Direct Compositionality

Greg Kobele

Universität Leipzig

Winter Semester, 2020

Derived structures

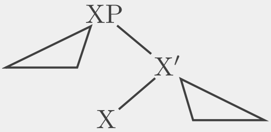
Minimalist expressions

- Expressions derived by MGs are *binary branching trees* with two *partial orderings* on internal nodes:

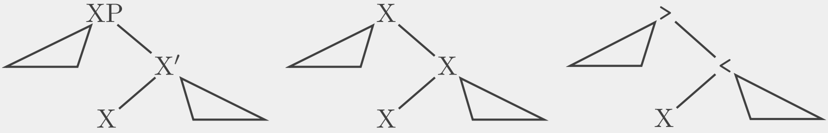
  linear precedence which sister is pronounced first

  projection which sister projects over the other

- Traditional way to represent this:



No more X-bar

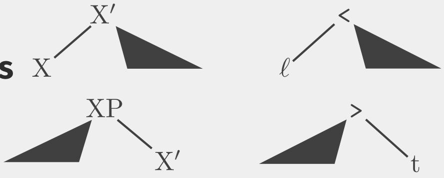


- the only *real* difference:

Spec and Comp

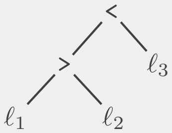
Complements

Specifiers



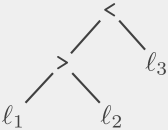
Heads

- The head of an expression *t* is
    1. *t* itself, if it is a leaf



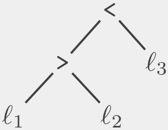
Heads

- The head of an expression t is
    1. t itself, if it is a leaf
    2. the head of t_1, if $t = {<}(t_1, t_2)$

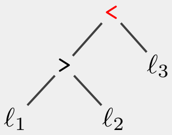


HEADS

- The head of an expression t is
  1. t itself, if it is a leaf
  2. the head of t_1, if $t = <(t_1, t_2)$
  3. the head of t_2, if $t = >(t_1, t_2)$

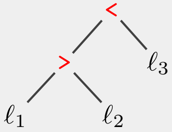


- The head of an expression t is
  1. t itself, if it is a leaf
  2. the head of t_1, if $t = <(t_1, t_2)$
  3. the head of t_2, if $t = >(t_1, t_2)$



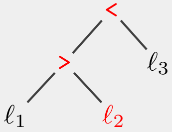
Heads

- The head of an expression t is
  1. t itself, if it is a leaf
  2. the head of t_1, if $t = <(t_1, t_2)$
  3. the head of t_2, if $t = >(t_1, t_2)$



Heads

- The head of an expression t is
  1. t itself, if it is a leaf
  2. the head of t_1, if $t = <(t_1, t_2)$
  3. the head of t_2, if $t = >(t_1, t_2)$



Basic Grammar

Putting things together

derivational (or *algebraic*) perspective

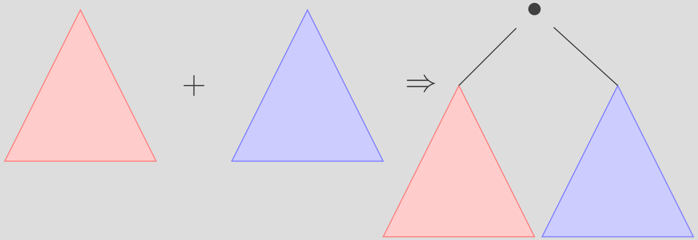
basic elements *lexical items*

ways of building complex things from simpler things *grammatical operations*

Language of the grammar

is simply the set of things that can be built from basic elements using the available operations

Merge



English

- *John laughed*
- $*$ *laughed John*

`merge`

- should be defined on *John* + *laughed*
- but not on *laughed* + *John*

Everyone's solution:

operations are sensitive to the categories of the basic elements

- *John* is a DP

Notation

α **is a X**	α has feature x
α **combines with a X**	
on the left	x=
on the right	=x

- *John* is a DP ⇝ *John* has feature d
- *laughed* combines with a DP (on the left) to give an S ⇝ *laughed* has features d= and s

Categories are structured

***laughed* isn't an S until it has combined with a DP**

laughed has first feature d=, and second feature s

Lexical items

Feature bundles

A list of features (separated with periods)

d=.s

Lexical items

pairs of

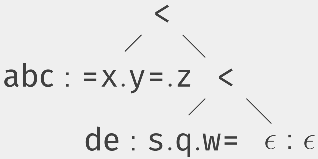
- *morpho-phonological* info (I'm the lexeme *laughed*)
- *categorial* info (my feature bundle is d=.s)

written laughed : d=.s

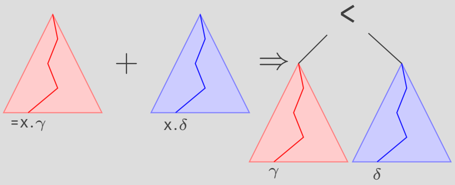
Revisiting derived structures

Leaves

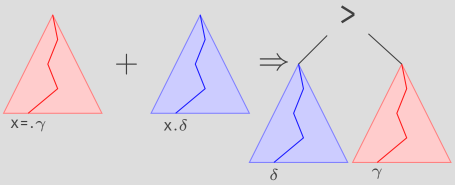
leaves are similarly pairs of strings and feature bundles



On the right



On the left



Feature checking

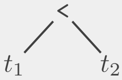
- Leaves of trees contain sequences of features.
  - determine whether an operation can apply
- Once an op applies, features are *checked*
  - here: deleted
- Ops are ‘trying’ to remove features from trees
  - An exp is well-formed (‘complete’) iff
    - head has only feature in tree
    - it is x (for some *x*)

- given t, we write t^f to denote the result of adding f as the first feature on the head of t:
  - if the head of t is $\sigma : \delta$, then t^f is the tree just like t except that its head is $\sigma : f.\delta$
- t displays feature f, if the head of t is $\sigma : f.\delta$
  - t^f displays feature f
- Checking the first feature of t^f gives us t

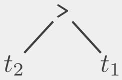
Merge, again

- $\langle t, t' \rangle \in \text{dom}(\textbf{merge})$ iff
  - $t = t_1^{=\text{x}}$ and $t' = t_2^{\text{x}}$, or
  - $t = t_1^{\text{x}=}$ and $t' = t_2^{\text{x}}$

$\textbf{merge}(t_1^{=\text{x}}, t_2^{\text{x}}) =$



$\textbf{merge}(t_1^{\text{x}=}, t_2^{\text{x}}) =$



English Auxiliaries

Simple Sentences I

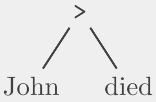
We begin with simple intransitive sentences, such as the below.

1. John died.
2. John will die.
3. John had died.
4. John has been dying.

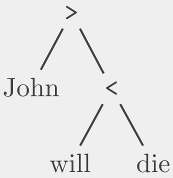
Structural Assumptions

We treat these sentences as being divided into a subject (*John*), and a predicate (the rest). The predicate is treated as right branching, with elements to the left projecting over those to their right.

1. John died.



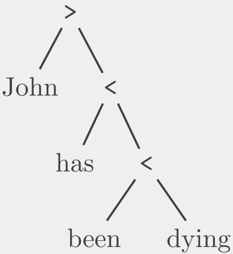
2. John will die.



Simple Sentences II

A slightly bigger example…

3. John has been dying.



A Grammar for this Fragment

- We want a grammar to generate these expressions.
- To specify a grammar, we need to specify four things:

  The features which features we will use in our grammar

  The lexicon which syntactic feature sequences are assigned to which words

  The grammatical operations currently, this will just be merge, so I will leave it implicit in the following

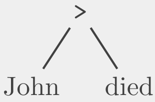
  The start category what is the category of complete sentences

  - Breaking with tradition, I will call the start category s – it reminds me of {s}entence, as well as {s}tart!

- Thus, all that is left is to determine the *features* we will use, and the *lexical items* we have

Grammatical Reasoning I

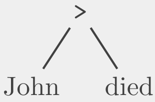
Given an expression like the below, we know that its head must have category s, and that no other leaves may have syntactic features.



- What features must *John* and *died* have in order to combine into the structure above of category s?

Grammatical Reasoning I

Given an expression like the below, we know that its head must have category s, and that no other leaves may have syntactic features.

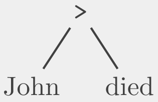


- What features must *John* and *died* have in order to combine into the structure above of category s?
- We can only build the above structure from lexical items of the following shape:

John : x   died : x=.s

Grammatical Reasoning I

Given an expression like the below, we know that its head must have category s, and that no other leaves may have syntactic features.



- What features must *John* and *died* have in order to combine into the structure above of category s?
- We can only build the above structure from lexical items of the following shape:

$$\text{John} : \mathsf{x} \qquad \text{died} : \mathsf{x{=}.s}$$

- What should ‘x’ be? It doesn’t matter! All that matters is whether two features match, not what they are called. Let’s take ‘x’ to be ‘d’ (for ‘DP’), as a nod to tradition.

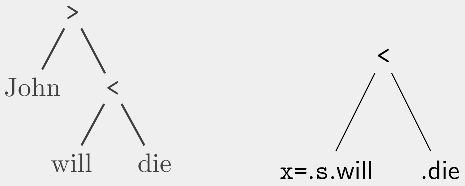
Grammatical Reasoning II

We can perform the same line of reasoning on the structure on the left below, too.

- The structure on the left must be the result of merging a lexical item John : x with the structure on the right

Grammatical Reasoning II

We can perform the same line of reasoning on the structure on the left below, too.

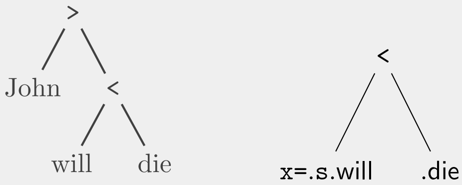


- The structure on the left must be the result of merging a lexical item John : x with the structure on the right
- This righthand structure then must be the result of merging the following two lexical items.

will : =y.x=.s   die : y

Grammatical Reasoning II

We can perform the same line of reasoning on the structure on the left below, too.



- The structure on the left must be the result of merging a lexical item John : x with the structure on the right
- This righthand structure then must be the result of merging the following two lexical items.

will : =y.x=.s   die : y

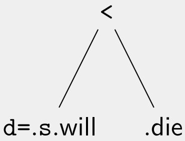
- As feature names don't matter, lets call 'y' 'v', and 'x' 'd'.

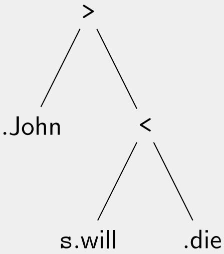
John : d   will : =v.d=.s   die : v

Is this right? – A sanity check

So we have decomposed the tree we assigned to the sentence *John will die* into the three lexical items below – Let's make sure they allow us to derive this sentence!

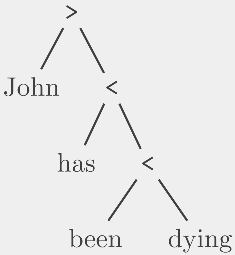
John : d   will : =v.d=.s   die : v

1. **merge**(will : =v.d=.s, die : v) = 

2. **merge**(1, John : d) = 

Grammatical Reasoning III

In the same way, from a structure like that below, we obtain the following lexical items:



John : `d`	has : `=perf.d=.s`
been : `=prog.perf`	dying : `prog`

Fragment Analysed

In this way, from the sentences below, we arrive at the following set of lexical items, which determine a grammar.

