a *V*, and *laugh* is a *V*



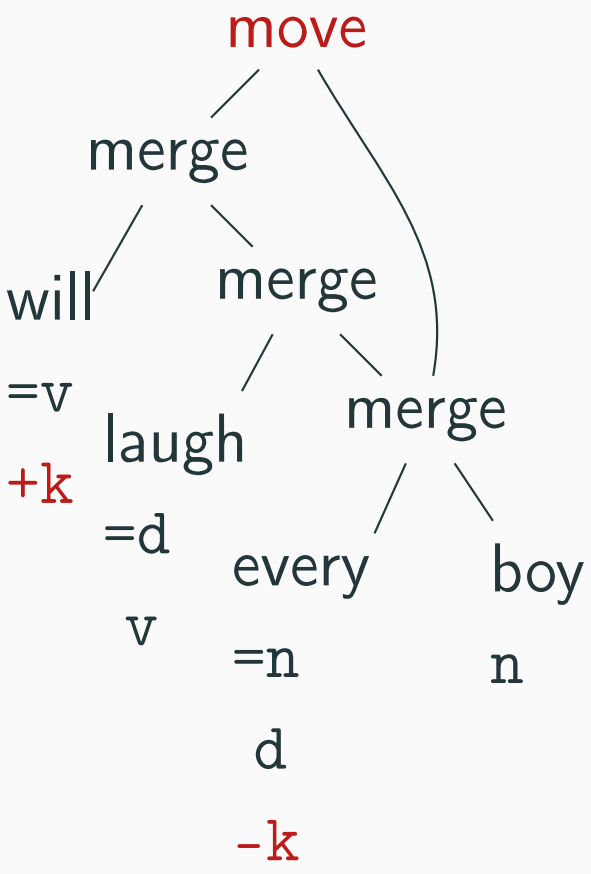
Why do derivations look the way they do?

Why?
because *every boy* needs case, and *will* assigns case

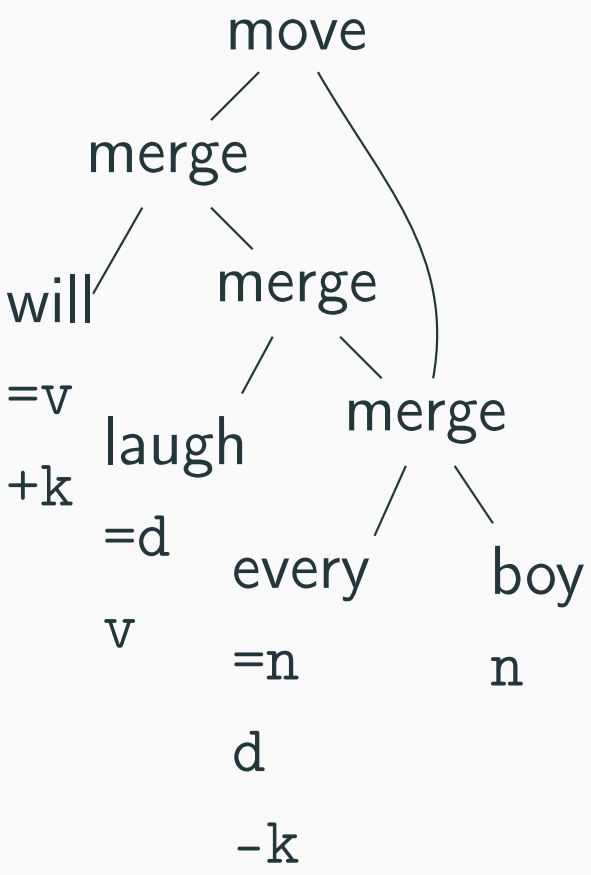


Why do derivations look the way they do?

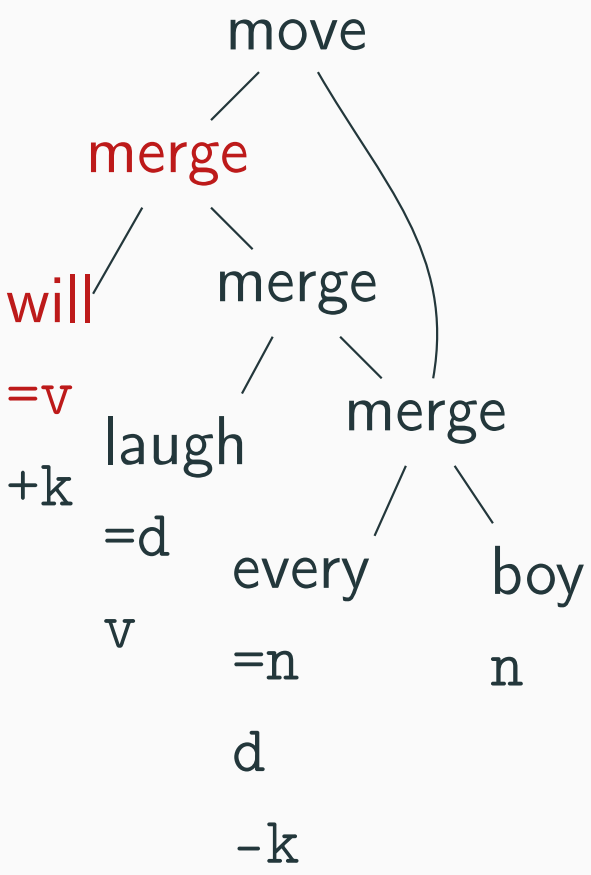
Why?
because *every boy* needs case, and *will* assigns case



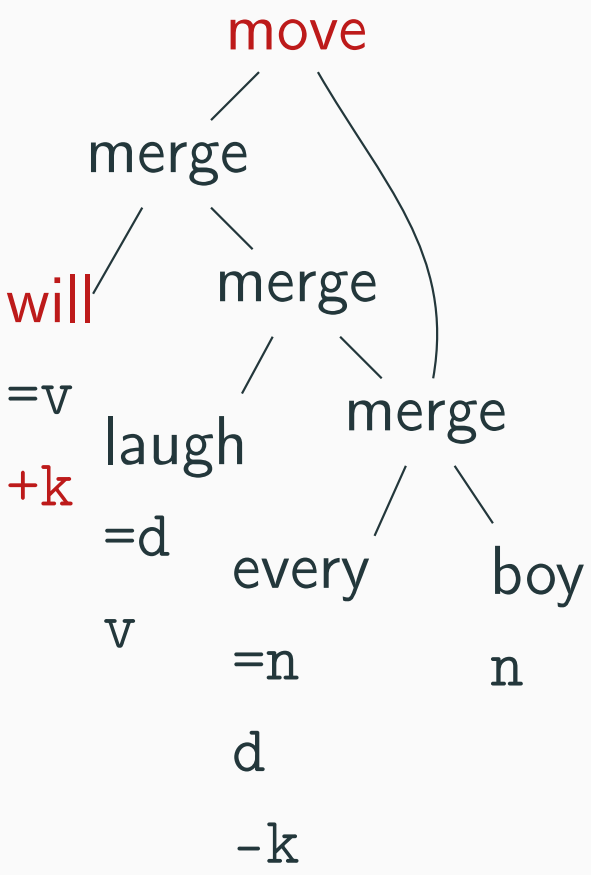
Derivations are endocentric



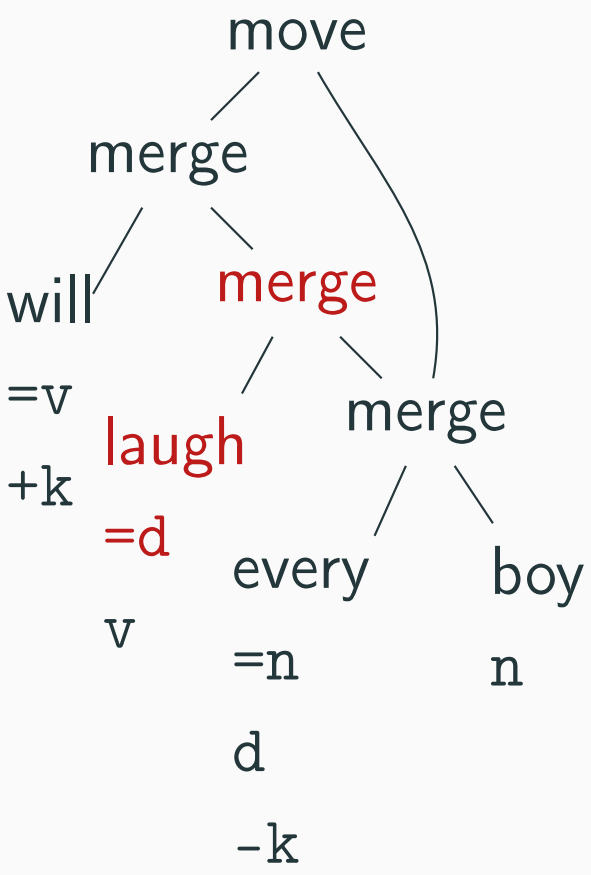
Derivations are endocentric



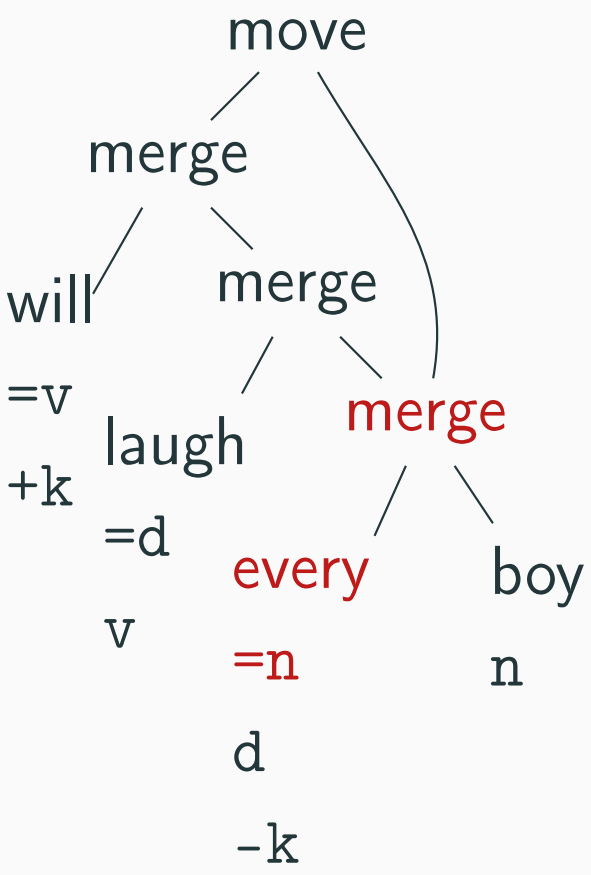
Derivations are endocentric



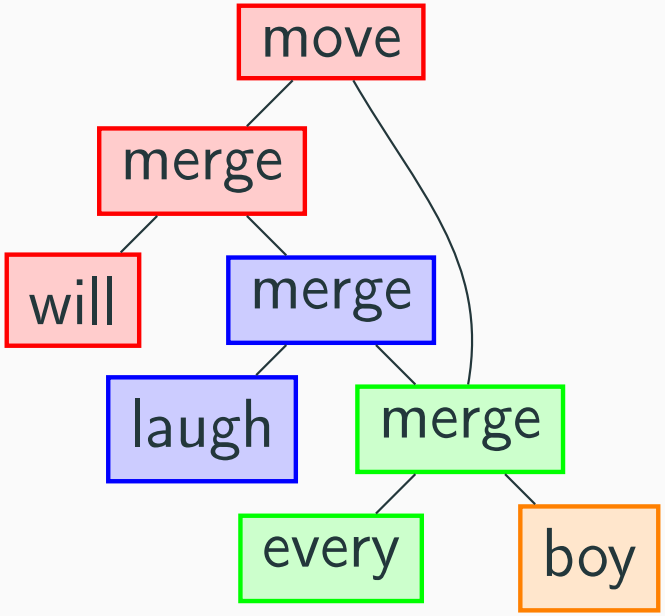
Derivations are endocentric



Derivations are endocentric

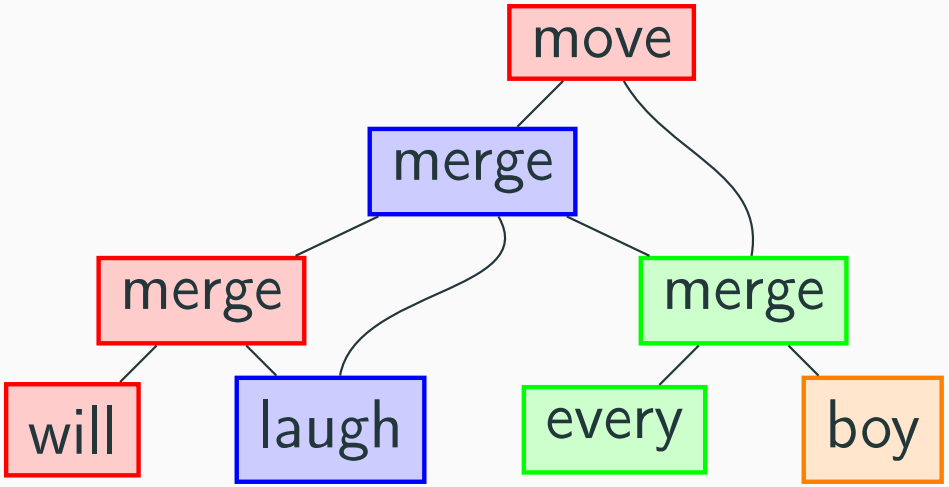


Derivations are endocentric

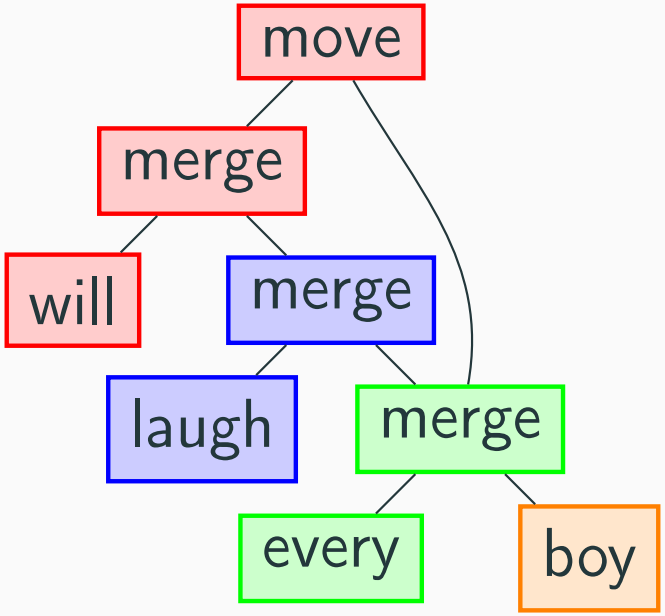


(. . . unless countercyclicity)

