An Introduction to
Mildly Context-Sensitive Grammar Formalisms

— *Minimalist Grammars* —

Gerhard Jäger & Jens Michaelis

Universität Potsdam

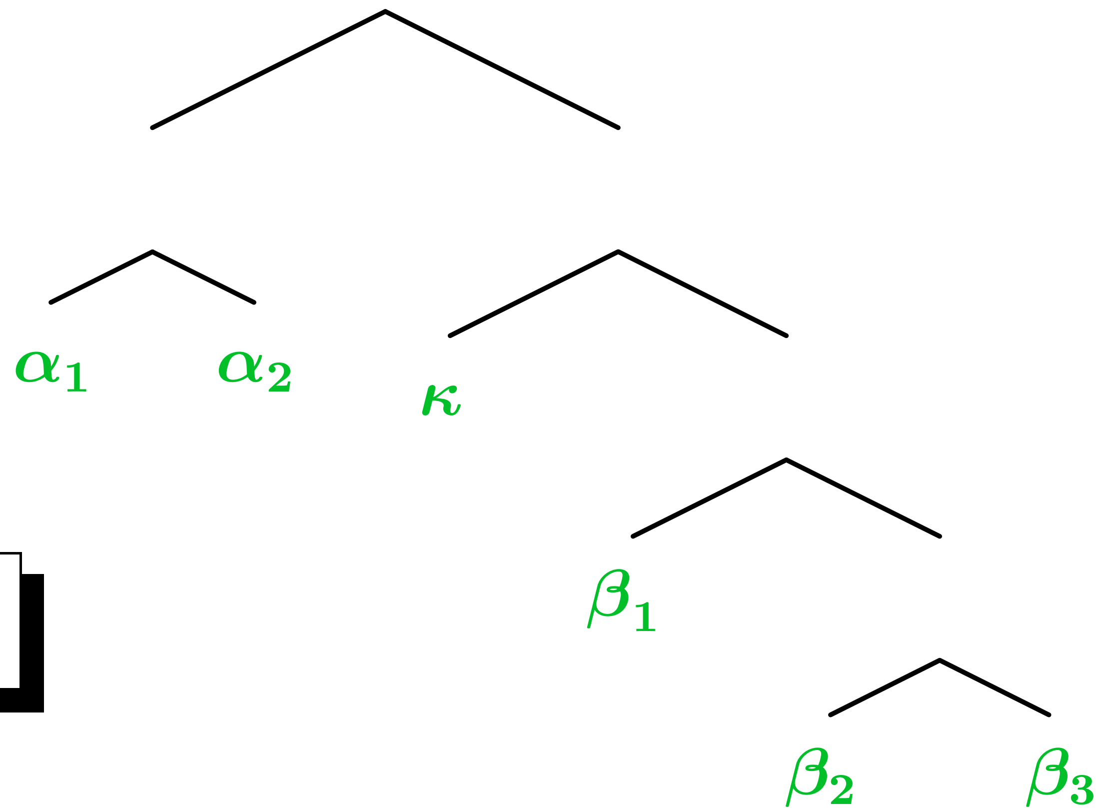
{jaeger,michael}@ling.uni-potsdam.de

Expressions over a feature set

$\text{Feat} = \text{Syn} \cup \text{NonSyn}$

Expressions over a feature set

Feat $=$ Syn $\cup$ NonSyn

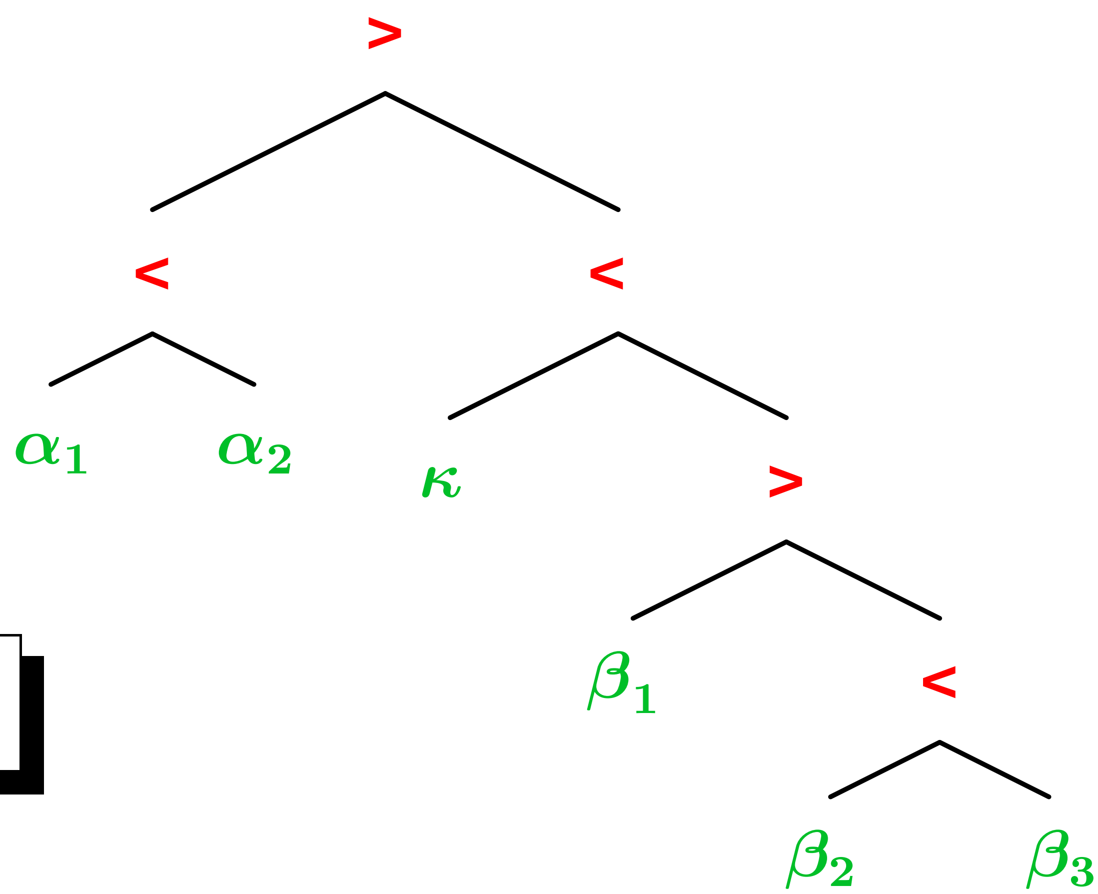