John dies \ John died \ John will die \ John has died \ John had died \ John is dying

John was dying \ John has been dying \ John had been dying \ John will be dying \ John will have died \ John will have been dying

die : v	will : =v.d=.s	is : =prog.d=.s
died : perf		was : =prog.d=.s
dying : prog	have : =perf.v	be : =prog.v
died : d=.s	has : =perf.d=.s	been : =prog.perf
dies : d=.s	had : =perf.d=.s	

Analysis Criticised

die : v	will : =v.d=.s	is : =prog.d=.s
died : perf		was : =prog.d=.s
dying : prog	have : =perf.v	be : =prog.v
died : d=.s	has : =perf.d=.s	been : =prog.perf
dies : d=.s	had : =perf.d=.s	

- These lexical items are highly redundant:
  1. all of the *be* forms select for something in the progressive
  2. all the *have* forms something in the perfective
  3. all and only the tensed forms (*died*, *dies*, *has*, *had*, …) select an argument
- Whenever a new verb is added to the language, we need to add five new lexical items:

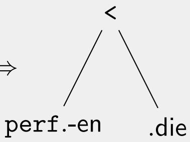
laugh : v	laughed : perf
laughing : prog	laughed : d=.s
laughs : d=.s	

Head Movement

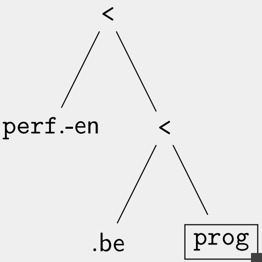
Morphological Decomposition

- Let's begin with lexical items of category `perf` (*died* and *been*, but also *broken*,…)
- Instead of lexical items, think of them as having been built from the perfective suffix *-en* as well as a verb (*die*) or auxiliary (*be*)

perf.died ⟹



=prog.perf.been ⟹



Morphological Composition

If we syntactically decompose *died* into a root verb *die* and an affix -*en*, how do we end up pronouncing it as one word?

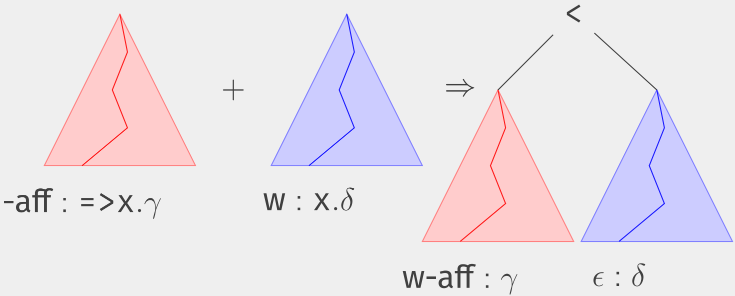
Post-syntactic morphology

- Distributed Morphology
- Mirror theory
- Head movement

- These theories presuppose that certain syntactic configurations can give rise to morphological composition
- (at least) `head` - `complement`

MW Formation during Merge

- Only from a complement



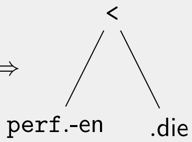
Must specify how w-aff is pronounced

need a real theory of morphology

here just a list

Now we can assign features to our affixes:

perf.died $\Longrightarrow$



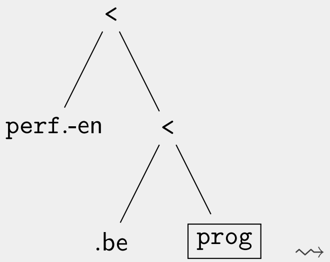
$\rightsquigarrow$

die : x =>x.perf.-en
die-en $\mapsto$ died

Syntactic Decomposition

Now we can assign features to our affixes:

=prog.perf.been $\Longrightarrow$



be : =prog.x
=>x.perf.-en

be-en $\mapsto$ been

More Decomposition

has : =>perf.d=.s	⇝	have : =perf.x -s : =>x.d=.s have-s ↦ has
had : =>perf.d=.s	⇝	have : =perf.x -ed : =>x.d=.s have-ed ↦ had
is : =>prog.d=.s	⇝	be : =prog.y -s : =>y.d=.s be-s ↦ is
was : =>prog.d=.s	⇝	be : =prog.y -ed : =>y.d=.s be-ed ↦ was

More Redundancy

- Note though that now we have two versions each of the present and past tense morphemes:

-s : =>x.d=.s	-ed : =>x.d=.s
-s : =>y.d=.s	-ed : =>y.d=.s

- There are three options:
  1. collapse x and y into a third category (perhaps v)

-s : =>v.d=.s	-ed : =>v.d=.s

  2. allow an isa-relationship to obtain between x and y

-s : =>y.d=.s	-ed : =>y.d=.s
ϵ : =>x.y	

  3. allow an isa-relationship to obtain between x and y

-s : =>x.d=.s	-ed : =>x.d=.s
ϵ : =>y.x	

Distributional arguments

- Note that whenever *have* and *be* occur together, *have* always precedes *be*:
  - John has been dying
  - ∗John is having died
  - John will have been dying
  - ∗John will be having died
- and that, whenever *be* occurs incorporated into -*s* or -*ed*, *have* is not present:
  - John is dying
  - ∗John is having died
  - John was dying
  - ∗John was having died
- These facts argue against the first option (treating *have* and *be* as having the same category)

More Redundancy again

- We have the same difficulty with the perfective -*en*!

  `-en` : `=>v.perf`   `-en` : `=>y.perf`

- There are again three options:

  1. collapse v and y together:

     `-en` : `=>v.perf`

  2. allow an isa-relationship to obtain between v and y:

     `-en` : `=>v.perf`
     ϵ : `=>v.y`

  3. allow an isa-relationship to obtain between v and y:

     `-en` : `=>y.perf`
     ϵ : `=>y.v`

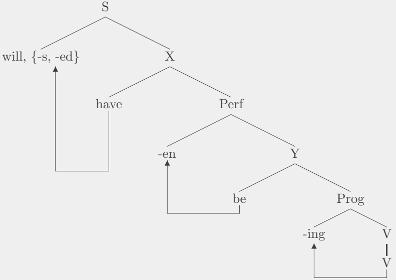
More distributional arguments

- Note that whenever *be* and *die* occur together, *be* always precedes *die*:
  - John has been dying
  - $*$John has died be
  - John will have been dying
  - $*$John will have died be
- and that, whenever *die* occurs incorporated into -*en*, *be* is not present:
  - John has died
- The first option again is seen to be incorrect
- Note that if we assume that v isa y, and that y isa x, then we predict that *die* can incorporate into -*s* and -*ed*!
  - John dies
  - John died

Head movement in the auxiliary system

Following similar reasoning, we arrive at the lexicon below:

will : =x.d=.s	have : =perf.x	be : =prog.y	die : v
-s : =>x.d=.s	-en : =>y.perf	-ing : =>v.prog	
-ed : =>x.d=.s	ϵ : =>y.x	ϵ : =>v.y	



- To add a new verb, we add just a single lexical item: laugh : v

Decompositional Methodology

Whenever we have a lexical item

$$\text{uv} : \alpha\beta$$

We can split it up into two:

$$\text{u} : \alpha.\text{x} \qquad \text{-v} : \text{=>x}.\beta$$

Proliferation of functional projections

is simply one of the natural moves in this architecture

Raising to Subject

Basic Alternation

- Verbs like *seem* allow for the following alternation:
  1. It will seem that John laughed
  2. John will seem to have laughed
- New lexical items:

it : d	to : =x.i	seem : =i.v
	that : =s.c	seem' : =c.v

- Observations:
  1. *it* as main clause subject requires finite *that*-complement
  2. DP as main clause subject forbids finite *that*-complement

Problem

how to transmit information from one point to another

Analytical Possibilities

1. Syntactic feature percolation

$$\text{seem'} : \texttt{=c.v}' \quad \text{will}_2 : \texttt{=x}'.\texttt{d}'\texttt{=.s} \quad \text{it} : \texttt{d}'$$

$$\text{seem} : \texttt{=i.v} \quad \text{will} : \texttt{=x.d=.s} \quad \text{John} : \texttt{d}$$

Analytical Possibilities

1. Syntactic feature percolation

$$\text{seem'} : =\texttt{c}.\text{v}' \quad \text{will}_2 : =\text{x}'.\text{d}'=.\text{s} \quad \text{it} : \text{d}'$$

$$\text{seem} : =\texttt{i}.\text{v} \quad \text{will} : =\text{x}.\text{d}=.\text{s} \quad \text{John} : \text{d}$$

2. Semantic type

$$\text{seem'} : tt \quad \text{will'} : tt \quad \text{it} : tt$$

$$\text{seem} : (et)et \quad \text{will} : (et)et \quad \text{John} : e$$

Analytical Possibilities

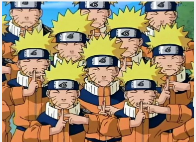
1. Syntactic feature percolation

seem’ : =c.v′	will$_2$: =x′.d′=.s	it : d′
seem : =i.v	will : =x.d=.s	John : d

2. Semantic type

seem’ : *tt*	will’ : *tt*	it : *tt*
seem : (*et*)*et*	will : (*et*)*et*	John : *e*

3. Ninja technique (Kage Bunshin no Jutsu):
   Main clause subject is in two places at once. Must satisfy properties of both positions to be well-formed.



RAISING

$[\![$*John will laugh*$]\!] = \text{WILL}(\text{LAUGH}(\text{JOHN}))$

Surface	will : =x.d=.s	laugh : v
Deep	will : =x.s	laugh : d=.v

Raising

⟦*John will laugh*⟧ = WILL(LAUGH(JOHN))

Surface	will : =x.d=.s	laugh : v
Deep	will : =x.s	laugh : d=.v

Not quite right:

Ninja	will : =x.d=.s	laugh : d=.v

Deep vs Surface Positions

a DP should have two positions

1. where it is *base generated* (via `merge`)
2. where it *appears on the surface*

it must be syntactically active after `merge`

- `merge` deletes the d feature
- so it must have another feature

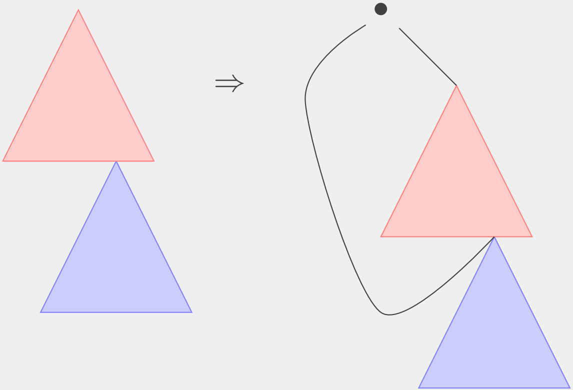
we don't currently have a way of checking features after something is `merged`

so we need another operation

Move (MDS)

blue is a *maximal projection*

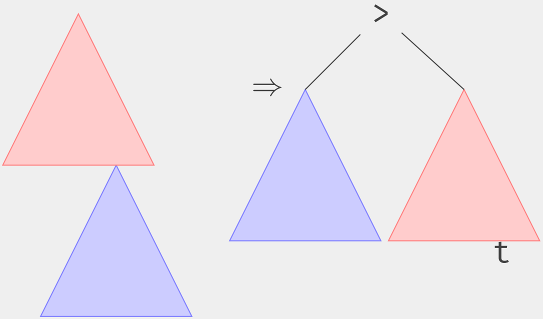
blue is (literally) in two places at once



Move (traces)

t

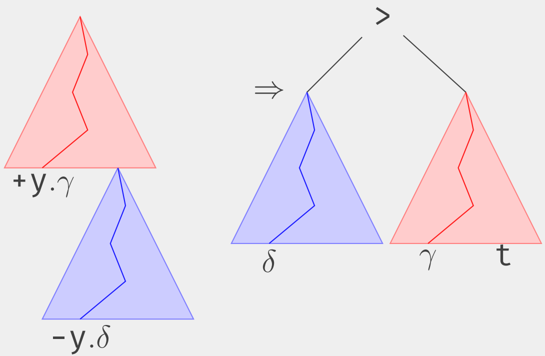
- stands for ϵ :
- a *trace* is just a silent leaf with no features



Move (features)

Want to *control* when move can apply

+y	move something to me
-y	move me somewhere



Deep vs Surface Positions (II)

a DP should have two positions

1. where it is *base generated* (via `merge`)
2. where it *appears on the surface* (via `move`)

it must be syntactically active after `merge`

- `merge` deletes the d feature
- so it must have another feature, `-k`

A DP feature bundle: `d.-k`

d	how to be well-formed in the base position
-k	how to be well-formed in the surface position

Raising (II)

$[\![\textit{John will laugh}]\!] = \textsc{will}(\textsc{laugh}(\textsc{john}))$

Surface	will : =x.d=.s	laugh : v
Deep	will : =x.s	laugh : d=.v

Raising (II)

⟦*John will laugh*⟧ = WILL(LAUGH(JOHN))