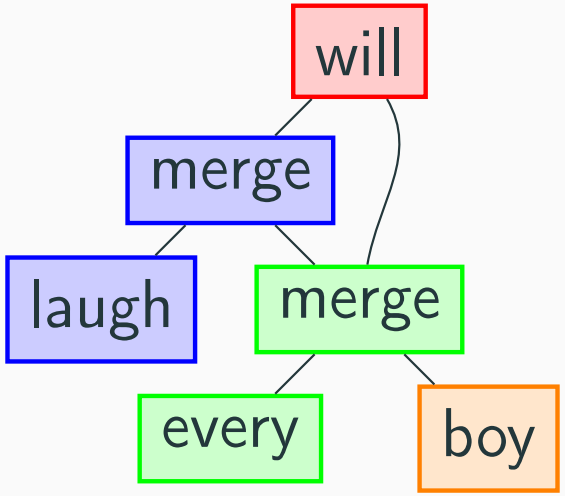
1. `select` *laugh*
2. `select` *will*
3. `merge` 2 and 1
4. `select` *every*
5. `select` *boy*
6. `merge` 4 and 5
7. `LATE merge` 6 to 1 in 3
8. `move` 6 in 7



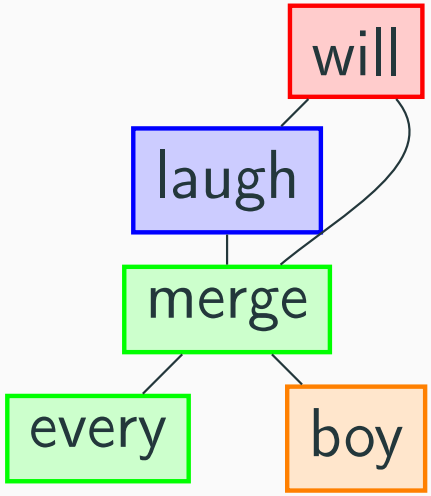
Headedness



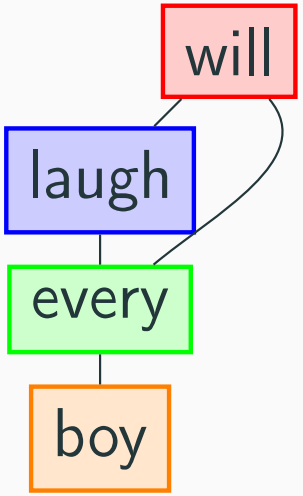
Headedness



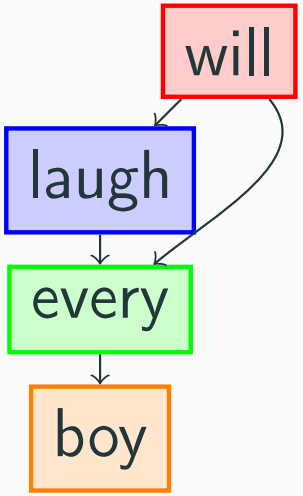
Headedness



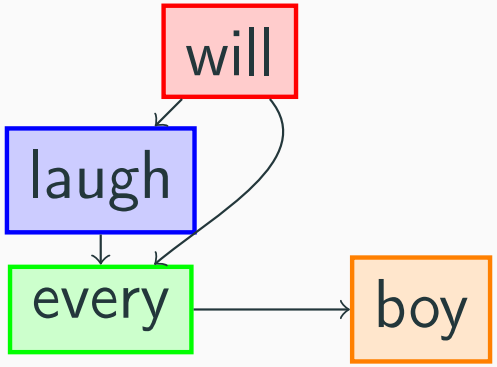
Headedness



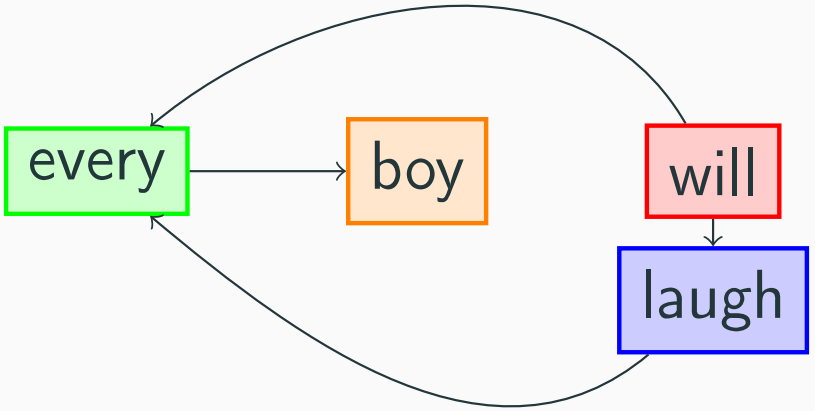
Headedness



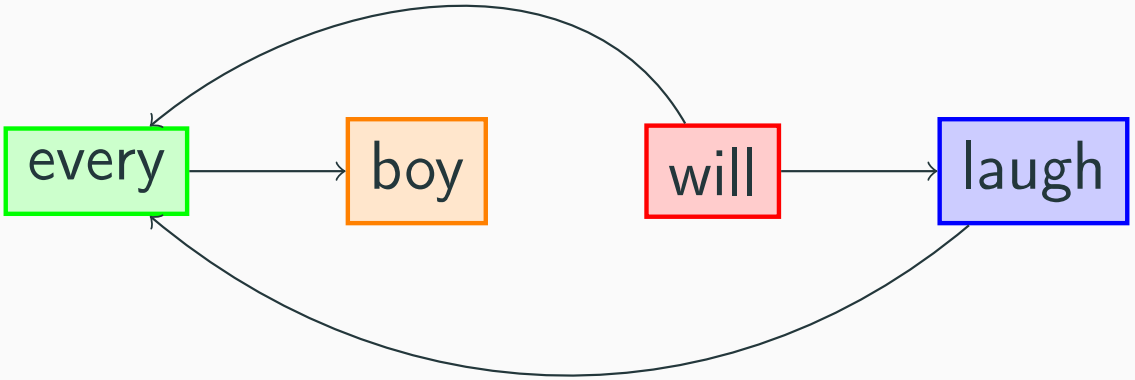
Headedness



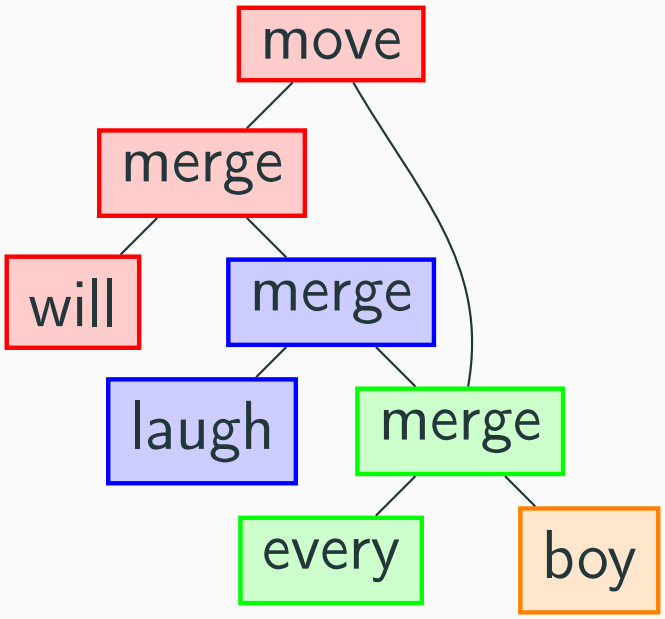
Headedness

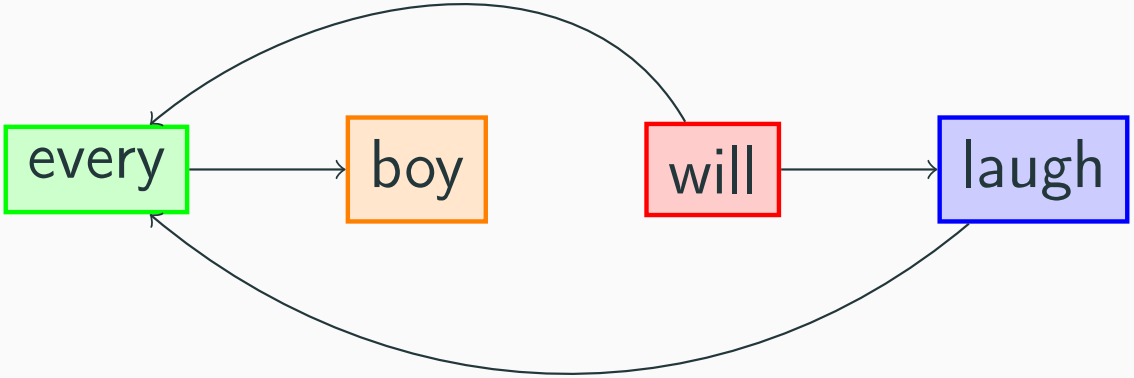


Headedness



The same recipe





Derived structure

every

every

Derived structure

every

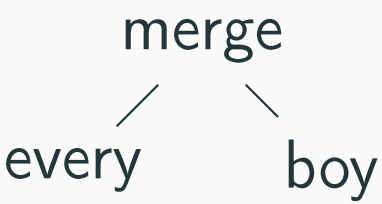
every

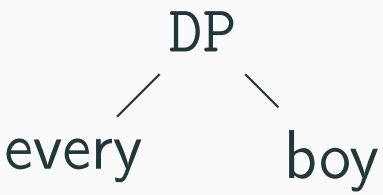
Derived structure

every boy

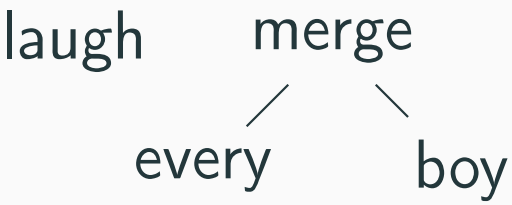
every boy

Derived structure

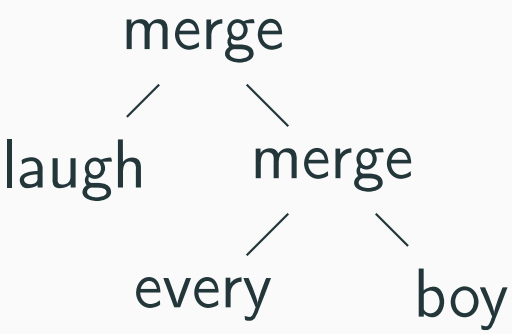


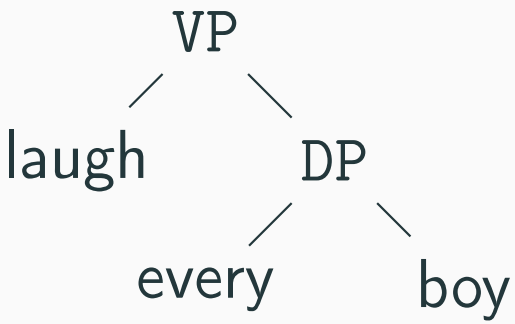


Derived structure

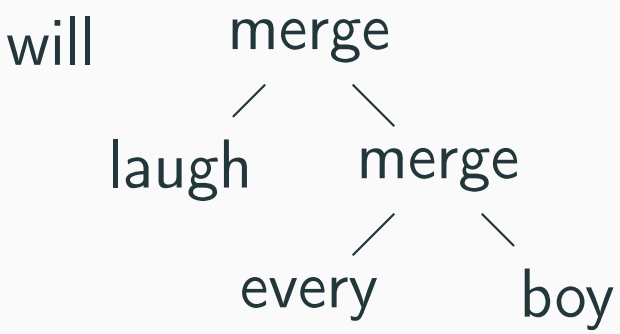


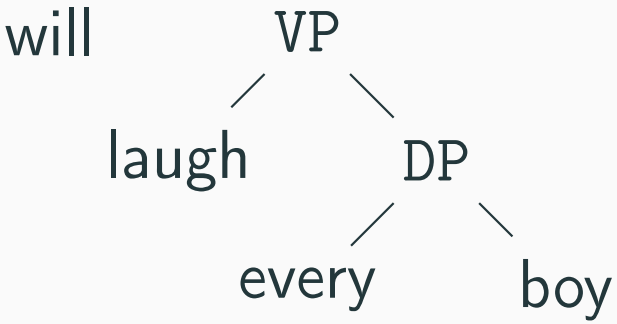
Derived structure



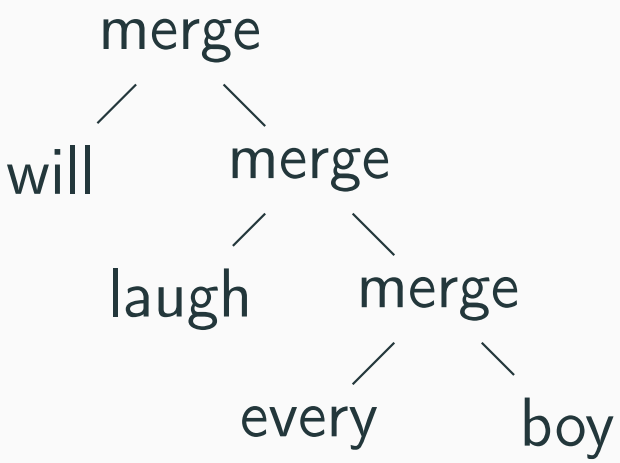


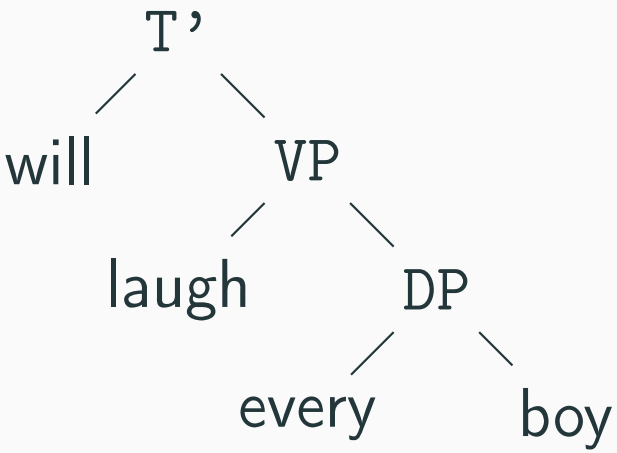
Derived structure



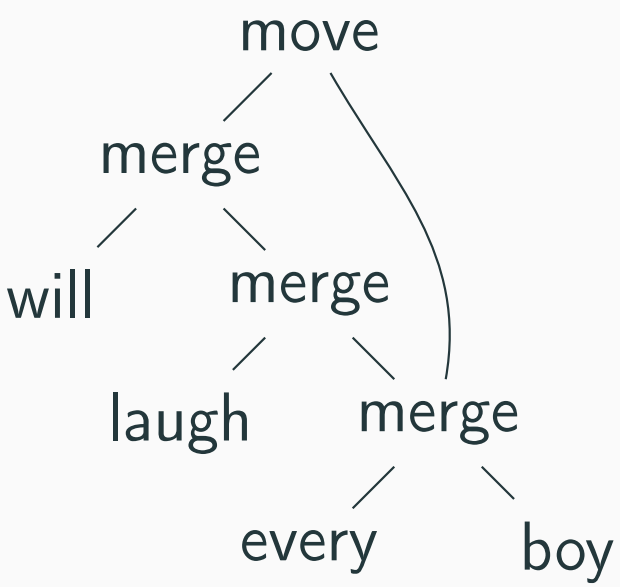


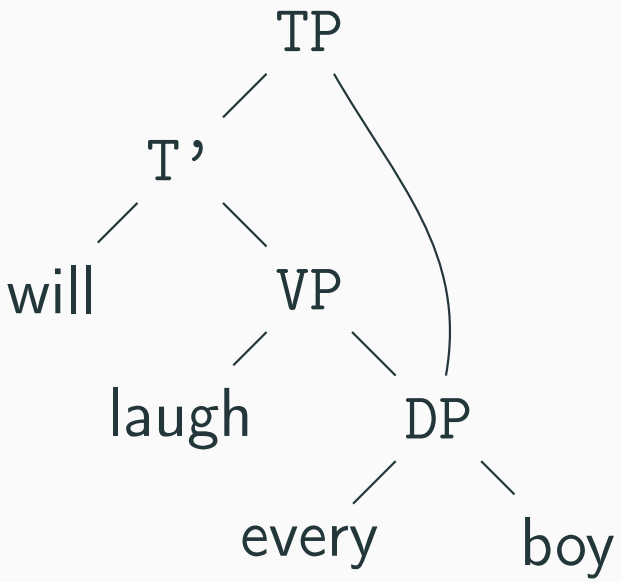
Derived structure





Derived structure





Derived structure (II)