$$\tau = \langle \mathsf{N}_\tau , \lhd^*_\tau , \prec_\tau , \mathsf{label}_\tau \rangle$$

- $\langle \mathsf{N}_\tau , \lhd^*_\tau , \prec_\tau \rangle$ is a finite, binary tree
- $\mathsf{label}_\tau : \mathsf{Leaves}_\tau \longrightarrow$ Syn*NonSyn*

Expressions over a feature set

$\mathrm{Feat} = \mathrm{Syn} \cup \mathrm{NonSyn}$



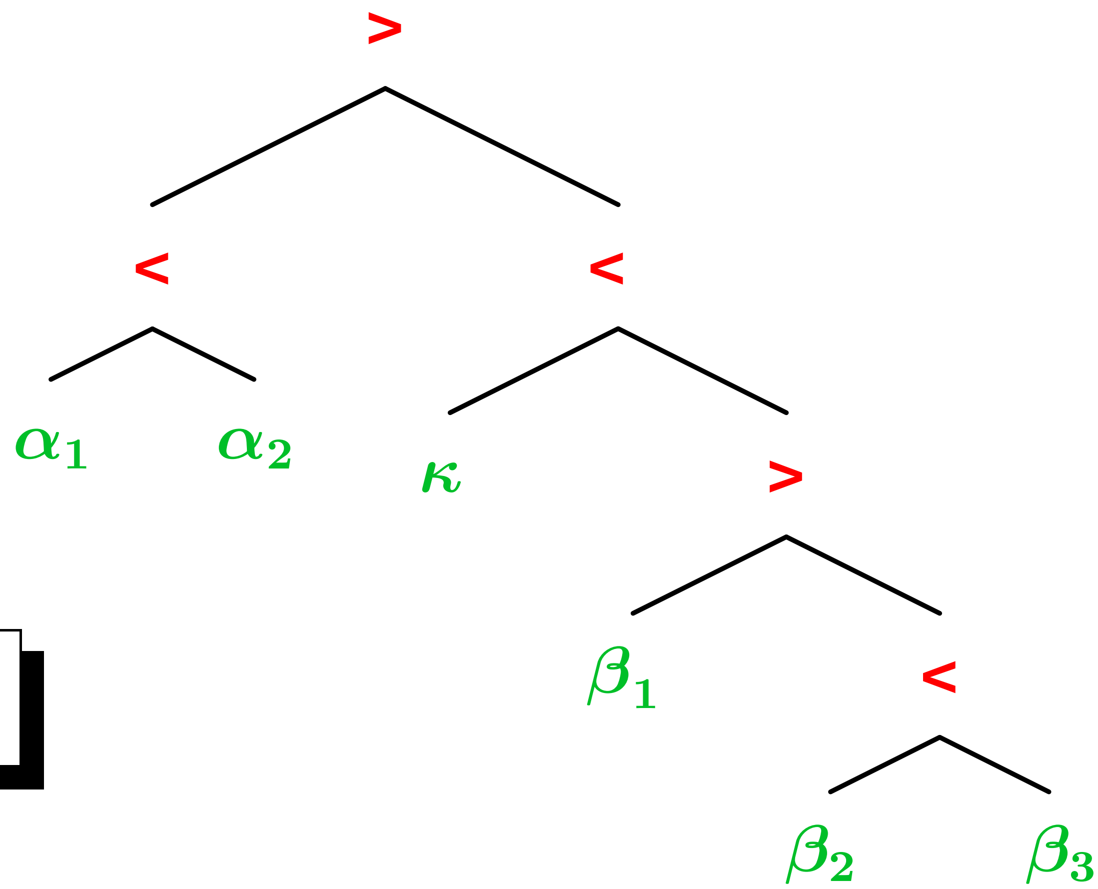
$$\tau = \langle \mathrm{N}_\tau , \triangleleft^*_\tau , \prec_\tau , \mathrm{label}_\tau \rangle$$