Surface	will : =x.d=.s	laugh : v
Deep	will : =x.s	laugh : d=.v

Here we go:

Ninja	will : =x.+k.s	laugh : d=.v

Updating the lexicon

- The d= feature on the lexical items *will*, *-s*, and *-ed* were originally intended to introduce the predicate's argument in its surface position. Now the argument is already present, but not in its surface position.
- We thus assign the tense lexical items the type:

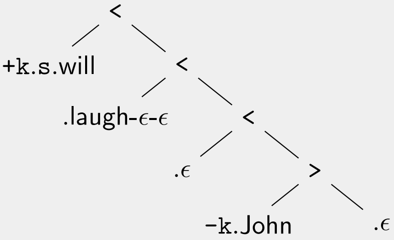
  =x.+k.s

  This indicates that a lexical item like *will* provides a *surface* position (for something with a -k feature, like a DP)
- Crucially, *to* doesn't provide such a surface position:

  to : =x.i

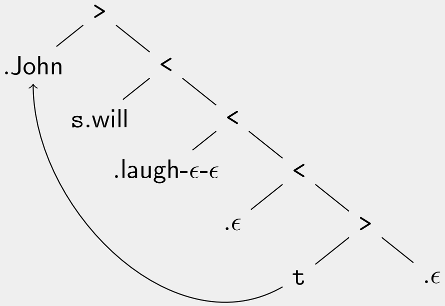
Short Raising to Subject

Surface subjects in simple intransitive sentences raise to this position from within the vP:



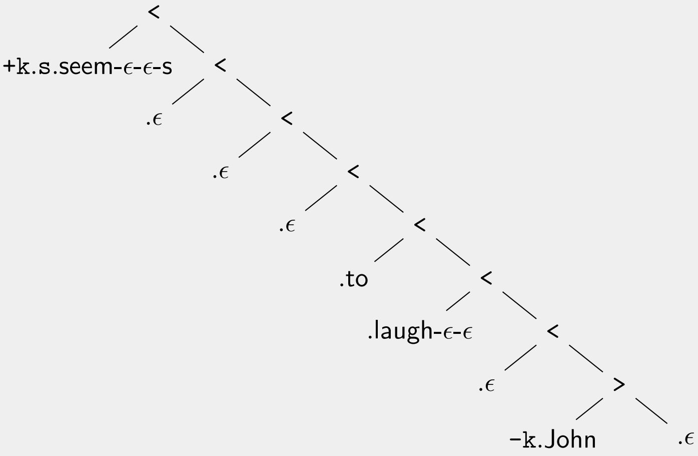
Short Raising to Subject

Surface subjects in simple intransitive sentences raise to this position from within the vP:

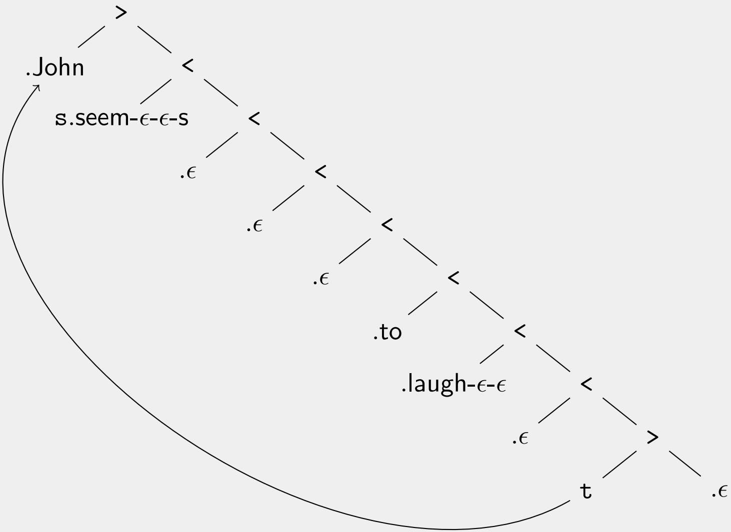


Long Raising to Subject

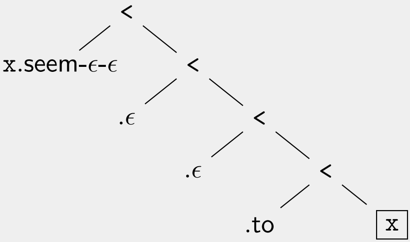
The same is true of surface subjects of *seem*:



The same is true of surface subjects of *seem*:

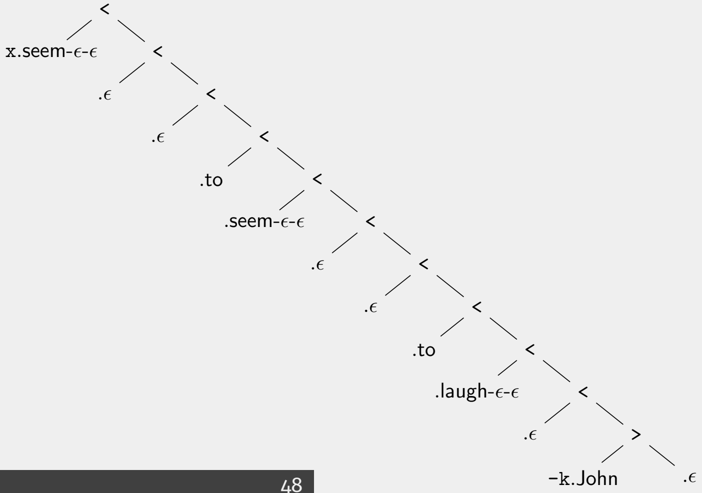


Note that we can add as many *seem to*'s as we want; only after we add a tense item do we trigger raising of the embedded DP:



Really Long Raising to Subject

Note that we can add as many *seem to*'s as we want; only after we add a tense item do we trigger raising of the embedded DP:



Alternations

How do we deal with the alternation:

1. It seems that John laughed
2. John seems to have laughed

- *it* appears as the subject of tensed clauses without semantic subjects
  - *it seems …*
  - *it rains*

From the perspective of the analysis,

it appears whenever we have a +k feature with nothing to check it

From the perspective of the analysis,

it appears whenever we have a `+k` feature with nothing to check it

Therefore:

it needs to have a feature bundle ending in `-k`

because it doesn't have the same distribution as a regular DP, we don't give it the same category:

it : `expl.-k`

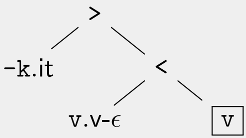
GETTING *IT* TO APPEAR

it : `expl.-k`

- We can treat *it* as a vP adjunct

ϵ : `=>v.expl=.v`

a vP is something which can optionally select an `expl`



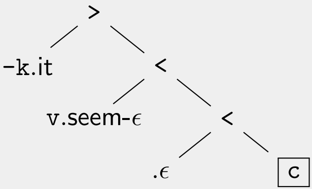
GETTING *IT* TO APPEAR

it : expl.-k

- We can treat *it* as a vP adjunct

ϵ : =>v.expl=.v

a vP is something which can optionally select an expl



SUPERRAISING

We currently generate the following sentence type:

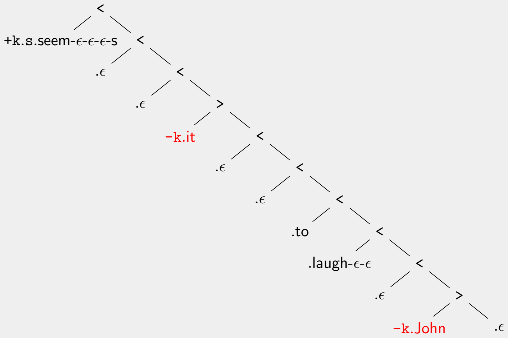
John$_i$ *is believed that it seems to* $\texttt{t}_i$ *laugh.*

In other words, nothing enforces the *last resort* character of *it*.

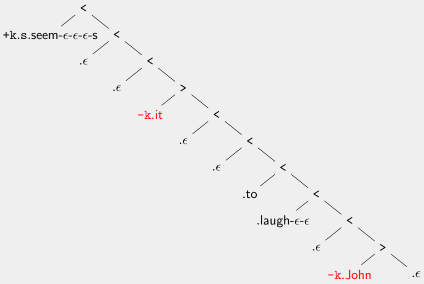
Superraising Derivation

John$_i$ is believed that it seems to t_i laugh.

- Right before moving *it*, we have:



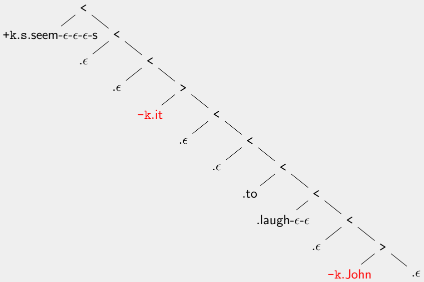
Toward Blocking Superraising



Some options:

1. should always move the lower candidate
2. should never have to make a choice
3. treat *it* differently

Toward Blocking Superraising



Some options:

1. should always move the lower candidate
2. should never have to make a choice
3. treat *it* differently

Give up

Don’t make a choice

- whoever you don’t choose will move farther than if you had chosen them (shortest move flavor)
- it’s easy (no need to calculate or compare)
- it works (pretty well)
- it is formally awesome (MCS)

SMC

move is only defined if there is exactly one maximal projection with the relevant first feature

Constraints on movement

Attract Closest more generally, *make deterministic*
Specifier Island can't extract from specifiers
SMC more generally, *at most k movees*

Results

with SpI-mv recursively enumerable (K. & Michaelis)

Claim

This really matters!

Constraints on movement

Attract Closest more generally, *make deterministic*
Specifier Island can't extract from specifiers
SMC more generally, *at most k movees*

Results

with SpI-mv recursively enumerable (K. & Michaelis)

with Nothing / Attract Closest not semilinear
at least 2-ExpSpace Hard, maybe undecidable (Salvati)

Claim

This really matters!

Constraints on movement

Attract Closest more generally, *make deterministic*
Specifier Island can't extract from specifiers
SMC more generally, *at most k movees*

Results

with SpI-mv recursively enumerable (K. & Michaelis)

with Nothing / Attract Closest not semilinear
at least 2-ExpSpace Hard, maybe undecidable (Salvati)

with SMC MCFL (Michaelis)

Claim

This really matters!

Constraints on movement

Attract Closest more generally, *make deterministic*
Specifier Island can't extract from specifiers
SMC more generally, *at most k movees*

Results

with SpI-mv recursively enumerable (K. & Michaelis)

with Nothing / Attract Closest not semilinear
at least 2-ExpSpace Hard, maybe undecidable (Salvati)

with SMC MCFL (Michaelis)

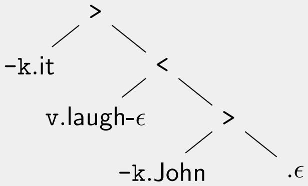
with SpI-mrg & SMC mb-MCFL (Michaelis)

Claim

This really matters!