every

every

Derived structure (II)

every

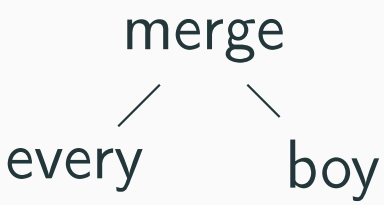
every

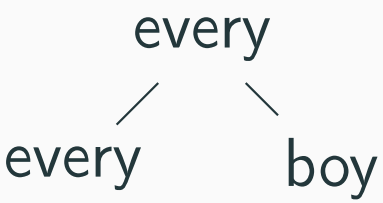
Derived structure (II)

every boy

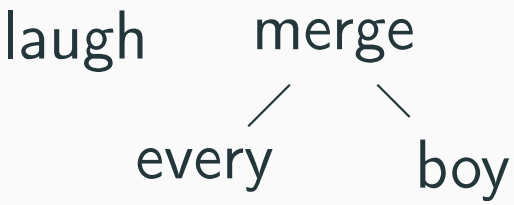
every boy

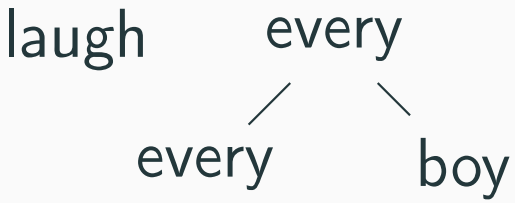
Derived structure (II)



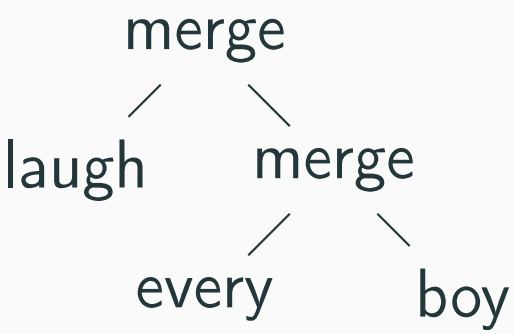


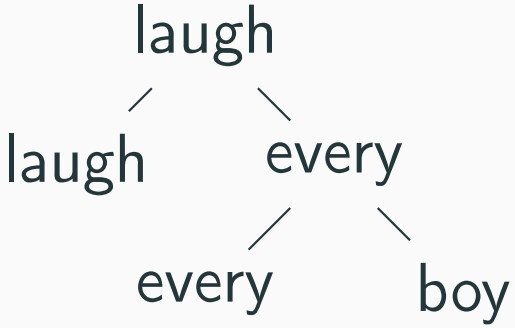
Derived structure (II)



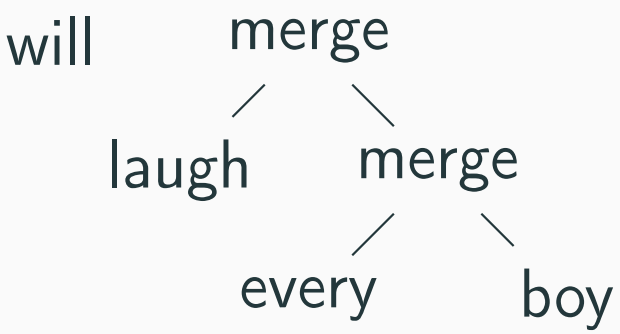


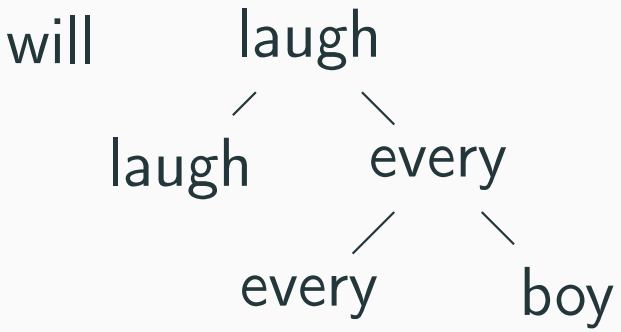
Derived structure (II)



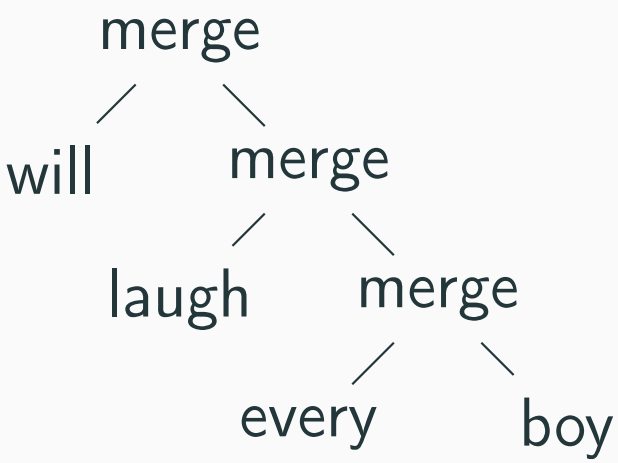


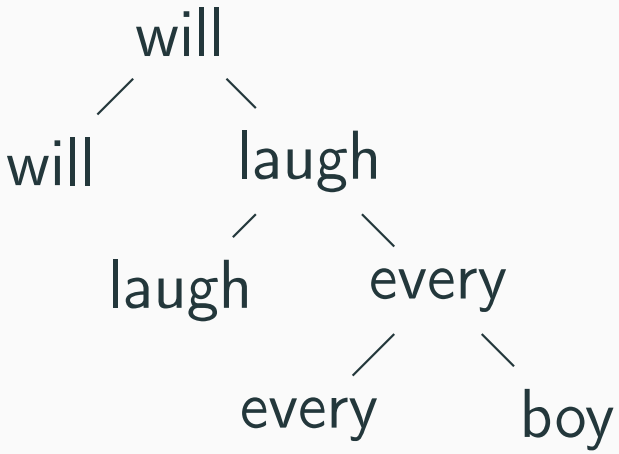
Derived structure (II)



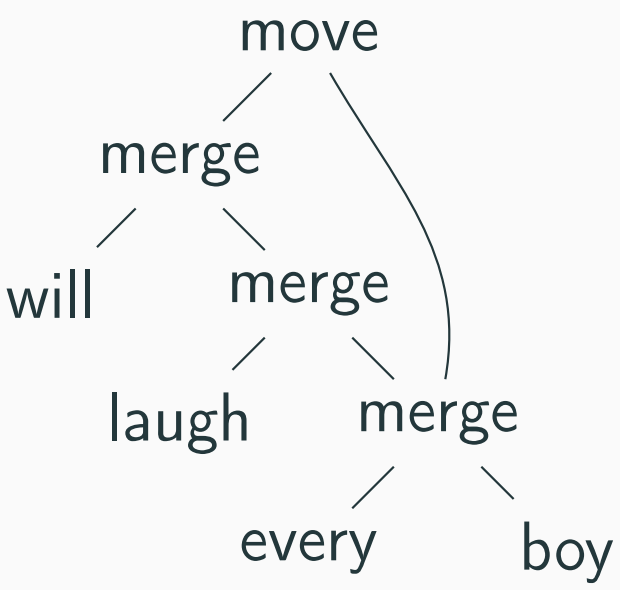


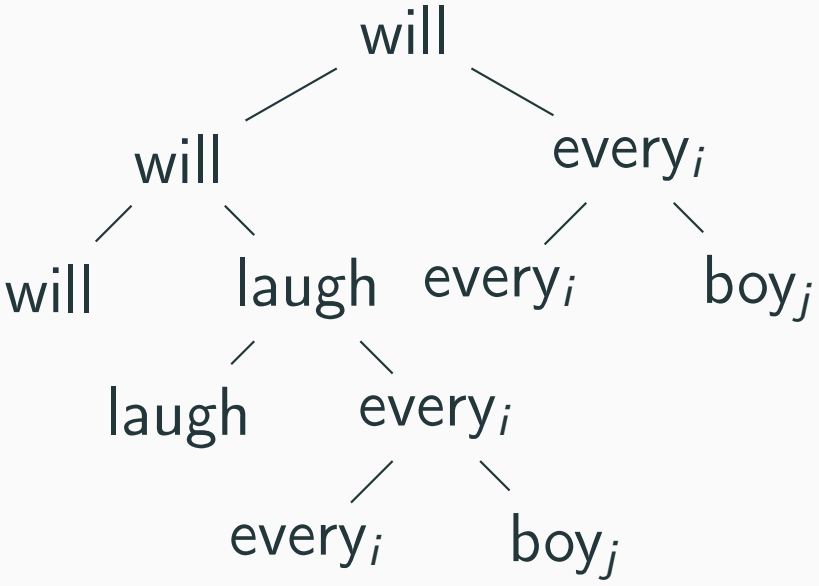
Derived structure (II)





Derived structure (II)





Derived structure (III)

every

every

Derived structure (III)

every

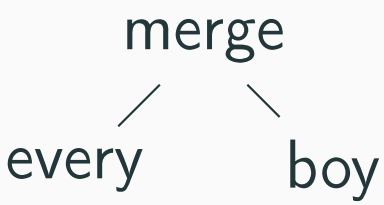
every

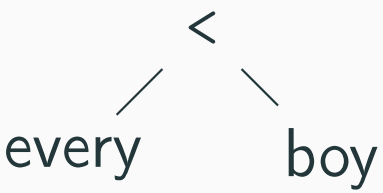
Derived structure (III)

every boy

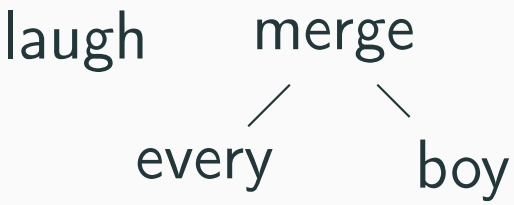
every boy

Derived structure (III)

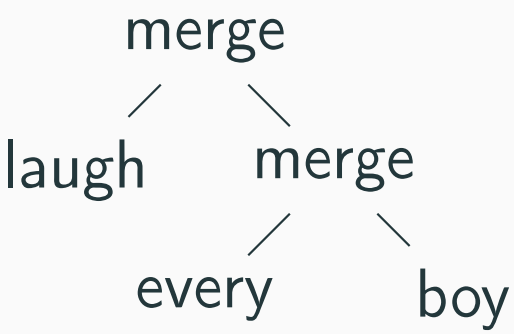




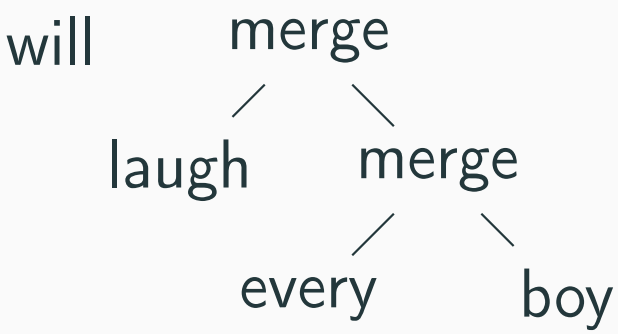
Derived structure (III)

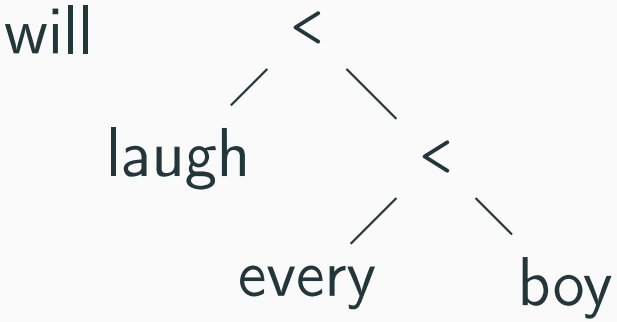


Derived structure (III)