- $\langle \mathrm{N}_\tau , \triangleleft^*_\tau , \prec_\tau \rangle$ is a finite, binary tree
- $\mathrm{label}_\tau : \mathrm{Leaves}_\tau \longrightarrow \mathrm{Syn}^*\mathrm{NonSyn}^*$

  $\mathrm{NonLeaves}_\tau \longrightarrow \{ <, > \}$

Expressions over a feature set

Feat $=$ Syn $\cup$ NonSyn



$$\boldsymbol{\tau} = \langle \mathsf{N}_\tau , \triangleleft_\tau^* , \prec_\tau , \mathsf{label}_\tau \rangle$$

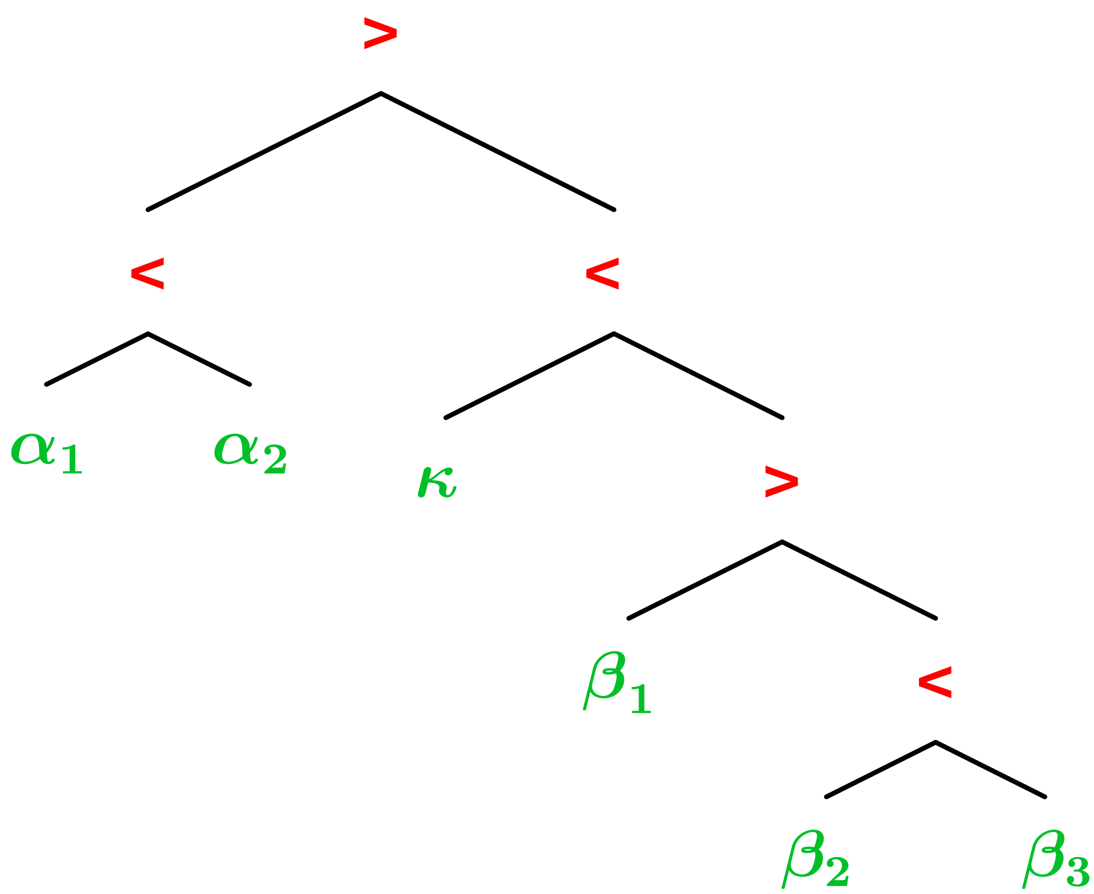
- $\langle \mathsf{N}_\tau , \triangleleft_\tau^* , \prec_\tau \rangle$ is a finite, binary tree