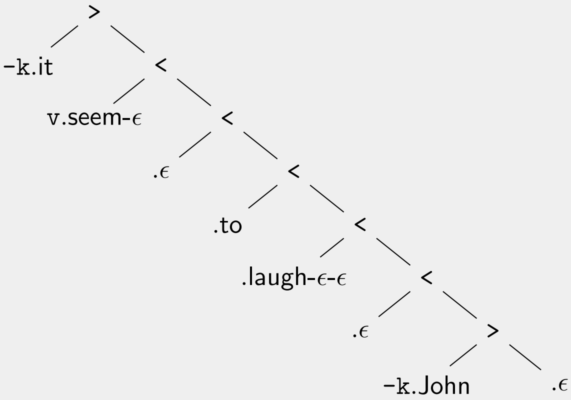
SMC at work

- this expression is generated by our analysis
- it has two subtrees displaying -k
- can never become a complete expression



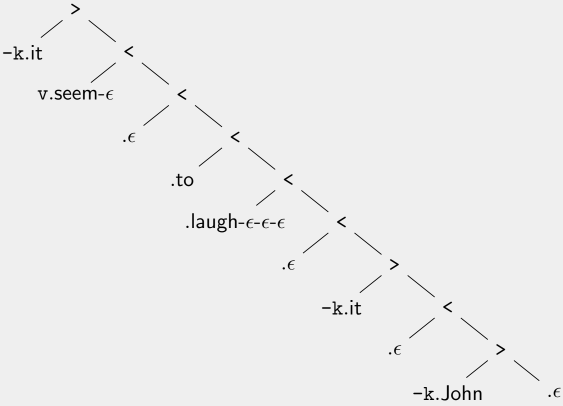
SMC at work

- this expression is generated by our analysis
- it has two subtrees displaying -k
- can never become a complete expression



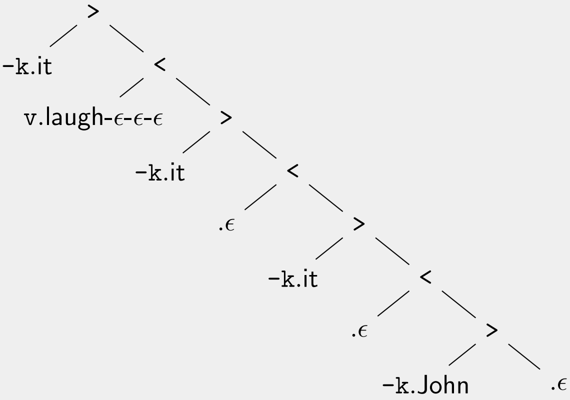
More work

- Even crazier things are now in the closure of our lexicon under the generating functions.
- They are all blocked by the SMC from ever becoming complete expressions.



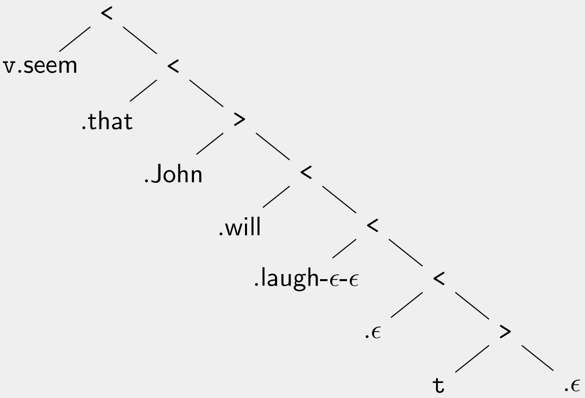
More work

- Even crazier things are now in the closure of our lexicon under the generating functions.
- They are all blocked by the SMC from ever becoming complete expressions.

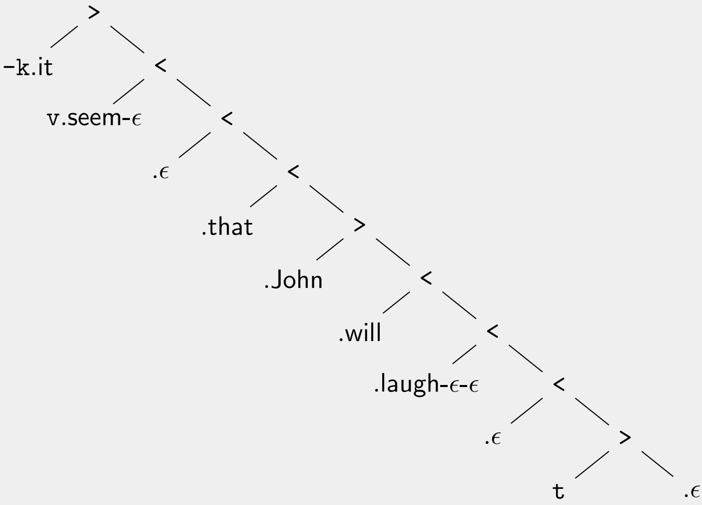


DERIVING *IT*

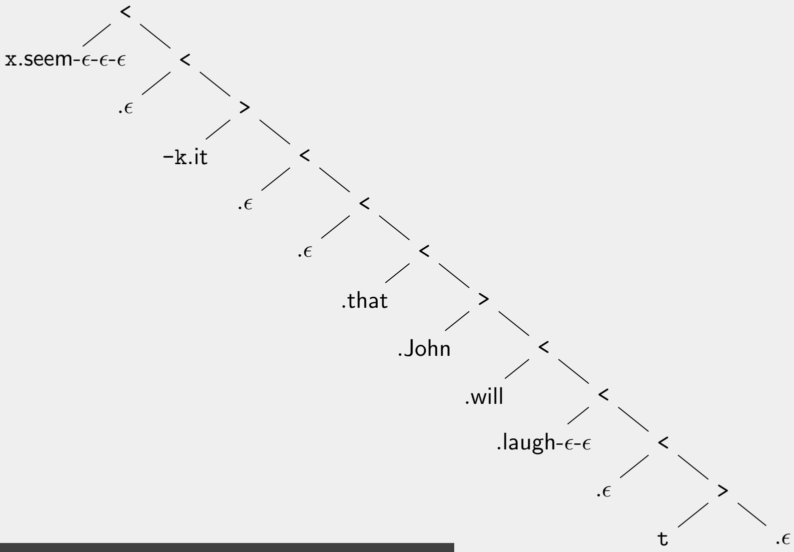
We assign the *it*-sentence the following structure:



We assign the *it*-sentence the following structure:

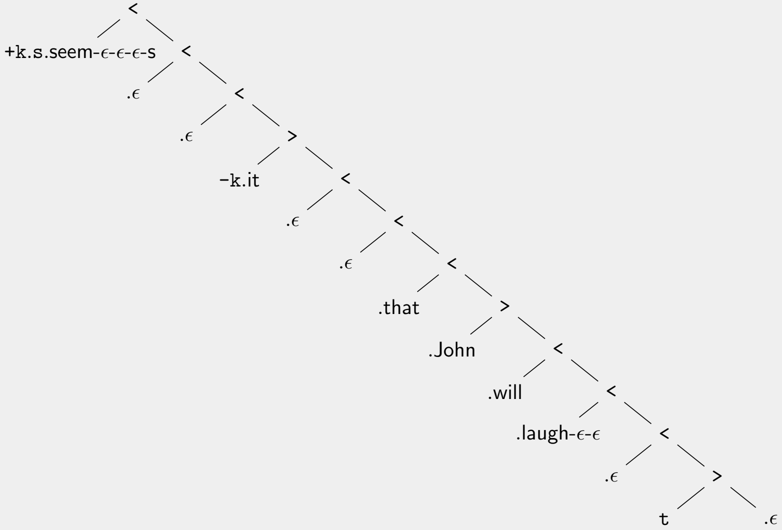


We assign the *it*-sentence the following structure:



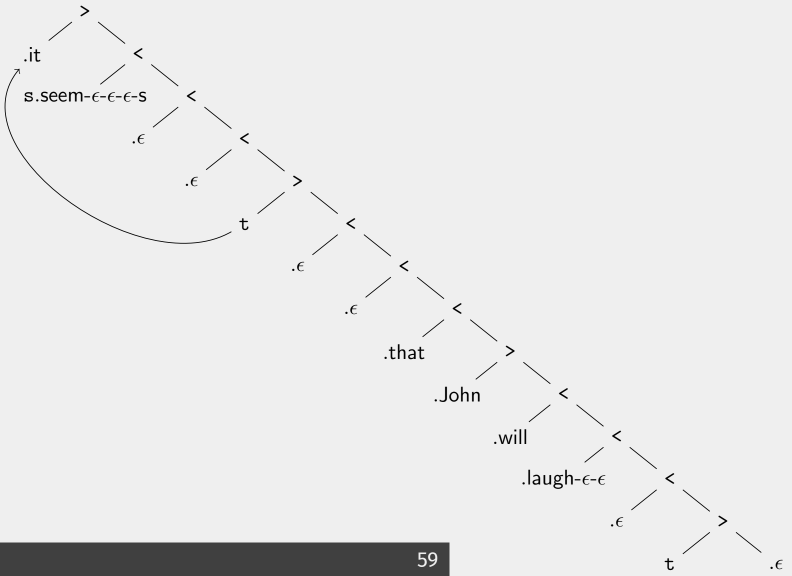
DERIVING *IT*

We assign the *it*-sentence the following structure:



DERIVING *IT*

We assign the *it*-sentence the following structure:



Observations

1. *it* as main clause subject requires finite *that*-complement
2. DP as main clause subject forbids finite *that*-complement

Problem

how to transmit information from one point to another

Solution

Main clause subject is in two places at once. Must satisfy properties of both positions to be well-formed.

ork?

- We still have two lexical entries for *seem*:

  seem' : =c.v   seem : =i.v

- However, there is no point to the distinction between i and c in our grammar. We unify these categories throughout our lexicon:

will : =x.+k.s	have : =perf.x	be : =prog.y
-s : =>x.+k.s	-en : =>y.perf	-ing : =>v.prog
-ed : =>x.+k.s	ϵ : =>y.x	ϵ : =>v.y
that : =s.c	to : =x.c	it : expl.-k
laugh : =d.v	John : d.-k	seem : =c.v

Whither the Weather

Verbs like *rain*, or *snow* can be represented as the below, allowing for *it*-insertion:

$$\text{rain} : v$$

We can then derive the following sentences:

1. It is raining.
2. It seems to be raining.
3. It seems that it is raining.

Raising to Object and Passivization

Raising to Object

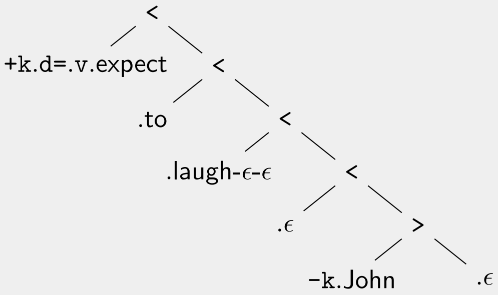
Raising to object, as in:

1. Bill expects John to laugh.
2. Bill expects that John will laugh.

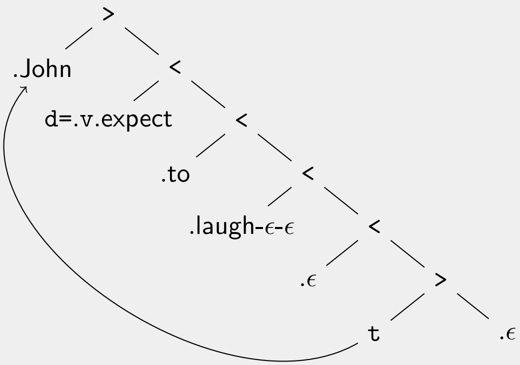
can be accommodated by assigning *expect* the types below:

- expect : =c.+k.d=.v
- expect : =c.d=.v

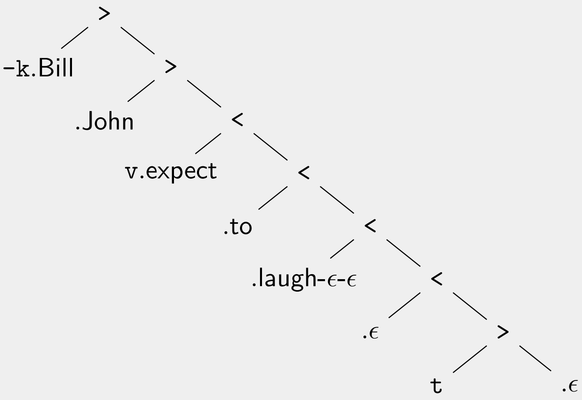
Deriving raised objects



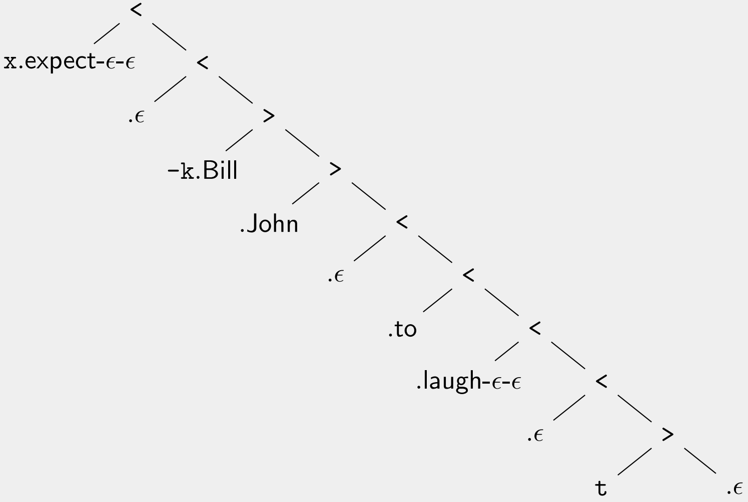
Deriving raised objects



Deriving raised objects



Deriving raised objects



Passive

- Using the idea that DPs have distinct deep and surface positions lets us use our current technology to account for passivization:
  1. Bill expects John to laugh.
  2. John is expected to laugh.
  3. Bill expects that Mary will laugh.
  4. It is expected that Mary will laugh.
- In the first case, the +k of the surface position of the object and the d= of the deep position of the subject are suppressed:

expect : =c.+k.d=.v   expected : =c.pass

be : =pass.v

Passive compression

We again see regularities lurking beneath the surface:

expected : =c.pass $\rightsquigarrow$ expect : =c.V, -en : =>V.pass
expect : =c.+k.d=.v $\rightsquigarrow$ expect : =c.V, ϵ : =>V.+k.d=.v