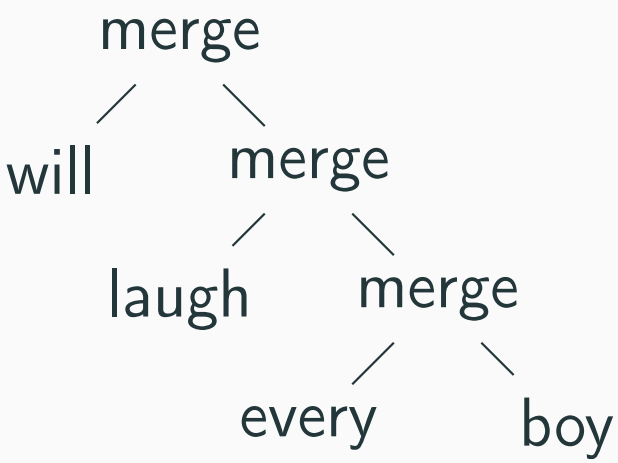


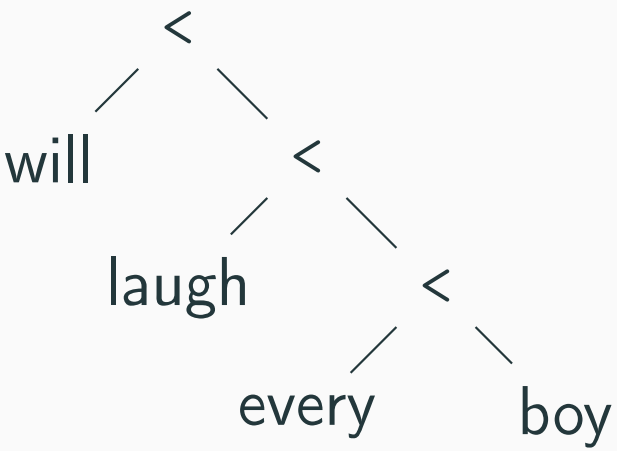
Derived structure (III)



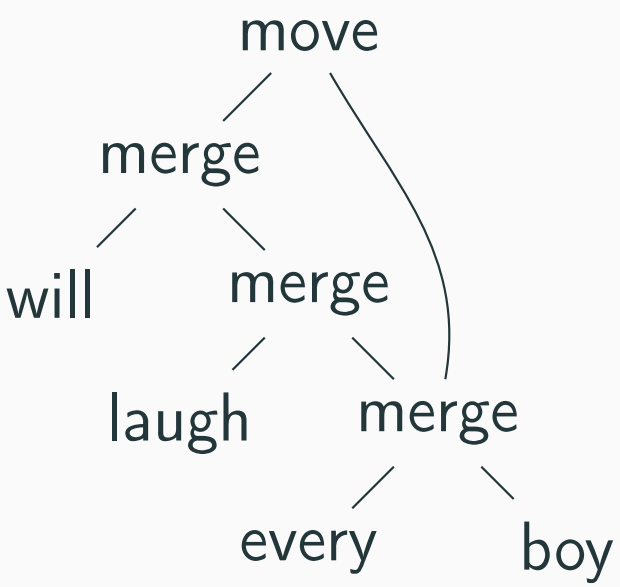


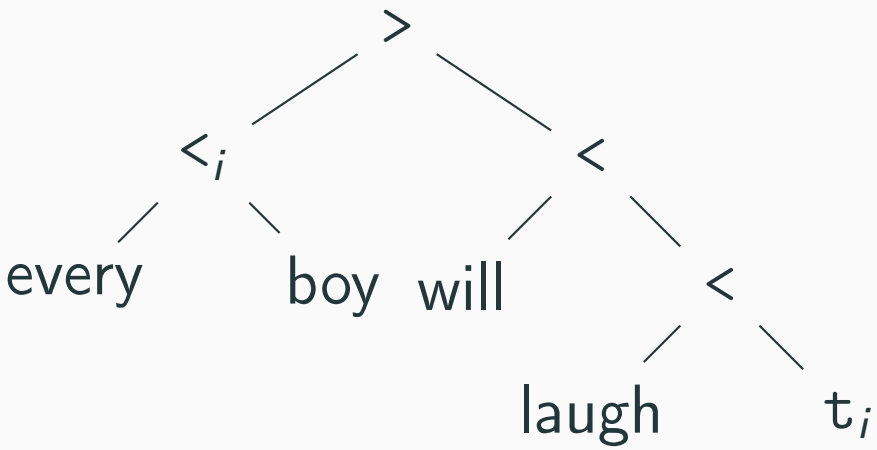
Derived structure (III)



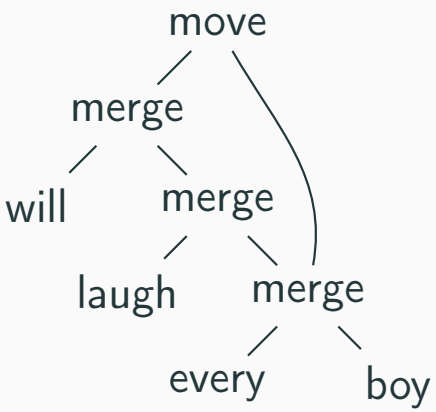


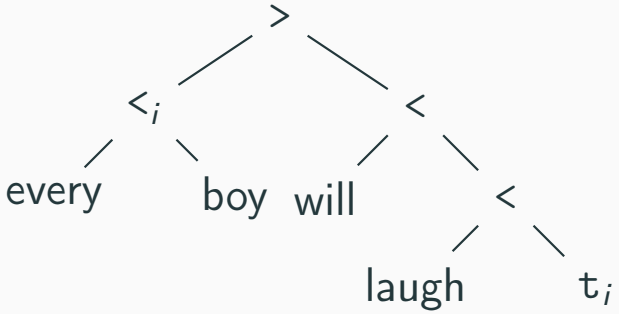
Derived structure (III)

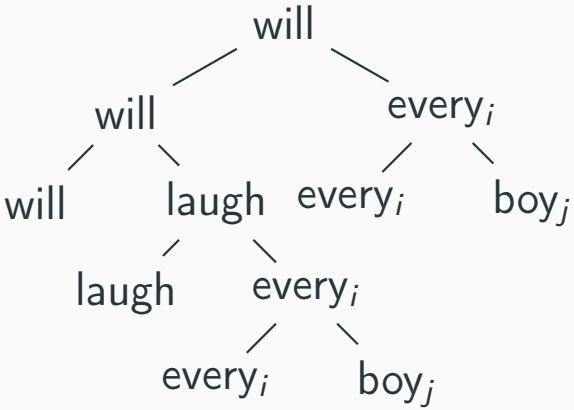


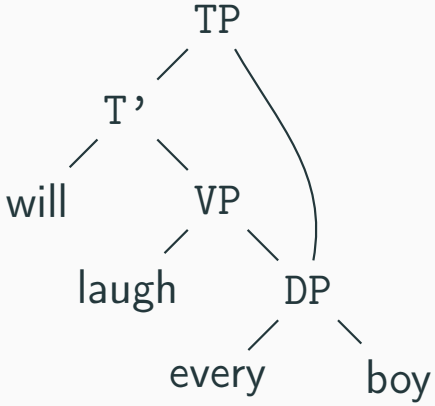


Same or Different?









Comparing Derived and Derivational Structure

- easy identity conditions for derivational structure
- derived structure is a copy of the derivation

Can we *replace* derived structure
with derivational structure?

- what is at issue here?

Processing

Is derivational structure real or not?

Previously:

Do we have to build derivation trees?

NO!!!

But now . . . ?

- am I proposing to replace derived trees w/ derivation trees?
- does this change things?

A parser

must construct a

1. well-formed
2. structure

the derivation

1. determines whether an expression is well-formed
2. gives you all the information you could ever want

a parser

A parser

must construct a

1. well-formed
2. structure

the derivation

1. determines whether an expression is well-formed
2. gives you all the information you could ever want

a parser 1. must reconstruct a derivation

A parser

must construct a

1. well-formed
2. structure

the derivation

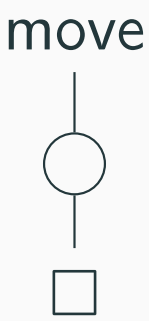
1. determines whether an expression is well-formed
2. gives you all the information you could ever want

a parser 1. must reconstruct a derivation and 2. needn't reconstruct anything else

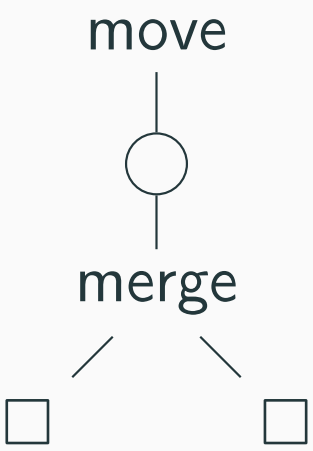
Parsing top down

□

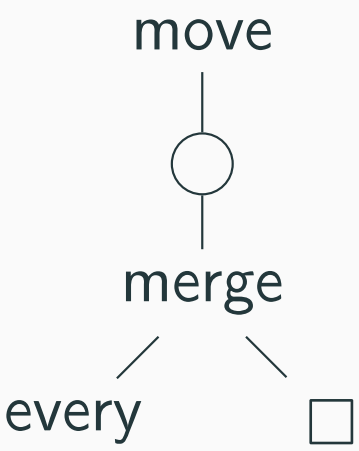
Parsing top down



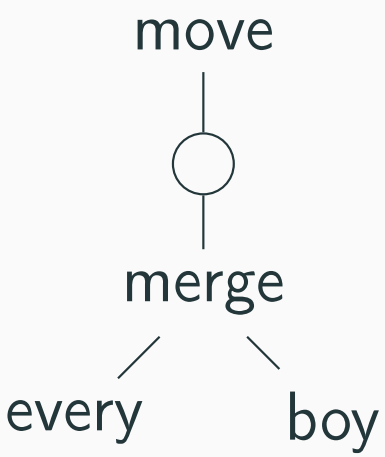
Parsing top down



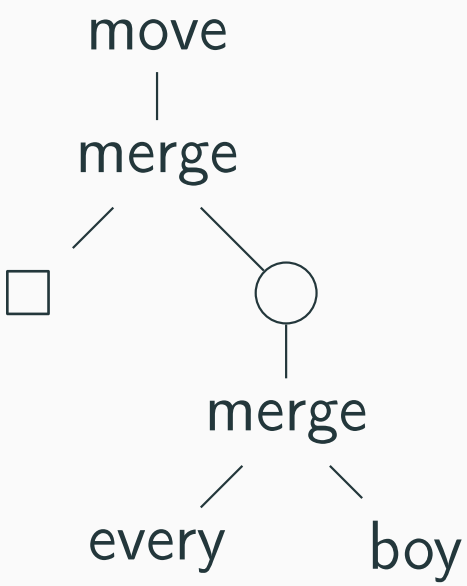
Parsing top down



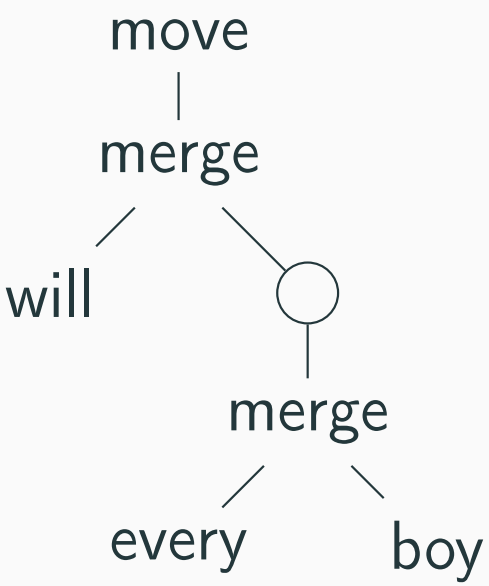
Parsing top down



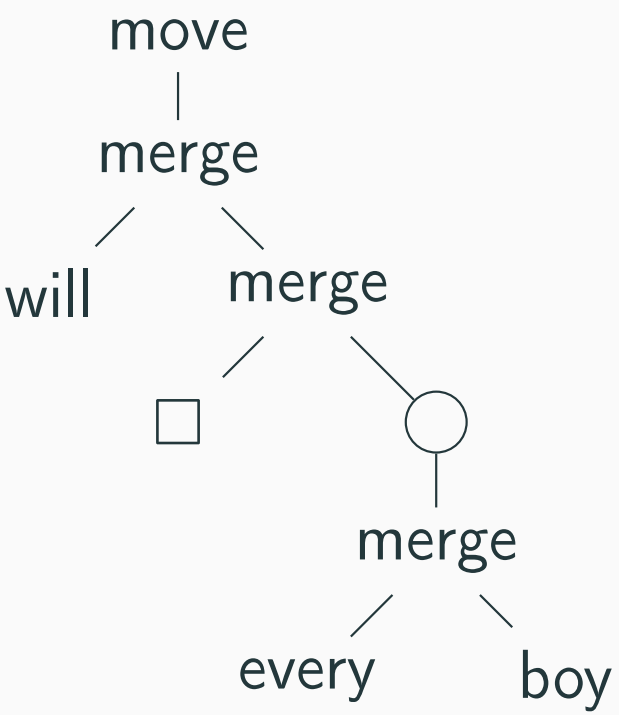
Parsing top down



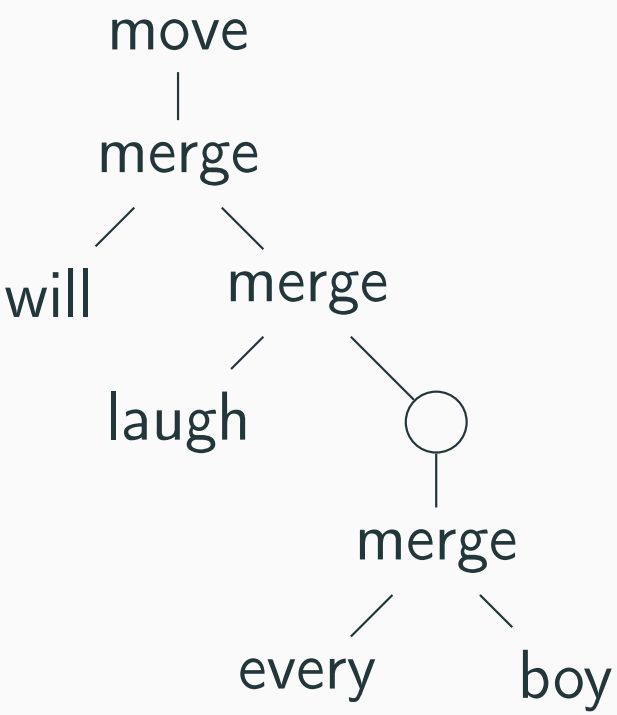
Parsing top down



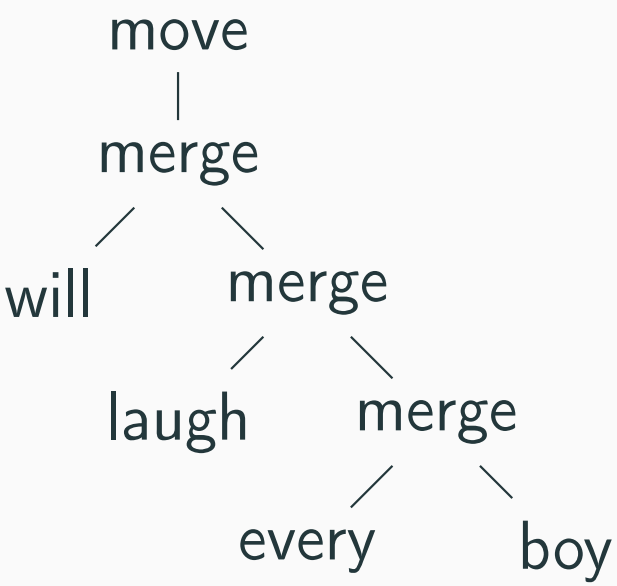
Parsing top down



Parsing top down



Parsing top down

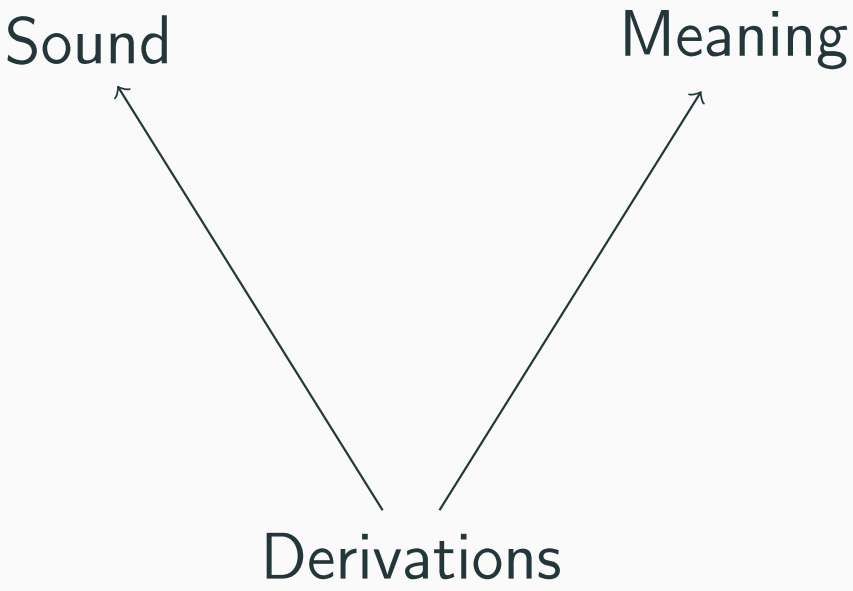


Looking at the parsing model

- parser must reconstruct the derivation
- so the derivation *is* a 'real' level of structure?