Remember: Decompositional Methodology

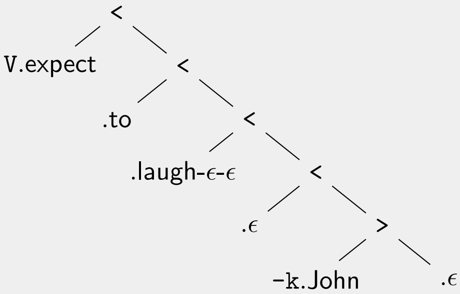
Whenever we have a lexical item

$$\text{uv} : \alpha\beta$$

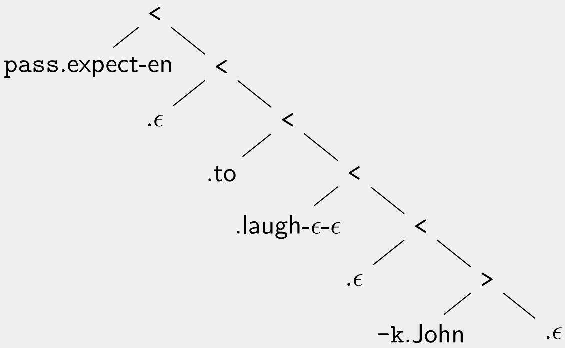
We can split it up into two:

$$\text{u} : \alpha.\text{x} \quad \text{-v} : \text{=>x}.\beta$$

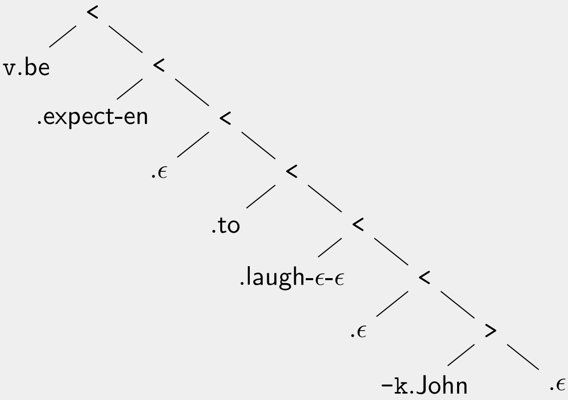
$$\text{u-v} \mapsto \text{uv}$$


<
V.expect
<
.to
<
.laugh-ϵ-ϵ
<
.ϵ
>
−k.John
.ϵ

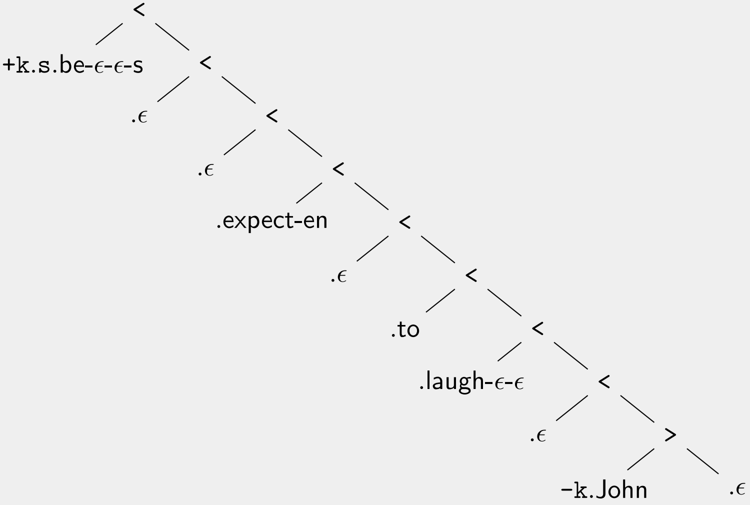
Passive structures

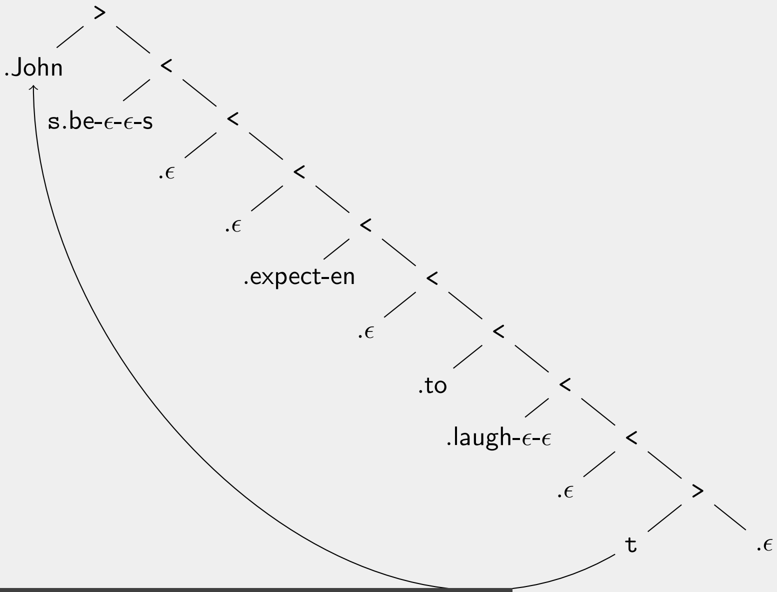


Passive structures



Passive structures



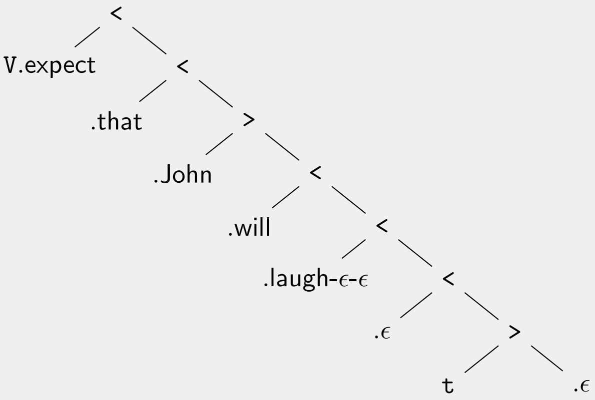

>
.John
<
s.be-ϵ-ϵ-s
<
.ϵ
<
.ϵ
<
.expect-en
<
.ϵ
<
.to
<
.laugh-ϵ-ϵ
<
.ϵ
>
t
.ϵ

Happy conspiricies

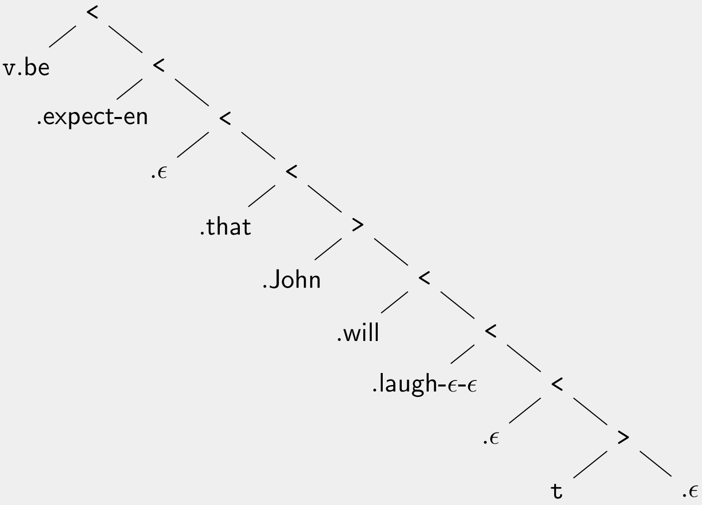
With these lexical entries, we already derive both passive forms:

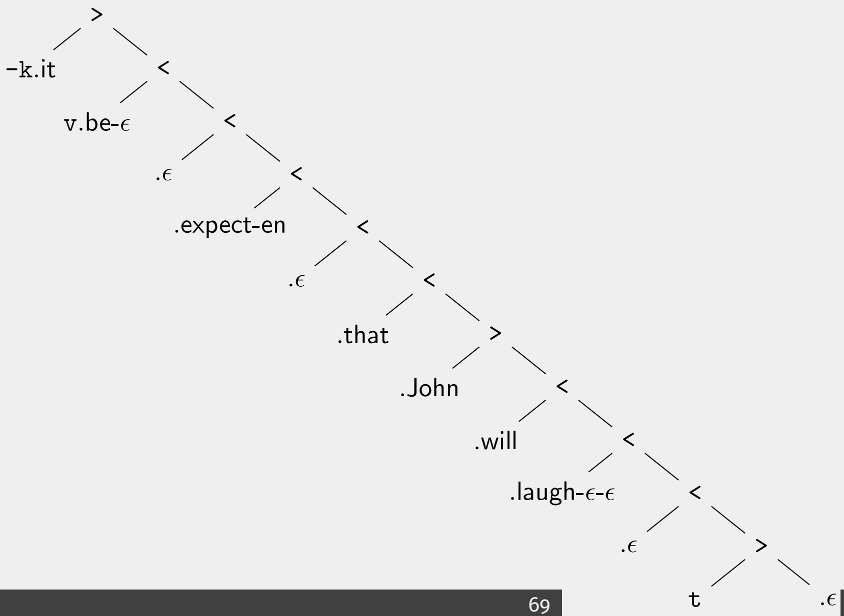
1. John is expected to laugh
2. It is expected that John will laugh

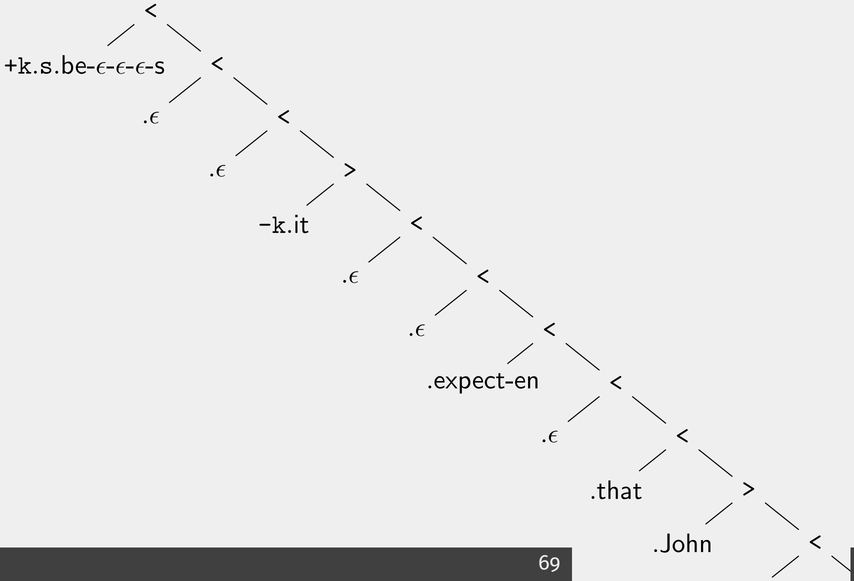
It passive



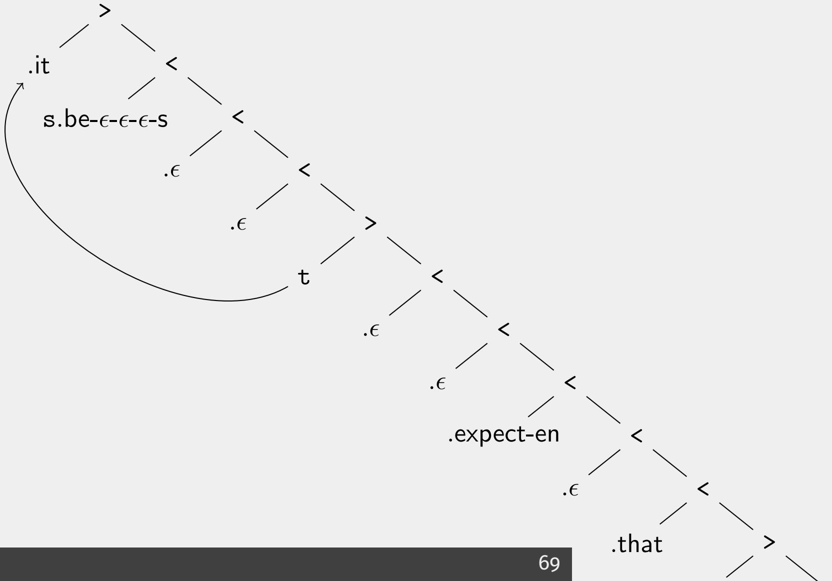
It passive



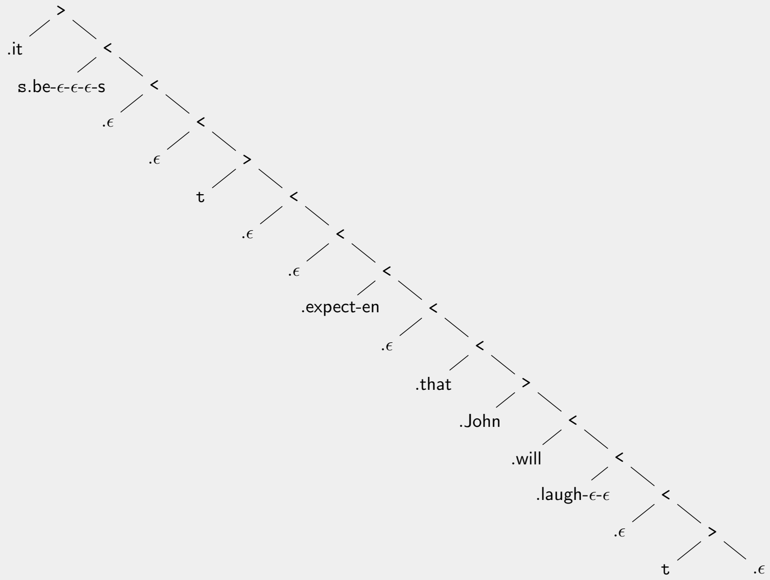

>
−k.it
<
v.be-ϵ
<
.ϵ
<
.expect-en
<
.ϵ
<
.that
>
.John
<
.will
<
.laugh-ϵ-ϵ
<
.ϵ
>
t
.ϵ


<
+k.s.be-ϵ-ϵ-ϵ-s
<
.ϵ
<
.ϵ
>
−k.it
<
.ϵ
<
.ϵ
<
.expect-en
<
.ϵ
<
.that
>
.John
<

It passive



It passive



Expecting compression

- What can we say about the two lexical entries for *expect*?
    1. expect : =c.V
    2. expect : =c.d=.v
- The latter we can decompose into

$$\text{expect} : \text{=c.V} \qquad \epsilon : \text{=>V.d=.v}$$

- The element on the right looks similar to our 'active voice head':
  ϵ : =>V.+k.d=.v
- We decompose once more, disentangling case assignment and external argument selection:

$$\text{expect} : \text{=c.V} \quad \epsilon : \text{=>V.+k.agrO} \quad \epsilon : \text{=>agrO.d=.v}$$
$$\epsilon : \text{=>V.agrO}$$

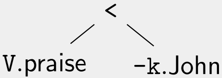
Taking stock

Our lexicon looks as follows:

will : =x.+k.s	have : =perf.x	be : =prog.y
-s : =>x.+k.s	-en : =>y.perf	-ing : =>v.prog
-ed : =>x.+k.s	ϵ : =>y.x	ϵ : =>v.y
that : =s.c	to : =x.c	it : expl.-k
ϵ : =>agrO.d=.v	ϵ : =>V.+k.agrO	ϵ : =>V.agrO
-en : =>V.pass	be : =pass.v	
laugh : d=.v	rain : v	John : d.-k
seem : =c.v	expect : =c.V	

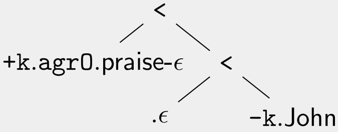
Transitivity

A simple transitive verb looks as follows: praise :: =d.V

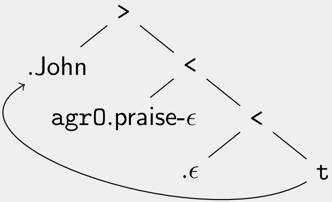


Transitivity

A simple transitive verb looks as follows: praise :: =d.V

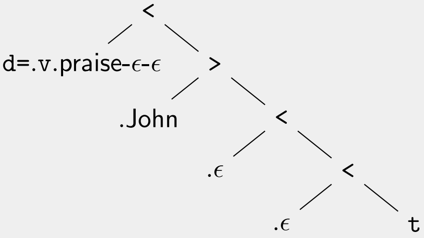


A simple transitive verb looks as follows: praise :: =d.V



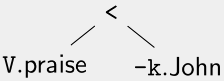
TRANSITIVITY

A simple transitive verb looks as follows: praise : =d.V



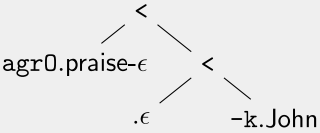
Which Agr0?

The SMC ensures that, in the active voice, agr0 must check the object's case



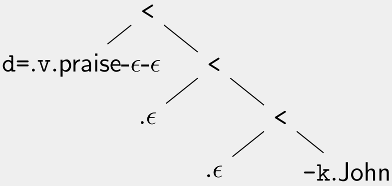
WHICH Agr0?

The SMC ensures that, in the active voice, agr0 must check the object's case



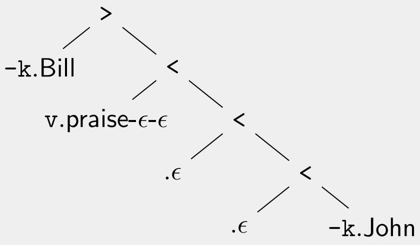
Which Agr0?

The SMC ensures that, in the active voice, agr0 must check the object's case



Which Agr0?

The SMC ensures that, in the active voice, agr0 must check the object's case



Obligatory Raising to Object

- Some raising to object verbs do not allow for *that*-complements

Obligatory Raising to Object

- Some raising to object verbs do not allow for *that*-complements
    1. Bill caused John to laugh.

Obligatory Raising to Object

- Some raising to object verbs do not allow for *that*-complements
    1. Bill caused John to laugh.
    2. $*$Bill caused that John laughed.

Obligatory Raising to Object

- Some raising to object verbs do not allow for *that*-complements
    1. Bill caused John to laugh.
    2. ∗Bill caused that John laughed.
- In order to describe verbs like these, we need to reimplement a distinction between finite and non-finite complements (`c` and `i`)

$$\text{cause} :: \texttt{=i.V} \qquad \text{to} :: \texttt{=x.i}$$

Obligatory Raising to Object

- Some raising to object verbs do not allow for *that*-complements
  1. Bill caused John to laugh.
  2. ∗Bill caused that John laughed.
- In order to describe verbs like these, we need to reimplement a distinction between finite and non-finite complements (c and i)

$$\text{cause} : \texttt{=i.V} \qquad \text{to} : \texttt{=x.i}$$

- However, in order to continue to be able to describe the distribution of *seem* with a single lexical item, we want to say that there is a relation between i and c; namely, that i isa c:

$$\epsilon : \texttt{=>i.c}$$

Obligatorily Passive

- Some verbs *only* appear in the passive:

Obligatorily Passive

- Some verbs *only* appear in the passive:
    1. It is rumored that John laughed.

Obligatorily Passive

- Some verbs *only* appear in the passive:
    1. It is rumored that John laughed.
    2. John is rumored to have laughed.

Obligatorily Passive

- Some verbs *only* appear in the passive:
  1. It is rumored that John laughed.
  2. John is rumored to have laughed.
  3. $*$Bill rumors that John laughed.

Obligatorily Passive

- Some verbs *only* appear in the passive:
    1. It is rumored that John laughed.
    2. John is rumored to have laughed.
    3. ∗Bill rumors that John laughed.
    4. ∗Bill rumors John to have laughed.

Obligatorily Passive

- Some verbs *only* appear in the passive:
  1. It is rumored that John laughed.
  2. John is rumored to have laughed.
  3. ∗Bill rumors that John laughed.
  4. ∗Bill rumors John to have laughed.
- we can assign such verbs the following type:

$$\text{rumored} : \mathtt{=c.pass}$$

Obligatorily Passive

- Some verbs *only* appear in the passive:
    1. It is rumored that John laughed.
    2. John is rumored to have laughed.
    3. *Bill rumors that John laughed.
    4. *Bill rumors John to have laughed.
- we can assign such verbs the following type:

$$\text{rumored} : \mathtt{=c.pass}$$

- alternatively, we can stipulate that the problem lies with the morphological component

$$\text{rumored-AgrO} \mapsto \bot$$

Obligatorily Passive

- Some verbs *only* appear in the passive:
  1. It is rumored that John laughed.
  2. John is rumored to have laughed.
  3. $*$Bill rumors that John laughed.
  4. $*$Bill rumors John to have laughed.
- we can assign such verbs the following type:

$$\text{rumored} : \texttt{=c.pass}$$

- alternatively, we can stipulate that the problem lies with the morphological component

$$\text{rumored-AgrO} \mapsto \perp$$

- or simply view this as a matter of frequency

$$P(\text{-en} : \texttt{=>V.pass} | \text{rumor} : \texttt{=c.V}) \approx 1$$

What is not under the sun

What this analysis doesn't really allow to be stated elegantly:

- a sentential complement taking verb which

What is not under the sun

What this analysis doesn't really allow to be stated elegantly:

- a sentential complement taking verb which
    1. is passivizable

What is not under the sun

What this analysis doesn't really allow to be stated elegantly:

- a sentential complement taking verb which
    1. is passivizable
    2. takes *only* a *that*-complement in the active

What is not under the sun

What this analysis doesn't really allow to be stated elegantly:

- a sentential complement taking verb which
  1. is passivizable
  2. takes *only* a *that*-complement in the active
  3. but takes either a *that*-complement or a *to*-complement in the passive

What is not under the sun

What this analysis doesn't really allow to be stated elegantly:

- a sentential complement taking verb which
  1. is passivizable
  2. takes *only* a *that*-complement in the active
  3. but takes either a *that*-complement or a *to*-complement in the passive
- Like *think*?

What is not under the sun

What this analysis doesn't really allow to be stated elegantly:

- a sentential complement taking verb which
    1. is passivizable
    2. takes *only* a *that*-complement in the active
    3. but takes either a *that*-complement or a *to*-complement in the passive
- Like *think*?
    1. Bill thinks that John is laughing

What is not under the sun

What this analysis doesn't really allow to be stated elegantly:

- a sentential complement taking verb which
  1. is passivizable
  2. takes *only* a *that*-complement in the active
  3. but takes either a *that*-complement or a *to*-complement in the passive
- Like *think*?
  1. Bill thinks that John is laughing
  2. ∗Bill thinks John to be laughing.

What is not under the sun

What this analysis doesn't really allow to be stated elegantly:

- a sentential complement taking verb which
    1. is passivizable
    2. takes *only* a *that*-complement in the active
    3. but takes either a *that*-complement or a *to*-complement in the passive
- Like *think*?
    1. Bill thinks that John is laughing
    2. ∗Bill thinks John to be laughing.
    3. It is thought that John is laughing.