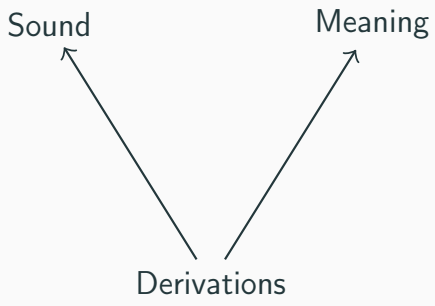
Compositionality

Grammatical architecture



The question
how do we go from *derivations* to sounds and meanings?

Interpreting derivations



a canonical idea

1. start w/ derivation tree
2. do the derivation described
3. interpret the derived object

But step 2. is just building a copy of what we started with!

Globality vs Locality

What is agreed upon?
never need to see the whole previous structure to decide about outcome of next step

'phases'

Ultra-locality

Compositionality
only use information about immediate arguments, and mode of combination, to determine result

$$\left[\!\!\left[\begin{array}{ccc} & \text{merge} & \\ \diagup & & \diagdown \\ \alpha & & \beta \end{array} \right]\!\!\right] = f_{\text{merge}}\ [\![\alpha]\!]\ [\![\beta]\!]$$

Ultra-locality

Compositionality
only use information about immediate arguments, and mode of combination, to determine result

$$\left[\!\!\left[\begin{array}{ccc} & \text{merge} & \\ \diagup & & \diagdown \\ \alpha & & \beta \end{array} \right]\!\!\right] = f_{\text{merge}}\ [\![\alpha]\!]\ [\![\beta]\!]$$

if interface maps are compositional

- then we never need to construct a derivation tree
- can interpret every step as we postulate it

(an example is coming)

The meaning of partial parse trees

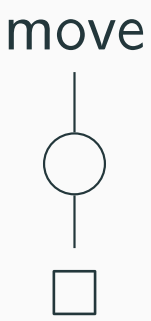
$$\left[\!\!\left[\begin{array}{c} \text{move} \\ | \\ \bigcirc \\ | \\ \text{merge} \\ \diagup \quad \diagdown \\ \text{every} \qquad \text{boy} \end{array} \right]\!\!\right] = \lambda f_{\bigcirc}.[\![move]\!](f_{\bigcirc}\,([\![merge]\!]\,[\![every]\!]\,[\![boy]\!]))$$

Deforestation of parsing

$$\square$$

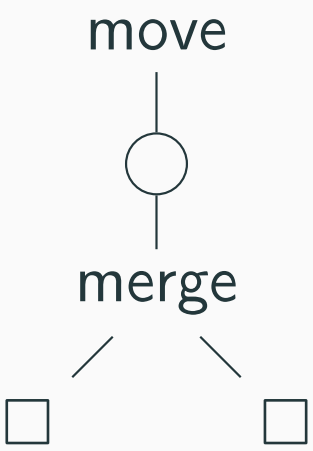
$$\lambda x_{\square}.x_{\square}$$

Deforestation of parsing



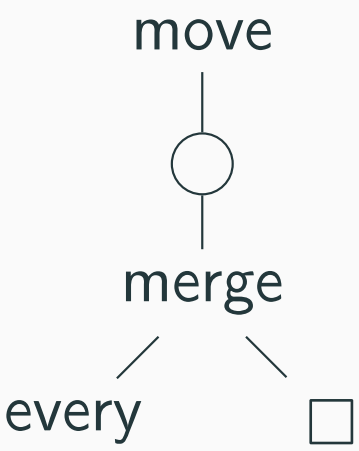
$$\lambda x_{\square}, f_{\bigcirc}.(f_{\bigcirc}\ x_{\square})'$$

Deforestation of parsing



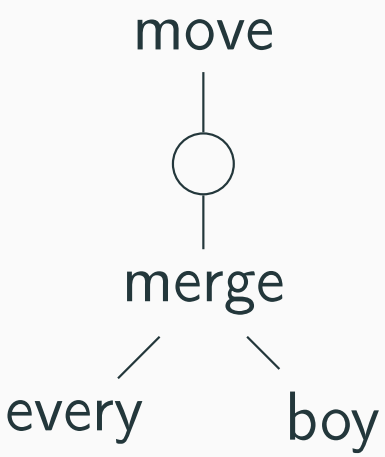
$$\lambda x_{\square}, y_{\square}, f_{\bigcirc}.(f_{\bigcirc}\ (x_{\square} \oplus y_{\square}))'$$

Deforestation of parsing



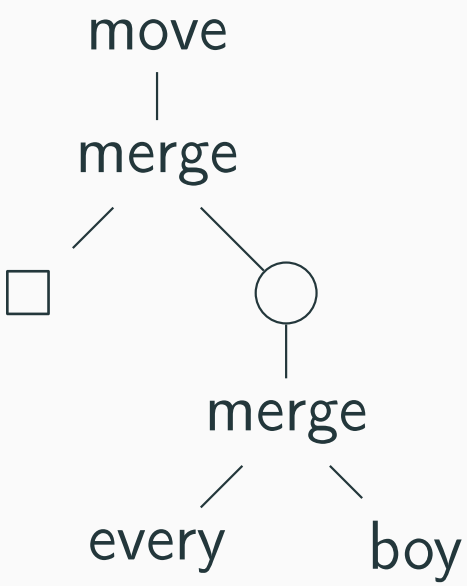
$$\lambda y_{\square}, f_{\bigcirc}.(f_{\bigcirc}\ ([\![every]\!] \oplus y_{\square}))'$$

Deforestation of parsing



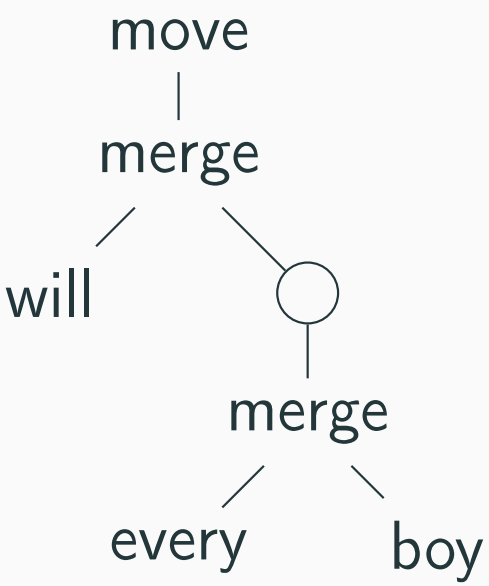
$$\lambda f_{\bigcirc}.(f_{\bigcirc}\ ([\![every]\!] \oplus [\![boy]\!]))'$$

Deforestation of parsing



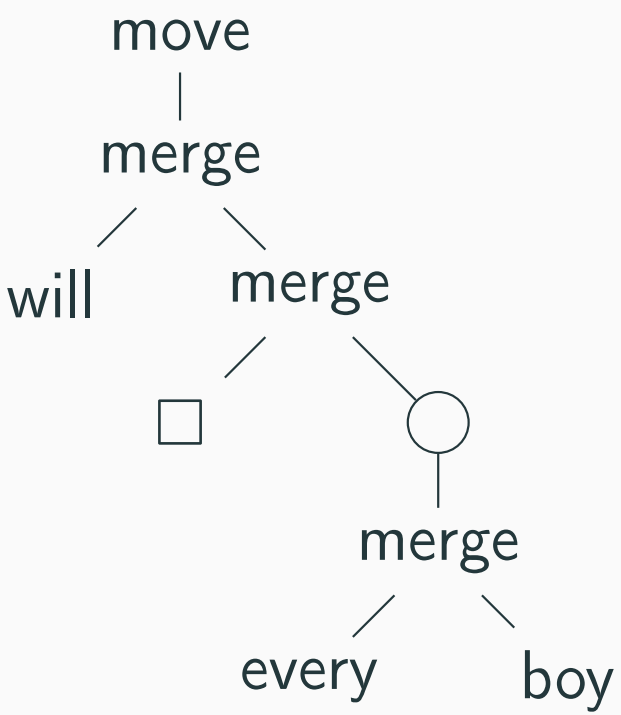
$$\lambda x_{\square}, f_{\bigcirc}.(x_{\square} \oplus (f_{\bigcirc}\ ([\![every]\!] \oplus [\![boy]\!])))'$$

Deforestation of parsing



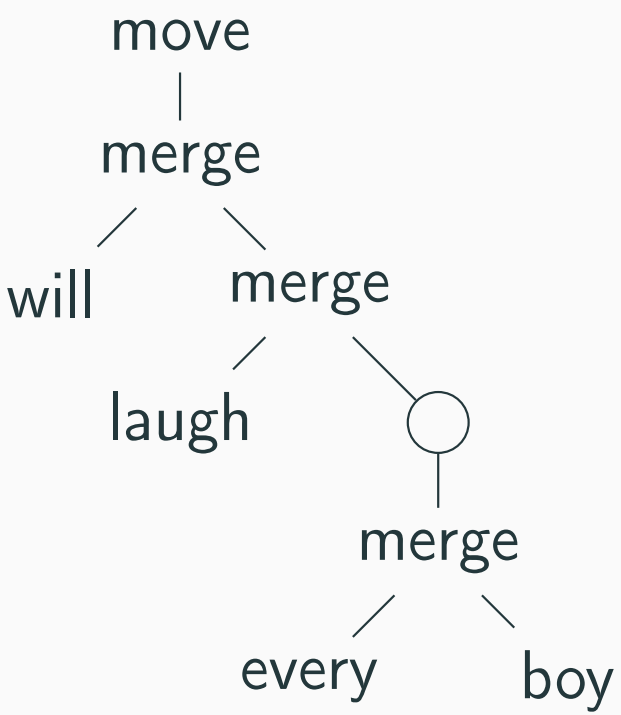
$$\lambda f_{\bigcirc}.([\![will]\!] \oplus (f_{\bigcirc}\ ([\![every]\!] \oplus [\![boy]\!])))'$$

Deforestation of parsing



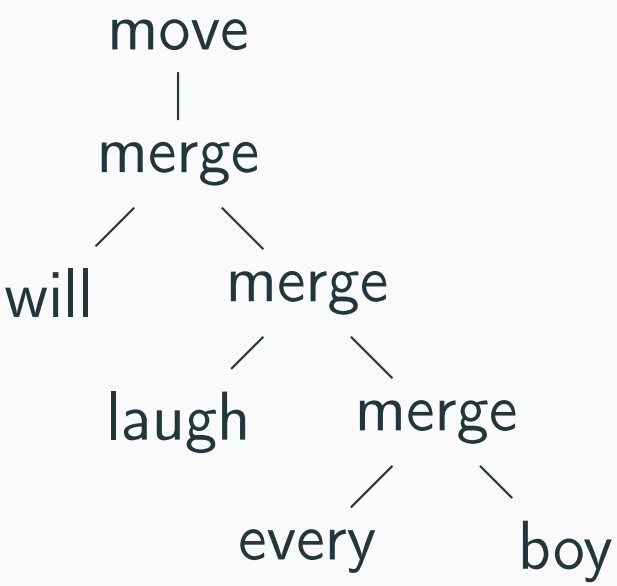
$$\lambda x_{\square}, f_{\bigcirc}.(\llbracket \mathit{will} \rrbracket \oplus (x_{\square} \oplus (f_{\bigcirc}\ (\llbracket \mathit{every} \rrbracket \oplus \llbracket \mathit{boy} \rrbracket))))'$$

Deforestation of parsing



$$\lambda f_{\bigcirc}.([\![\mathit{will}]\!] \oplus ([\![\mathit{laugh}]\!] \oplus (f_{\bigcirc}\ ([\![\mathit{every}]\!] \oplus [\![\mathit{boy}]\!]))))'$$

Deforestation of parsing



$$(\llbracket \mathit{will} \rrbracket \oplus (\llbracket \mathit{laugh} \rrbracket \oplus (\llbracket \mathit{every} \rrbracket \oplus \llbracket \mathit{boy} \rrbracket)))'$$

Dirty Tricks

A trick

add input structures to output domain

$$f_{\mathsf{merge}}\ a\ b = \begin{tikzcd}[column sep=small] & \mathsf{merge} \arrow[dl, dash] \arrow[dr, dash] & \\ \alpha & & \beta \end{tikzcd}$$

Dirty Tricks

A trick

add input structures to output domain

$$f_{\text{merge}}\ a\ b = \begin{tikzcd}[column sep=small] & \text{merge} \arrow[dl, dash] \arrow[dr, dash] & \\ \alpha & & \beta \end{tikzcd}$$

This is the point of derived structure

Compositionality

Compositionality is a restriction when **(Kracht)**

1. we limit what f_{merge} and f_{move} can do, *and*
2. we restrict what interpretations can be

Compositionality

Compositionality is a restriction when **(Kracht)**

1. we limit what f_{merge} and f_{move} can do, *and*
2. we restrict what interpretations can be

What should interpretations be?

- whatever we need
- if we end up needing craziness, we should worry

One man's junk . . .

Computer scientists
usually are happy to attach extra information to interpretations

- as long as it is finite

One man's junk . . .

Computer scientists
usually are happy to attach extra information to interpretations

- as long as it is finite

Example
add categorial information to strings

One man's junk . . .

Computer scientists
usually are happy to attach extra information to interpretations

- as long as it is finite

Example
add categorial information to strings

because we can think of this as being part of the operations instead:

not just `merge`, but `merge-D-NP`, `merge-V-DP`,...

What do we need

keep track of the unchecked syntactic features
(I won't talk about this here)

What do we need

keep track of the unchecked syntactic features
(I won't talk about this here)

For PF
keep track of which phrases are still moving

but not of their internal structure

An example

An example

every

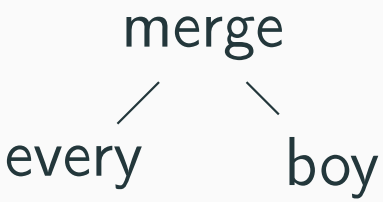
(every, =n.d.-k)

An example

every boy

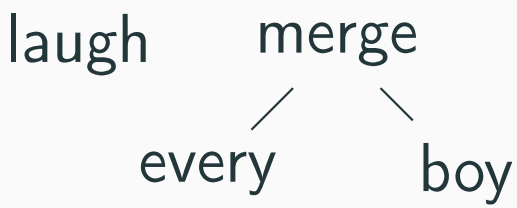
(every, =n.d.-k) (boy, n)

An example



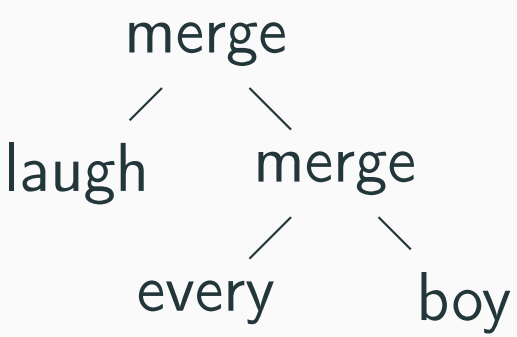
$$\frac{(\text{every}, \texttt{=n.d.-k}) \qquad (\text{boy}, \texttt{n})}{(\text{every boy}, \texttt{d.-k})}$$

An example



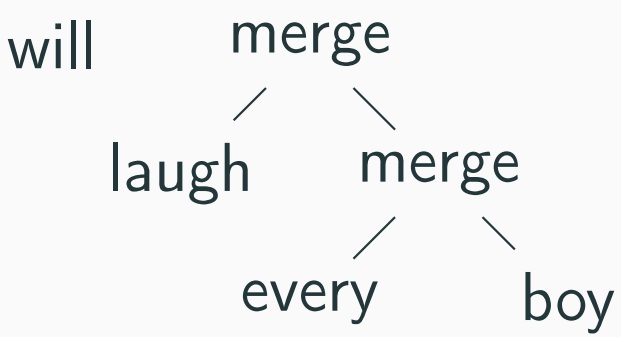
$$(\text{laugh}, \texttt{=d.v}) \qquad \frac{(\text{every}, \texttt{=n.d.-k}) \qquad (\text{boy}, \texttt{n})}{(\text{every boy}, \texttt{d.-k})}$$

An example



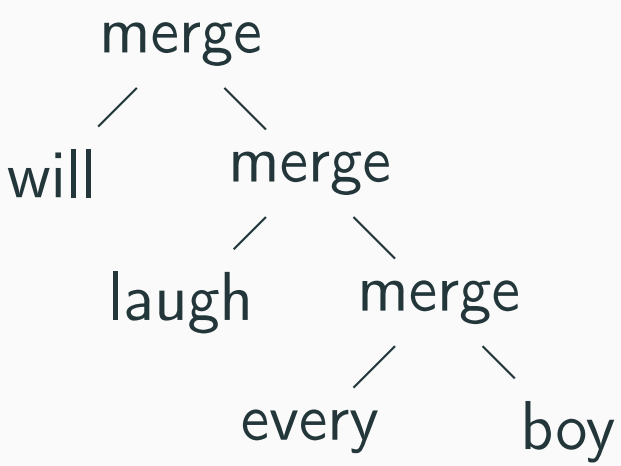
$$\dfrac{(\text{laugh}, \texttt{=d.v}) \qquad \dfrac{(\text{every}, \texttt{=n.d.-k}) \qquad (\text{boy}, \texttt{n})}{(\text{every boy}, \texttt{d.-k})}}{(\text{laugh}, \texttt{v}), (\text{every boy}, \texttt{-k})}$$

An example



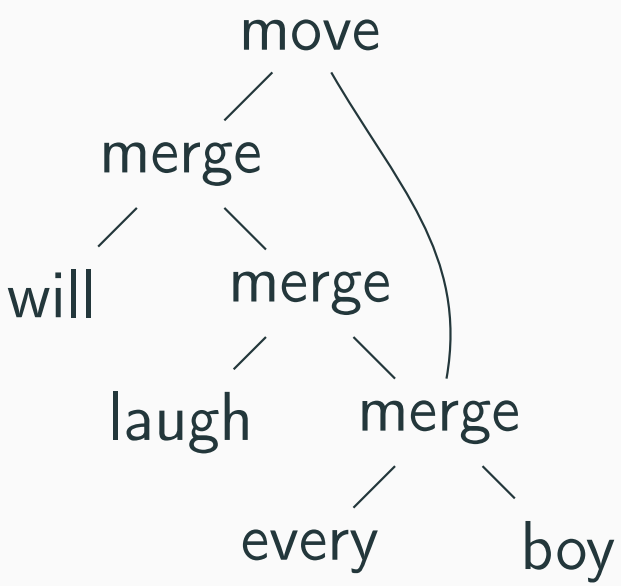
$$\frac{(\text{will}, \texttt{=v.+k.s}) \qquad \dfrac{(\text{laugh}, \texttt{=d.v}) \qquad \dfrac{(\text{every}, \texttt{=n.d.-k}) \qquad (\text{boy}, \texttt{n})}{(\text{every boy}, \texttt{d.-k})}}{(\text{laugh}, \texttt{v}), (\text{every boy}, \texttt{-k})}$$

An example



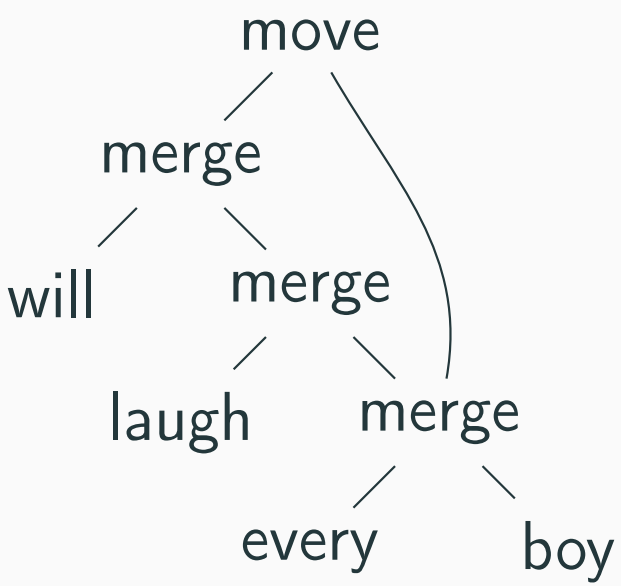
$$\dfrac{(\text{will}, \texttt{=v.+k.s}) \qquad \dfrac{(\text{laugh}, \texttt{=d.v}) \qquad \dfrac{(\text{every}, \texttt{=n.d.-k}) \qquad (\text{boy}, \texttt{n})}{(\text{every boy}, \texttt{d.-k})}}{(\text{laugh}, \texttt{v}), (\text{every boy}, \texttt{-k})}}{(\text{will laugh}, \texttt{+k.s}), (\text{every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

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



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

Semantics

What is necessary for semantics?

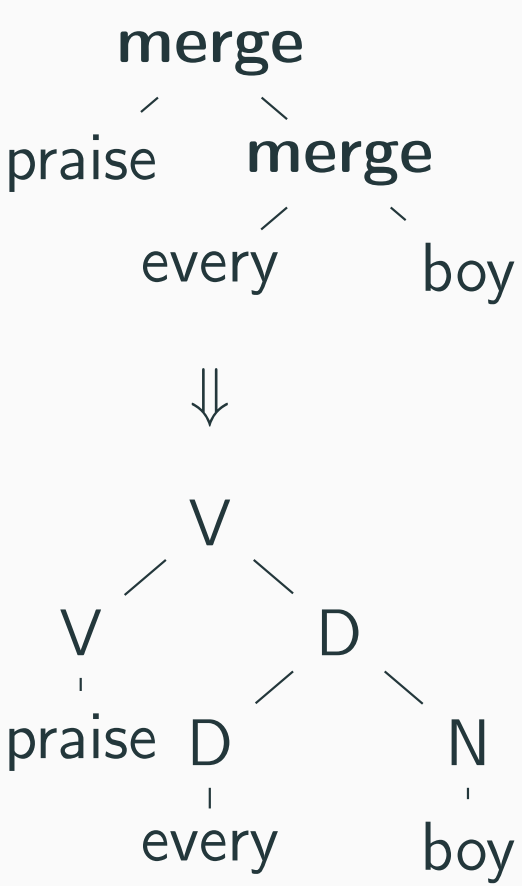
keep track of the unchecked syntactic features
(I won't talk about this here)

For PF
keep track of which phrases are still moving

but not of their internal structure

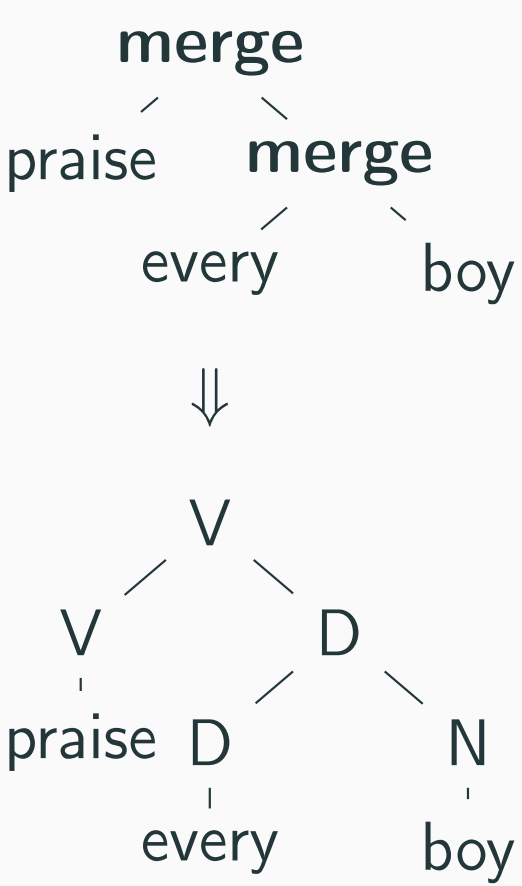
For LF
???

Revisiting meaningless parts



What is the contribution of *praise every boy* to expressions it is part of?

Revisiting meaningless parts



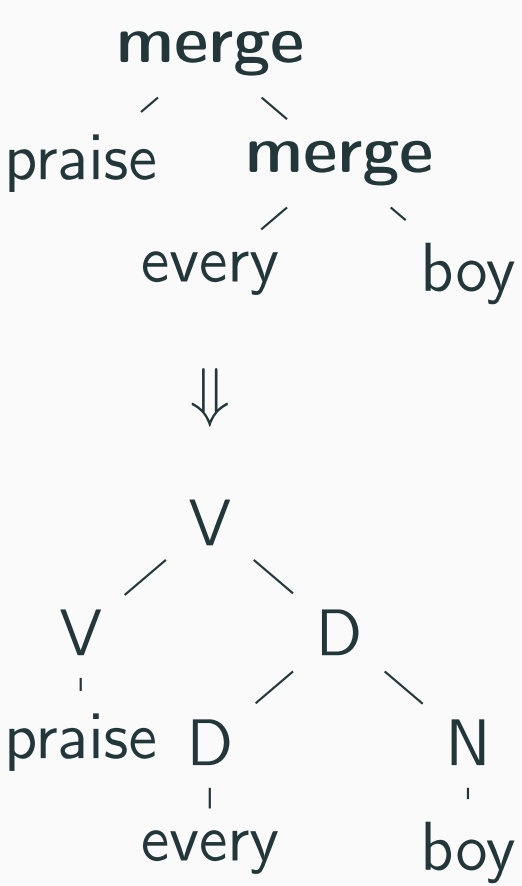
What is the contribution of *praise every boy* to expressions it is part of?

a quantifier part

$$\textbf{every}(\textbf{boy})(\lambda x. \ldots$$

and a property part $\textbf{praise}(x)$

Revisiting meaningless parts



What is the contribution of *praise every boy* to expressions it is part of?

a quantifier part

$$\textbf{every}(\textbf{boy})(\lambda x. \ldots$$

and a property part $\textbf{praise}(x)$

Let's write instead:

$$[\textbf{every}(\textbf{boy})]_x \vdash \textbf{praise}(x)$$

Notation and Operations

$$[\textbf{every(boy)}]_x \vdash \textbf{praise}(x)$$

The general case, with multiple stored quantifiers:

$$[Q_1]_{x_1}, \ldots, [Q_i]_{x_i} \vdash M$$

Notation and Operations

$$[\textbf{every(boy)}]_x \vdash \textbf{praise}(x)$$

The general case, with multiple stored quantifiers:

$$[Q_1]_{x_1}, \ldots, [Q_i]_{x_i} \vdash M$$

The entire point is to ignore what is stored

$$\frac{M}{\vdash M} \uparrow \qquad\qquad \frac{\Gamma \vdash M \qquad \Delta \vdash N}{\Gamma, \Delta \vdash M\ N} <*>$$

Working with Storage

$$\dfrac{\dfrac{\mathbf{seem}}{\vdash \mathbf{seem}}\uparrow \qquad \dfrac{\dfrac{\mathbf{Pass}}{\vdash \mathbf{Pass}}\uparrow \qquad \begin{array}{c}\vdots\\ [\mathbf{every}(\mathbf{boy})]_x \vdash \mathbf{praise}(x)\end{array}}{[\mathbf{every}(\mathbf{boy})]_x \vdash \mathbf{Pass}(\mathbf{praise}(x))}\,\text{<*>}}{[\mathbf{every}(\mathbf{boy})]_x \vdash \mathbf{seem}(\mathbf{Pass}(\mathbf{praise}(x)))}\,\text{<*>}$$