What is not under the sun

What this analysis doesn't really allow to be stated elegantly:

- a sentential complement taking verb which
  1. is passivizable
  2. takes *only* a *that*-complement in the active
  3. but takes either a *that*-complement or a *to*-complement in the passive
- Like *think*?
  1. Bill thinks that John is laughing
  2. ∗Bill thinks John to be laughing.
  3. It is thought that John is laughing.
  4. John is thought to be laughing.

What is not under the sun

What this analysis doesn't really allow to be stated elegantly:

- a sentential complement taking verb which
  1. is passivizable
  2. takes *only* a *that*-complement in the active
  3. but takes either a *that*-complement or a *to*-complement in the passive
- Like *think*?
  1. Bill thinks that John is laughing
  2. ∗Bill thinks John to be laughing.
  3. It is thought that John is laughing.
  4. John is thought to be laughing.
- But:

What is not under the sun

What this analysis doesn't really allow to be stated elegantly:

- a sentential complement taking verb which
  1. is passivizable
  2. takes *only* a *that*-complement in the active
  3. but takes either a *that*-complement or a *to*-complement in the passive
- Like *think*?
  1. Bill thinks that John is laughing
  2. ∗Bill thinks John to be laughing.
  3. It is thought that John is laughing.
  4. John is thought to be laughing.
- But:
  - `think me to be`, 1mil Google hits

What is not under the sun

What this analysis doesn't really allow to be stated elegantly:

- a sentential complement taking verb which
  1. is passivizable
  2. takes *only* a *that*-complement in the active
  3. but takes either a *that*-complement or a *to*-complement in the passive
- Like *think*?
  1. Bill thinks that John is laughing
  2. ∗Bill thinks John to be laughing.
  3. It is thought that John is laughing.
  4. John is thought to be laughing.
- But:
  - `think me to be`, 1mil Google hits
  - *it would hurt even my delicacy, little as you may think me to possess*

Full disclosure

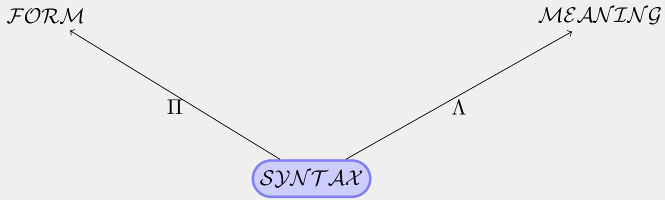
Here is an alternation that I'm not sure how to deal with:

1. I made him laugh
2. He was made to laugh
3. $*$ I made that he laughed
4. active :: make : =v.V
5. passive :: make : =i.pass

Derivations

Syntax

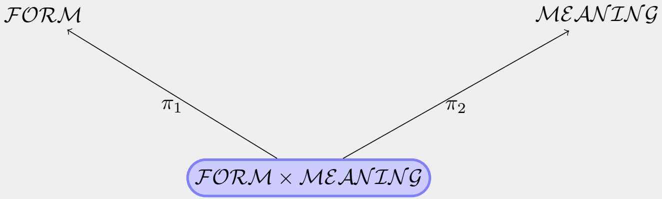
Glues together form and meaning



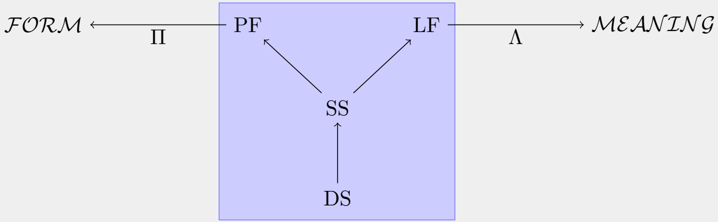
This is not syntactocentric

The point is to specify

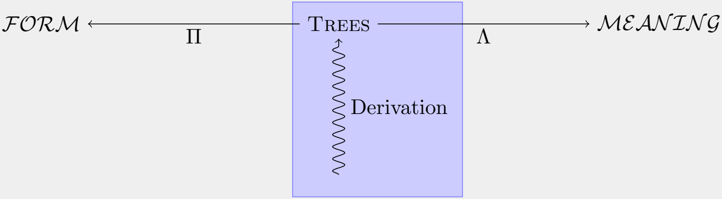
form-meaning pairs



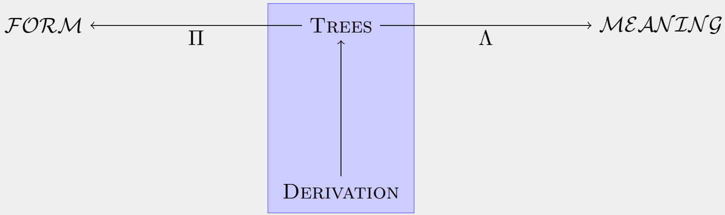
T-model



Minimalism (I)

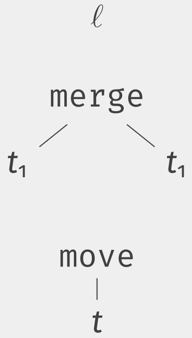


Minimalism (reifying derivations)

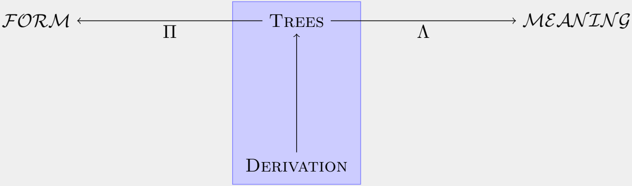


The set of *possible derivations* over a lexicon *Lex* is the set of terms over { `merge`, `move` } $\cup$ *Lex*

1. if ℓ is a lexical item, then ℓ is a derivation (of itself)
2. if t_1, t_2 are possible derivations, then so is their merger
3. if t is a possible derivation, its `move` is too



From derivations to derivata



how do we go from Derivation to Tree?

by *doing* what the derivation describes

Well-formedness

Not every derivation is well-formed

move
|
move
|
John : d.−k

How to determine whether a derivation is well-formed?

aka is there structure in well-formedness

Real-life linguistics

Borer's exoskeletalism

- syntax applies willy-nilly
- interface maps filter bad stuff out

Cool idea, but…

what's really at issue?

Relevant question

How hard is it to delimit bad derivations from good?

Checking derivations

we will see what information we need to determine well-formedness of a possible derivation tree

Three cases

1. lexical item
2. `merge`
3. `move`

we imagine checking by

walking up the tree

Checking lexical items

if we have a derivation tree of the form ℓ (i.e. a leaf)

we need to know what ℓ is

so we can check if it is in the lexicon

this requires just a finite amount of built-in information, as the lexicon is finite
(just a look-up table)

Checking merge

Given that t_1 and t_2 are well-formed, is $d = \texttt{merge}(t_1, t_2)$?

1. we need to know the first feature of each head so that we can check
   - whether they are the right kind (`x=`/`=x` and `y`)
   - whether they match
2. we need to continue to remember the next feature of the head of t_1
   - in case d is the argument to a later `merge`

In general,

we need to remember the features of the head

Checking move

Given that t is well-formed, is $d = \texttt{move}(t)$?

1. the first feature of the head
2. the first features of the moving expressions so that we know
   - whether there is someone that can move
   - whether there are too many (SMC)
3. we need to continue to remember
   - the next features of the head of t
   - the next features of the head of whoever moved

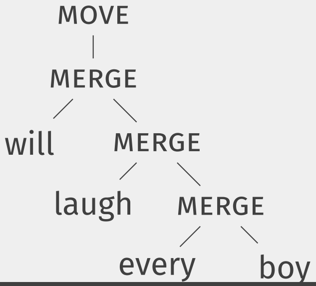
In general,

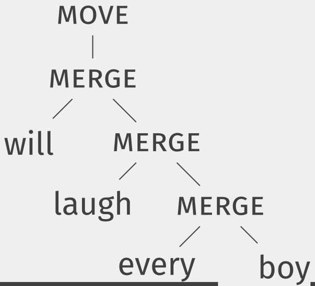
- we need to remember the feature bundle of the head
- and the feature bundles of all moving expressions

Example

given a simple lexicon

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

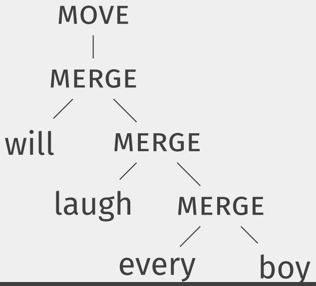


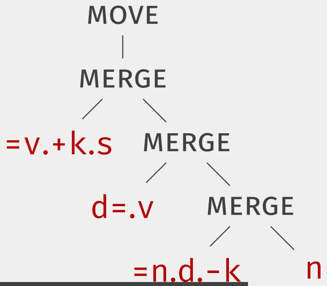


Example

given a simple lexicon

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

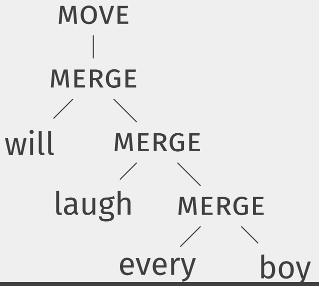


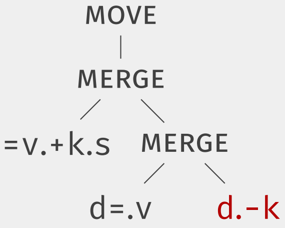


Example

given a simple lexicon

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

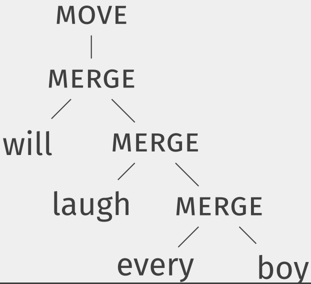


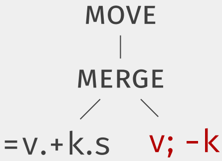


Example

given a simple lexicon

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

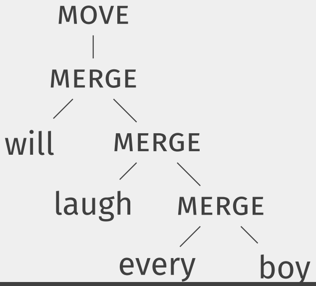


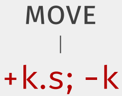


EXAMPLE

given a simple lexicon

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

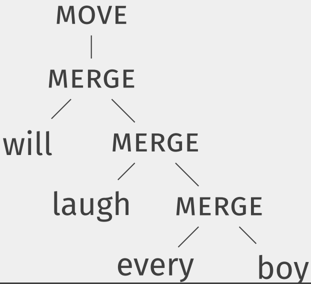




Example

given a simple lexicon

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



s

Regularity

It is very easy to check well-formedness

- finite state tree automaton
- MSO formula
- regular tree language

What does this depend on?

finite upper bound on number of unchecked features in any expression

- individual feature bundles only decrease (never grow larger)
- limit to how many movers can appear in a single tree (SMC)

Back to Borer

Theorem

If you have a *regular* tree set D, and a *partial* regular interface map f

- $D \cap \mathrm{dom}(f)$ is regular
- $f \upharpoonright D$ is regular

Borer's idea: shift work around

- we *know* we can do this
- no empirical content
- theoretical content:
  what is the optimal arrangement of work?