Building *praise every boy*

$$\dfrac{\dfrac{\textbf{praise}}{\vdash \textbf{praise}}\uparrow \qquad \dfrac{\dfrac{\textbf{every}}{\vdash \textbf{every}}\uparrow \qquad \dfrac{\textbf{boy}}{\vdash \textbf{boy}}\uparrow}{\vdash \textbf{every boy}}\,\text{<*>}}{\text{type mismatch!}}\,\text{<*>}$$

Building *praise every boy*

$$\dfrac{\textbf{praise}}{\vdash \textbf{praise}}\uparrow \qquad \dfrac{\dfrac{\textbf{every}}{\vdash \textbf{every}}\uparrow \qquad \dfrac{\textbf{boy}}{\vdash \textbf{boy}}\uparrow}{\vdash \textbf{every boy}}\,\text{<*>}$$

We want to 'insert a trace'

$$\dfrac{\vdash M}{[M]_x \vdash x}\,\Box$$

Building *praise every boy*

$$\dfrac{\textbf{praise}}{\vdash \textbf{praise}}\uparrow \qquad \dfrac{\dfrac{\dfrac{\textbf{every}}{\vdash \textbf{every}}\uparrow \quad \dfrac{\textbf{boy}}{\vdash \textbf{boy}}\uparrow}{\vdash \textbf{every boy}}\,\text{<*>}}{[\textbf{every boy}]_x \vdash x}\,\Box$$

We want to 'insert a trace'

$$\dfrac{\vdash M}{[M]_x \vdash x}\,\Box$$

Building *praise every boy*

$$\dfrac{\dfrac{\textbf{praise}}{\vdash \textbf{praise}}\uparrow \qquad \dfrac{\dfrac{\dfrac{\textbf{every}}{\vdash \textbf{every}}\uparrow \qquad \dfrac{\textbf{boy}}{\vdash \textbf{boy}}\uparrow}{\vdash \textbf{every boy}}\,\text{<*>}}{[\textbf{every boy}]_x \vdash x}\,\square}{[\textbf{every boy}]_x \vdash \textbf{praise}\ x}\,\text{<*>}$$

We want to 'insert a trace'

$$\dfrac{\vdash M}{[M]_x \vdash x}\,\square$$

Taking things out of storage

$$\dfrac{\dfrac{\mathbf{seem}}{\vdash \mathbf{seem}}\uparrow \qquad \dfrac{\dfrac{\mathbf{Pass}}{\vdash \mathbf{Pass}}\uparrow \qquad \begin{array}{c}\vdots\\ [\mathbf{every}(\mathbf{boy})]_x \vdash \mathbf{praise}(x)\end{array}}{[\mathbf{every}(\mathbf{boy})]_x \vdash \mathbf{Pass}(\mathbf{praise}(x))}\,\text{<*>}}{[\mathbf{every}(\mathbf{boy})]_x \vdash \mathbf{seem}(\mathbf{Pass}(\mathbf{praise}(x)))}\,\text{<*>}$$

Taking things out of storage

$$\dfrac{\dfrac{\mathbf{seem}}{\vdash \mathbf{seem}}\uparrow \qquad \dfrac{\dfrac{\mathbf{Pass}}{\vdash \mathbf{Pass}}\uparrow \qquad \begin{array}{c}\vdots\\ [\mathbf{every}(\mathbf{boy})]_x \vdash \mathbf{praise}(x)\end{array}}{[\mathbf{every}(\mathbf{boy})]_x \vdash \mathbf{Pass}(\mathbf{praise}(x))}\,\text{<*>}}{[\mathbf{every}(\mathbf{boy})]_x \vdash \mathbf{seem}(\mathbf{Pass}(\mathbf{praise}(x)))}\,\text{<*>}$$

retrieval

$$\dfrac{\Gamma, [M_i]_{x_i}, \Delta \vdash N}{\Gamma, \Delta \vdash M_i \oplus (\lambda x_i.N)}\,\langle\cdot\rangle^i_{\oplus}$$

Taking things out of storage

$$\dfrac{\dfrac{\dfrac{\mathbf{seem}}{\vdash \mathbf{seem}}\uparrow \qquad \dfrac{\dfrac{\mathbf{Pass}}{\vdash \mathbf{Pass}}\uparrow \qquad \begin{array}{c}\vdots\\ [\mathbf{every}(\mathbf{boy})]_x \vdash \mathbf{praise}(x)\end{array}}{[\mathbf{every}(\mathbf{boy})]_x \vdash \mathbf{Pass}(\mathbf{praise}(x))}\,\text{<*>}}{[\mathbf{every}(\mathbf{boy})]_x \vdash \mathbf{seem}(\mathbf{Pass}(\mathbf{praise}(x)))}\,\text{<*>}}{\vdash \mathbf{every}(\mathbf{boy})(\lambda x.\mathbf{seem}(\mathbf{Pass}(\mathbf{praise}(x))))}\,\langle\cdot\rangle^{1}_{\mathtt{FA}}$$

retrieval

$$\dfrac{\Gamma, [M_i]_{x_i}, \Delta \vdash N}{\Gamma, \Delta \vdash M_i \oplus (\lambda x_i.N)}\,\langle\cdot\rangle^{i}_{\oplus}$$

Manipulating Stores

pure

$$\frac{M}{\vdash M}\uparrow$$

apply

$$\frac{\Gamma \vdash M \qquad \Delta \vdash N}{\Gamma, \Delta \vdash M\ N}\ \text{<*>}$$

retrieve

$$\frac{\Gamma, [M_i]_{x_i}, \Delta \vdash N}{\Gamma, \Delta \vdash M_i \oplus (\lambda x_i.N)}\ \langle\cdot\rangle^i_\oplus$$

store

$$\frac{\vdash M}{[M]_x \vdash x}\ \square$$

Understanding stores

$$[M_1]_{x_1}, \ldots, [M_i]_{x_i} \ \vdash N$$
$$\Rightarrow \lambda k.k \ M_1 \ \ldots \ M_i \ (\lambda x_1, \ldots, x_i.N)$$

Example

$$[\textbf{every boy}]_x \ \vdash \textbf{praise} \ x$$
$$\Rightarrow \lambda k.k \ (\textbf{every boy}) \ (\lambda x.\textbf{praise} \ x)$$

Some examples

pure

$$\frac{M}{\vdash M}\uparrow$$

$$\Downarrow$$

$$\frac{M}{\lambda k.k\ M}\uparrow$$

$$M^{\uparrow} \equiv \lambda k.k\ M$$

storage

$$\frac{\vdash M}{[M]_x \vdash x}\Box$$

$$\Downarrow$$

$$\frac{\lambda k.k\ M}{\lambda k.k\ M\ (\lambda x.x)}\Box$$

$$\Box m \equiv \lambda k.m\ (\lambda M.k\ M\ (\lambda x.x))$$

More notation

idiom brackets

write $(\!| f \;\; a_1 \;\; \ldots \;\; a_i |\!)$

for $f^{\uparrow}$ <*> a_1 <*> ... <*> a_i

application

Forward $f \lhd a := f \;\; a$

Backward $a \rhd f := f \;\; a$

Minimalist semantics

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

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

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

for $\oplus \in \{\triangleleft, \triangleright\}$

Unpacking the notation

Recall that

$$\lambda m, n.(\!| m \lhd n |\!)$$

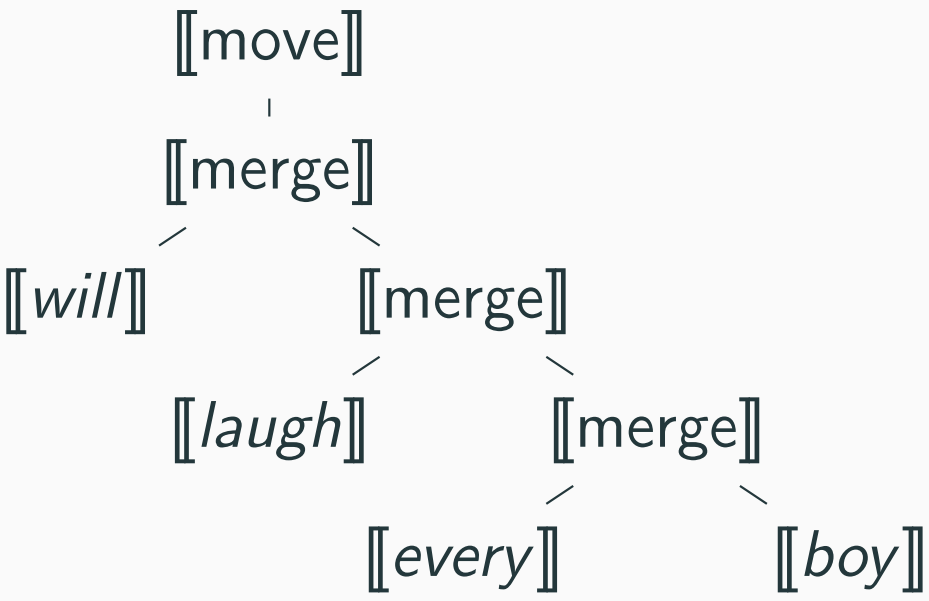
means

$$\lambda m, n.(\lhd)^{\uparrow} \mathbin{<*>} m \mathbin{<*>} n$$

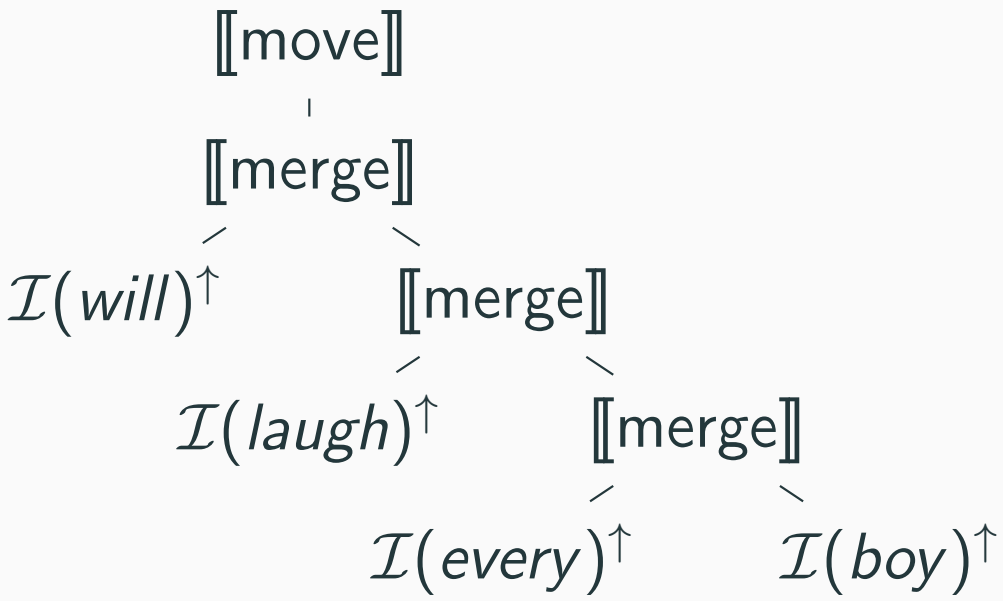
$$\dfrac{\dfrac{\dfrac{\lhd}{\vdash \lhd}\uparrow \qquad (m)}{\Gamma \vdash M}}{\Gamma \vdash M \lhd}\,{<*>} \qquad (n)$$

$$\dfrac{\Gamma \vdash M \lhd \qquad \Delta \vdash N}{\Gamma, \Delta \vdash M \lhd N}\,{<*>}$$