Feature percolation vs Movement

Feature percolation

seem' : $=\mathtt{c}.\mathtt{v}^{\mathtt{c}}$	will' : $=\mathtt{x}^{\mathtt{c}}.\mathtt{d}^{\mathtt{expl}}=.\mathtt{s}$	it : $\mathtt{d}^{\mathtt{expl}}$
seem : $=\mathtt{i}.\mathtt{v}$	will : $=\mathtt{x}.\mathtt{d}=.\mathtt{s}$	John : d

In a derivation of

- *John seemed to laugh*
  the featural content of the complement of *seem* is c; -k
- *It seemed that John laughed*
  the featural content of the complement of *seem* is c

Movement is derivational feature percolation

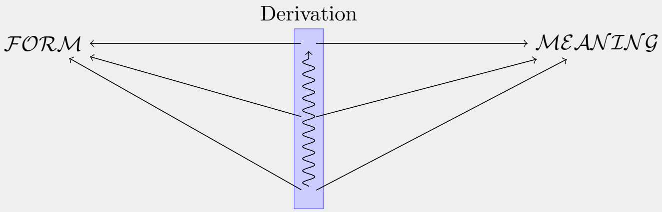
Moral

understanding proposals in terms of derivational structure is informative

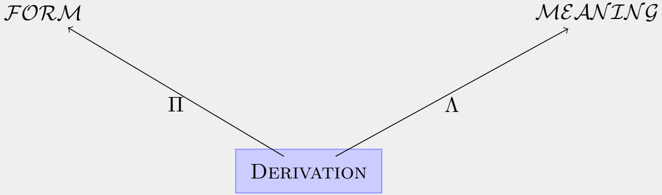
Remember

- We needn’t reify derivations
- We are simply studying the structure implicit in the derivational process

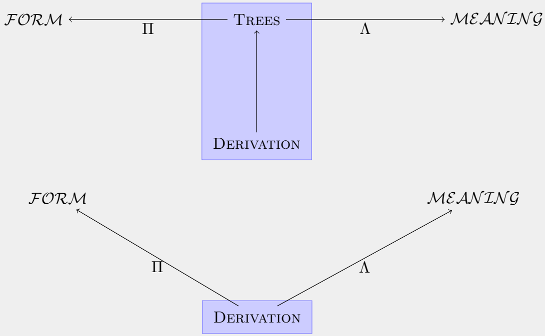
Minimalism (Multiple Spell-out)



Minimalism



Who's right?



FORM
Π
Trees
Λ
MEANING
Derivation
FORM
MEANING
Π
Λ
Derivation

Normally, this is a hard question

here it is easy because

derivations *are isomorphic to* derived structures

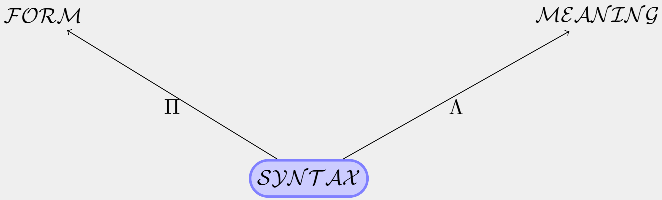
this is normally not the case

(because there's no point to transform something into itself)

Promises, promises

Syntax

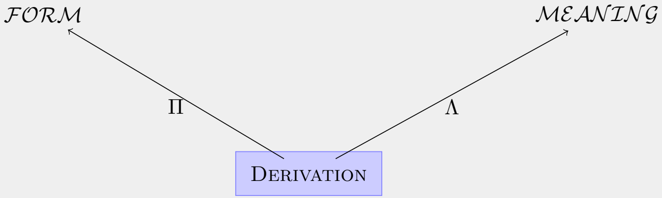
Glues together form and meaning



Promises, promises

Syntax

Glues together form and meaning

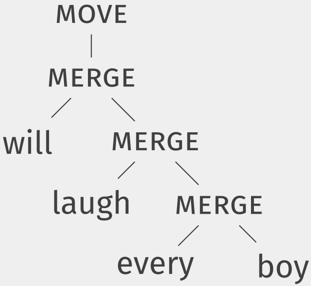


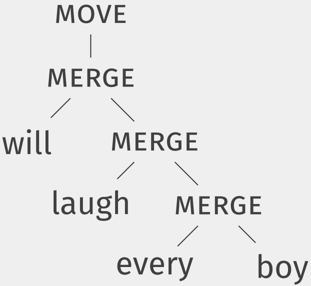
what are Π and Λ?

interface objects are sequences of strings

(Head; mvr; ...; mvr) (Michaelis,98)

@@ @@ only need to keep track of which strings have reached their final position, not of their internal structure

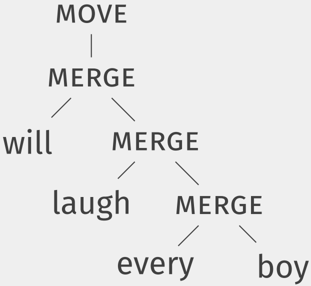


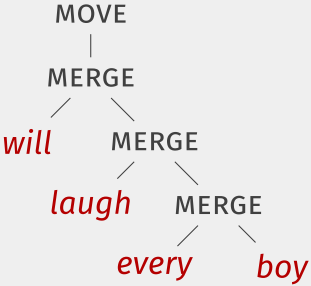


interface objects are sequences of strings

(Head; mvr; . . . ; mvr) (Michaelis,98)

@@ @@

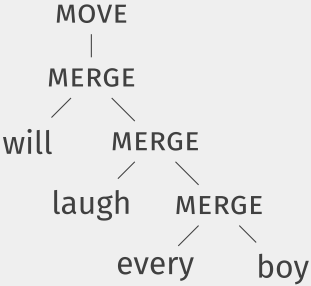


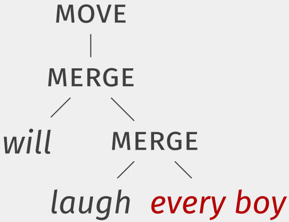


interface objects are sequences of strings

(Head; mvr; …; mvr) (Michaelis,98)

@@ @@

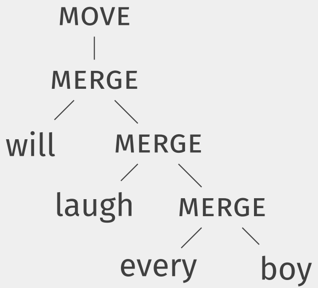


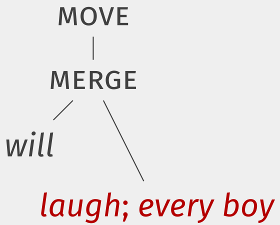


interface objects are sequences of strings

(Head; mvr; ...; mvr) (Michaelis,98)

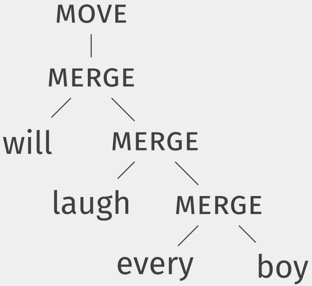
@@ @@





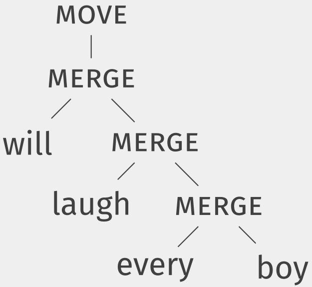
interface objects are sequences of strings

(Head; mvr; …; mvr) (Michaelis,98)
@@ @@



interface objects are sequences of strings

(Head; mvr; ...; mvr) (Michaelis,98)
@@ @@



every boy will laugh

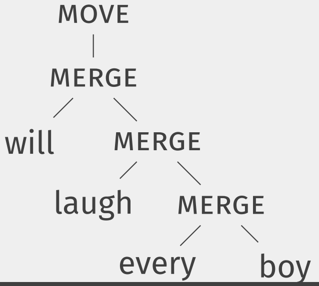
interface objects are sequences of strings

(Head; `mvr`; ...; `mvr`) (Michaelis,98)

@@ @@

Survive minimalism (Stroik, 99)

Nontransformational derivations (Brosziewski, 00)

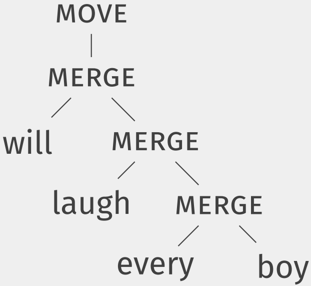


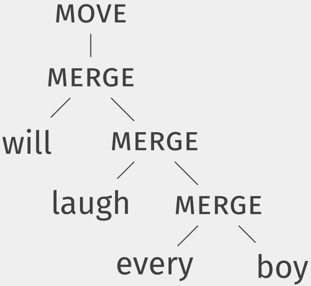
every boy will laugh

interface objects are sequences of λ -terms

(Head; mvr; ...; mvr) (Kobele,12)

but written: $\texttt{mvr}, \ldots, \texttt{mvr} \vdash \texttt{Head}$

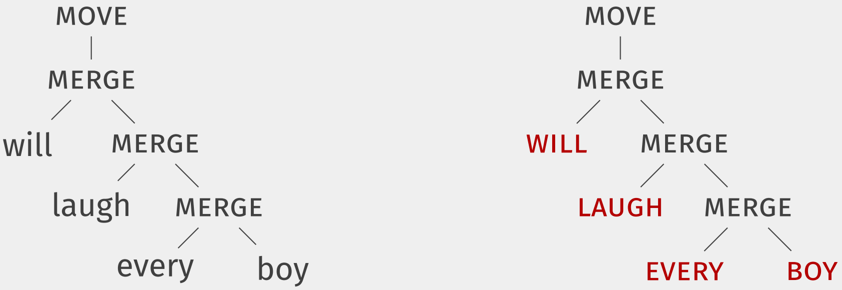




interface objects are sequences of λ -terms

($\mathtt{Head}$; $\mathtt{mvr}$; …; $\mathtt{mvr}$) (Kobele,12)

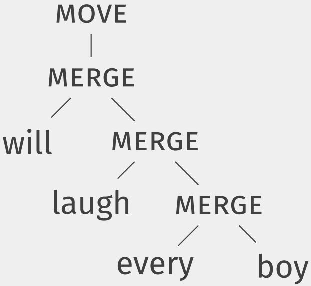
but written: $\mathtt{mvr}, \ldots, \mathtt{mvr} \vdash \mathtt{Head}$

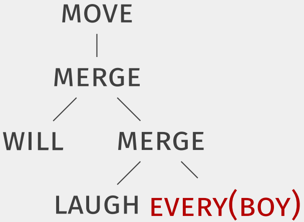


interface objects are sequences of λ -terms

(Head; mvr; ...; mvr) (Kobele,12)

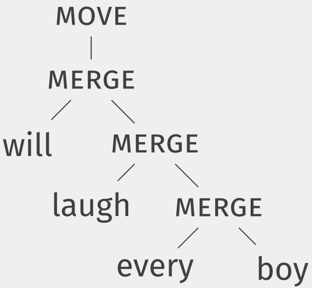
but written: $\texttt{mvr}, \ldots, \texttt{mvr} \vdash \texttt{Head}$

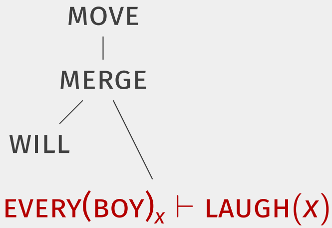




interface objects are sequences of λ -terms

(Head; mvr; …; mvr) (Kobele,12)
but written: $\texttt{mvr}, \ldots, \texttt{mvr} \vdash \texttt{Head}$

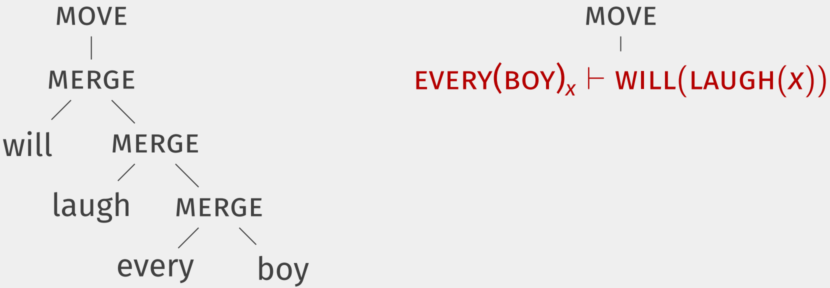




interface objects are sequences of λ -terms

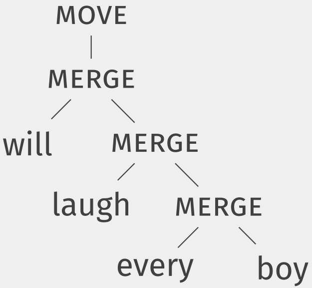
(Head; mvr; …; mvr) (Kobele,12)

but written: $\texttt{mvr}, \ldots, \texttt{mvr} \vdash \texttt{Head}$



interface objects are sequences of λ -terms

(Head; mvr; ...; mvr) (Kobele,12)
but written: $\mathtt{mvr}, \ldots, \mathtt{mvr} \vdash \mathtt{Head}$



$\text{EVERY}(\text{BOY})(\lambda x.\text{WILL}(\text{LAUGH}(x)))$