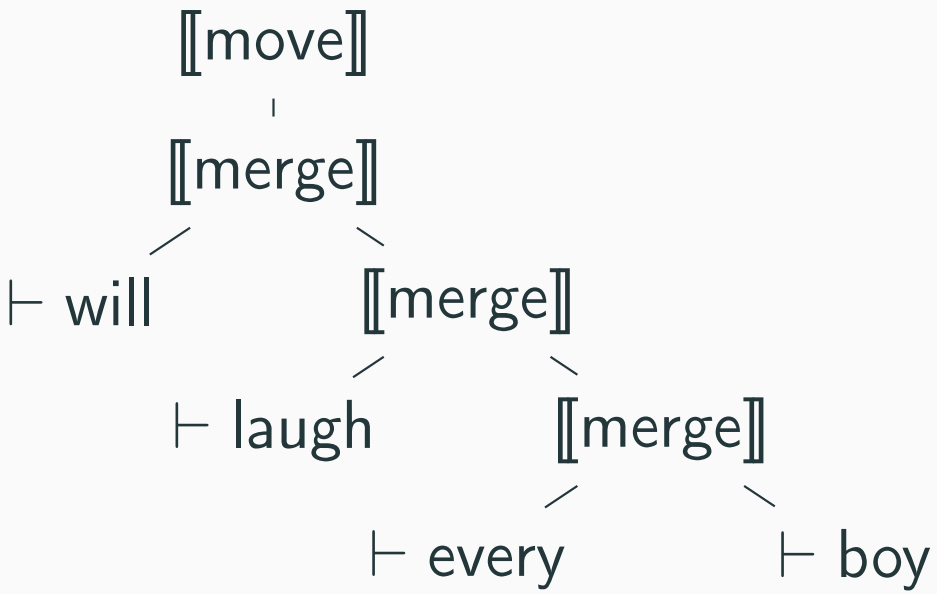
Every boy laughs



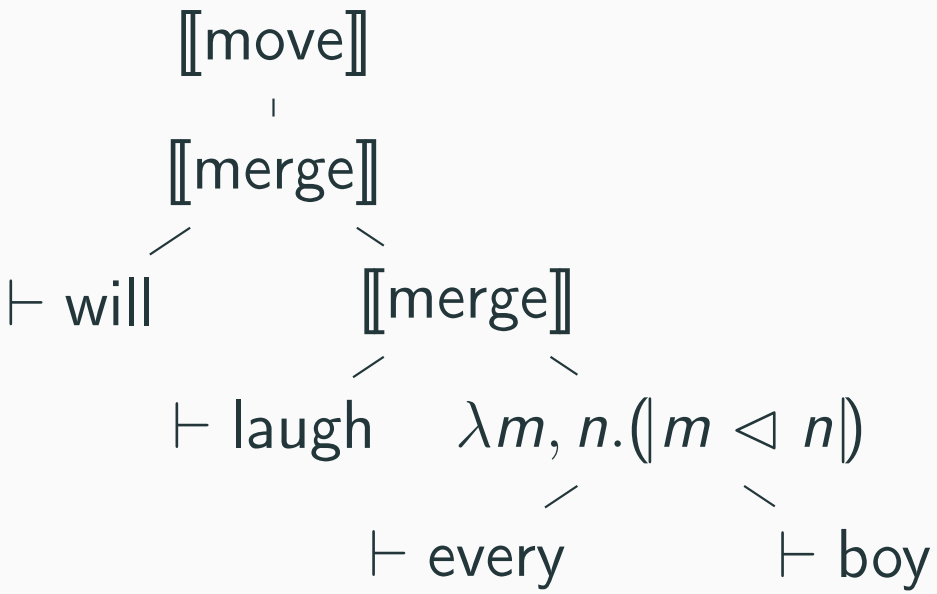
Every boy laughs



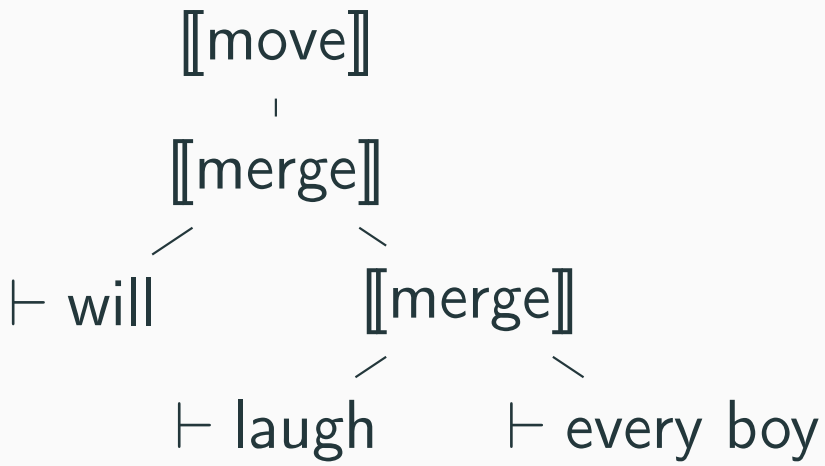
Every boy laughs



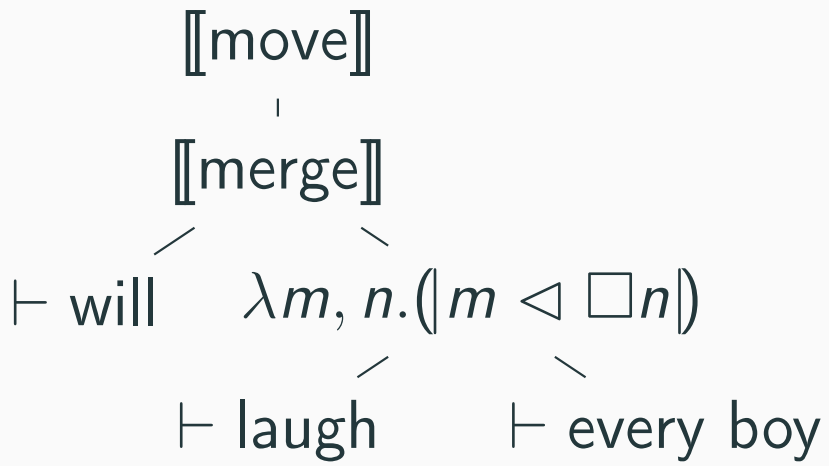
Every boy laughs



Every boy laughs



Every boy laughs



Every boy laughs

$$
\begin{tikzcd}
 & [\![\text{move}]\!] \arrow[d, dash] & \\
 & [\![\text{merge}]\!] \arrow[dl, dash] \arrow[dr, dash] & \\
\vdash \text{will} & & [\text{every boy}]_x \vdash \text{laugh } x
\end{tikzcd}
$$

Every boy laughs

$$[\![\text{move}]\!]$$

$$\lambda m, n.(\!| m \lhd n |\!)$$

$$\vdash \text{will} \qquad [\text{every boy}]_x \vdash \text{laugh } x$$

Every boy laughs

$$[\![\text{move}]\!]$$

$$[\text{every boy}]_x \vdash \text{will (laugh } x\text{)}$$

Every boy laughs

$$\lambda m.\langle m \rangle^{1}_{\triangleright}$$

$$[\text{every boy}]_x \vdash \text{will (laugh } x)$$

Every boy laughs

$$\vdash \text{every boy } (\lambda x.\text{will } (\text{laugh } x))$$

Compositional interfaces

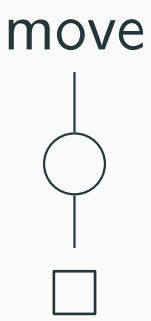
. . . allow for elimination of representational structure
Performance systems can 'use' the derivation in 'the wrong order' to construct the desired interface objects

Deforestation of parsing (Again)

$\square$

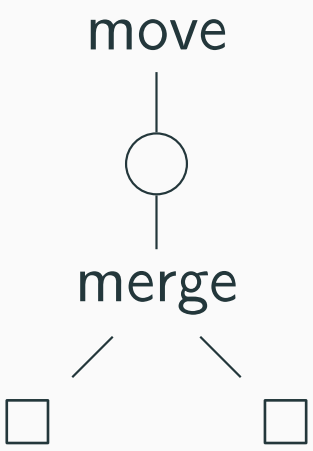
$\lambda x_{\square}.x_{\square}$

Deforestation of parsing (Again)



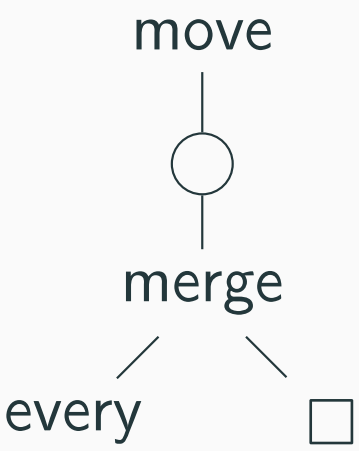
$$\lambda x_{\square}, f_{\bigcirc}.\langle f_{\bigcirc}\ x_{\square}\rangle^{k}_{\oplus}$$

Deforestation of parsing (Again)



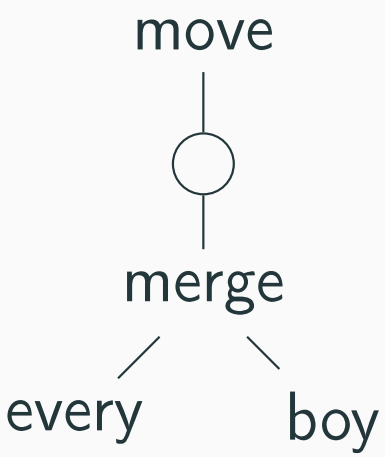
$$\lambda x_{\square}, y_{\square}, f_{\bigcirc}.\langle f_{\bigcirc}\ ((\!| x_{\square} \oplus y_{\square} |\!))\rangle^{k}_{\oplus}$$

Deforestation of parsing (Again)



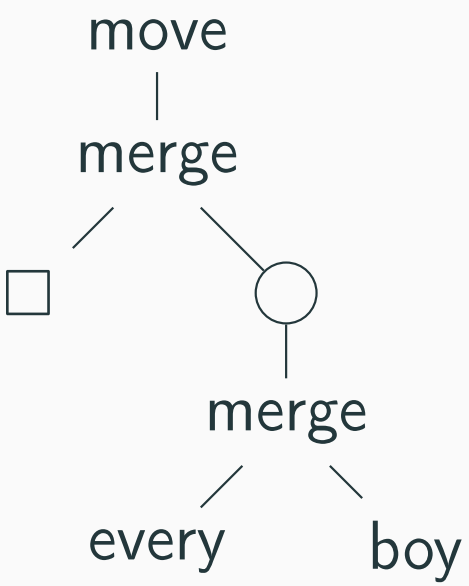
$$\lambda y_{\square}, f_{\bigcirc}.\langle f_{\bigcirc}\ ((\!|(\lambda z.\mathit{every} \oplus z)^{\uparrow}\ y_{\square}|\!))\rangle^{k}_{\oplus}$$

Deforestation of parsing (Again)



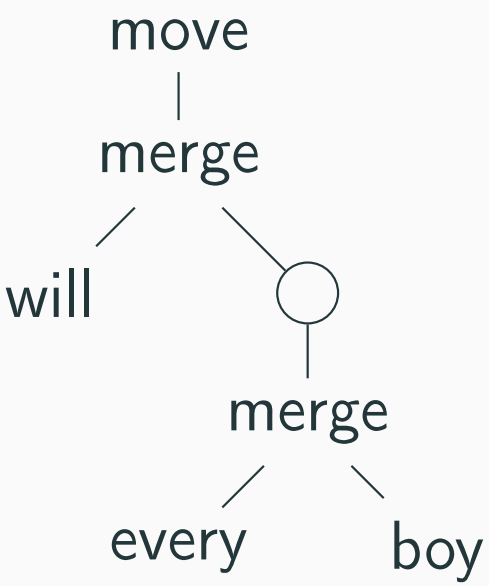
$$\lambda f_{\bigcirc}.\langle f_{\bigcirc}\ (\mathit{every}\ \mathit{boy}^{\uparrow})\rangle^{k}_{\oplus}$$

Deforestation of parsing (Again)



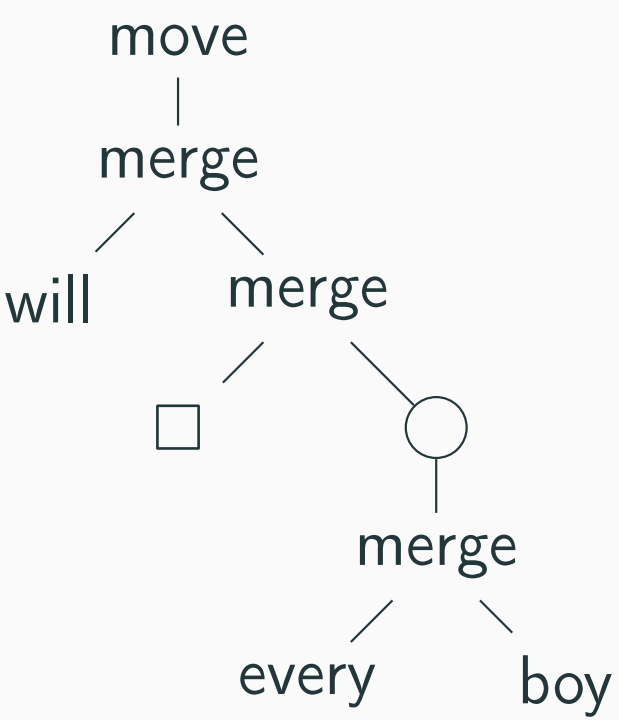
$$\lambda x_{\square}, f_{\bigcirc}.\langle(\!|x_{\square} \oplus (f_{\bigcirc}\ (every\ \ boy^{\uparrow}))|\!)\rangle^{k}_{\oplus}$$

Deforestation of parsing (Again)



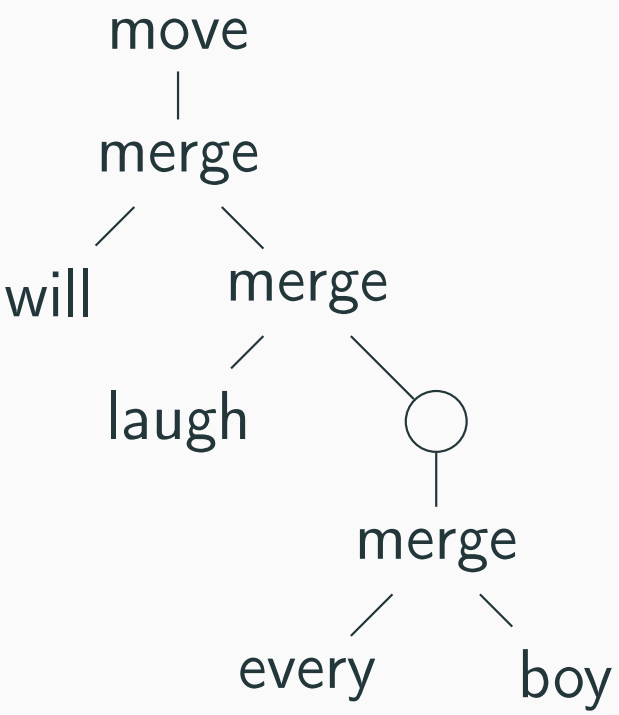
$$\lambda f_{\bigcirc}.\langle(\!|(\lambda z.\mathit{will} \oplus z)^{\uparrow}\ (f_{\bigcirc}\ (\mathit{every}\ \ \mathit{boy}^{\uparrow}))|\!)\rangle^{k}_{\oplus}$$

Deforestation of parsing (Again)



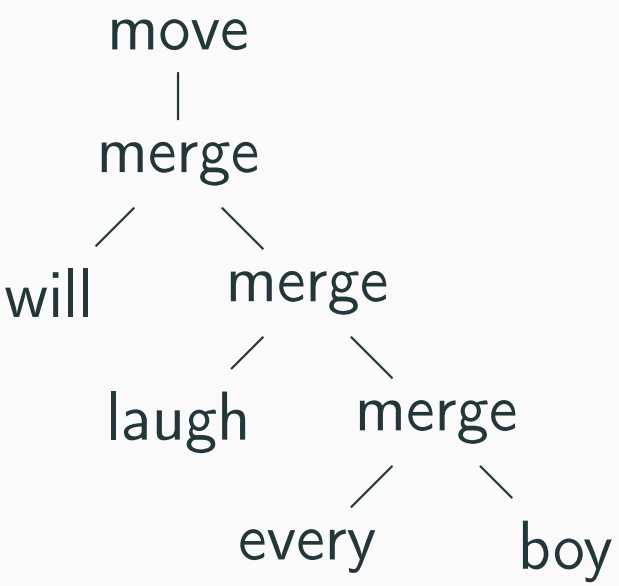
$$\lambda x_{\square}, f_{\bigcirc}.\langle (\!| (\lambda z.\mathit{will} \oplus z)^{\uparrow} \; (\!| x_{\square} \oplus (f_{\bigcirc} \; (\mathit{every} \;\; \mathit{boy}^{\uparrow}))|\!)|\!) \rangle^{k}_{\oplus}$$

Deforestation of parsing (Again)



$$\lambda f_{\bigcirc}.\langle(\!|(\lambda z.\mathit{will}\ (\mathit{laugh} \oplus z))^{\uparrow}\ (f_{\bigcirc}\ (\mathit{every}\ \mathit{boy}^{\uparrow}))|\!)\rangle^{k}_{\oplus}$$

Deforestation of parsing (Again)



$$\mathit{every}\ \mathit{boy}\ (\lambda z.\mathit{will}\ (\mathit{laugh}\ z))^{\uparrow}$$

Conclusions

Derivations have structure

- with clear identity conditions
- of just the kind we want to assign

Interface maps
focus our attention on what matters:

how much information (representation) we need to compositionally interpret our derivations

Derived structure
is a familiar trick to circumvent compositionality