- $\mathsf{label}_\tau : \mathsf{Leaves}_\tau \longrightarrow$ Syn*NonSyn*

  $\mathsf{NonLeaves}_\tau \longrightarrow \{ <, > \}$ [“relation” of projection]

Expressions over a feature set

- **<** “left daughter projects”
- **>** “right daughter projects”

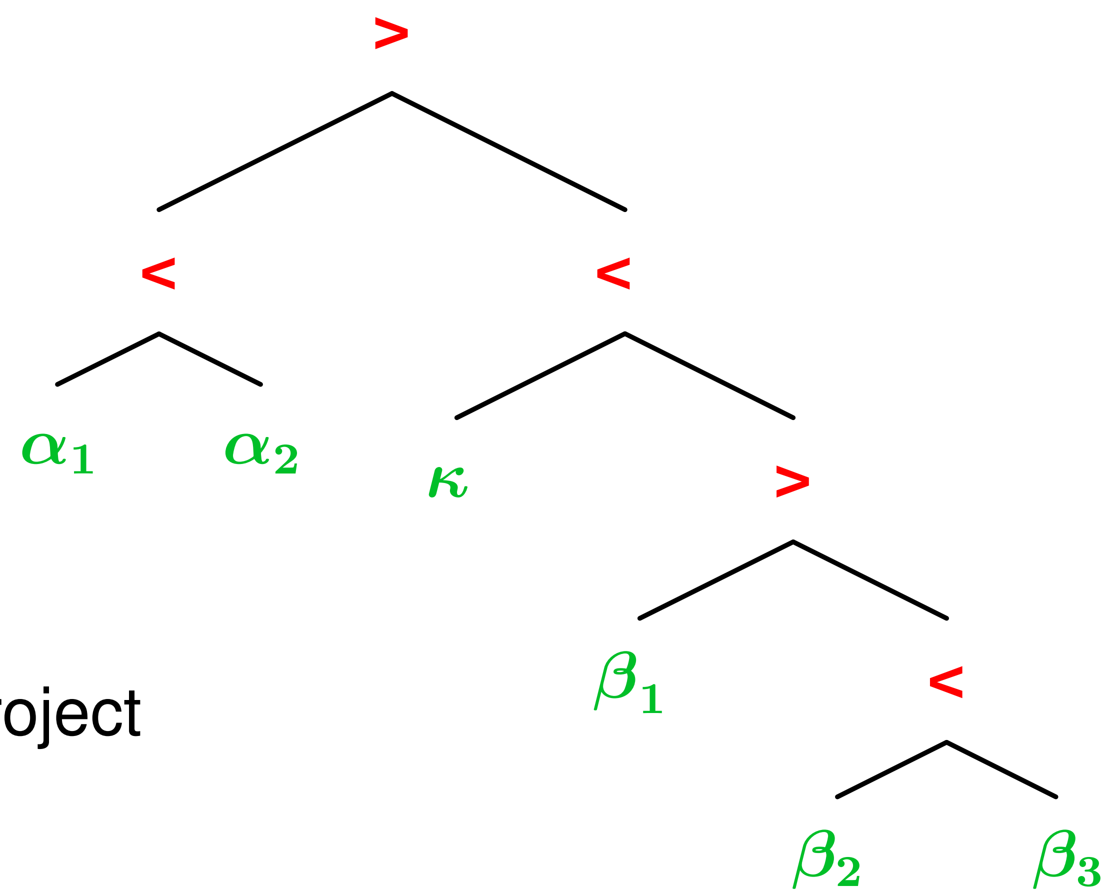


[“relation” of projection]

Expressions over a feature set

< “left daughter projects”

> “right daughter projects”



maximal projections :

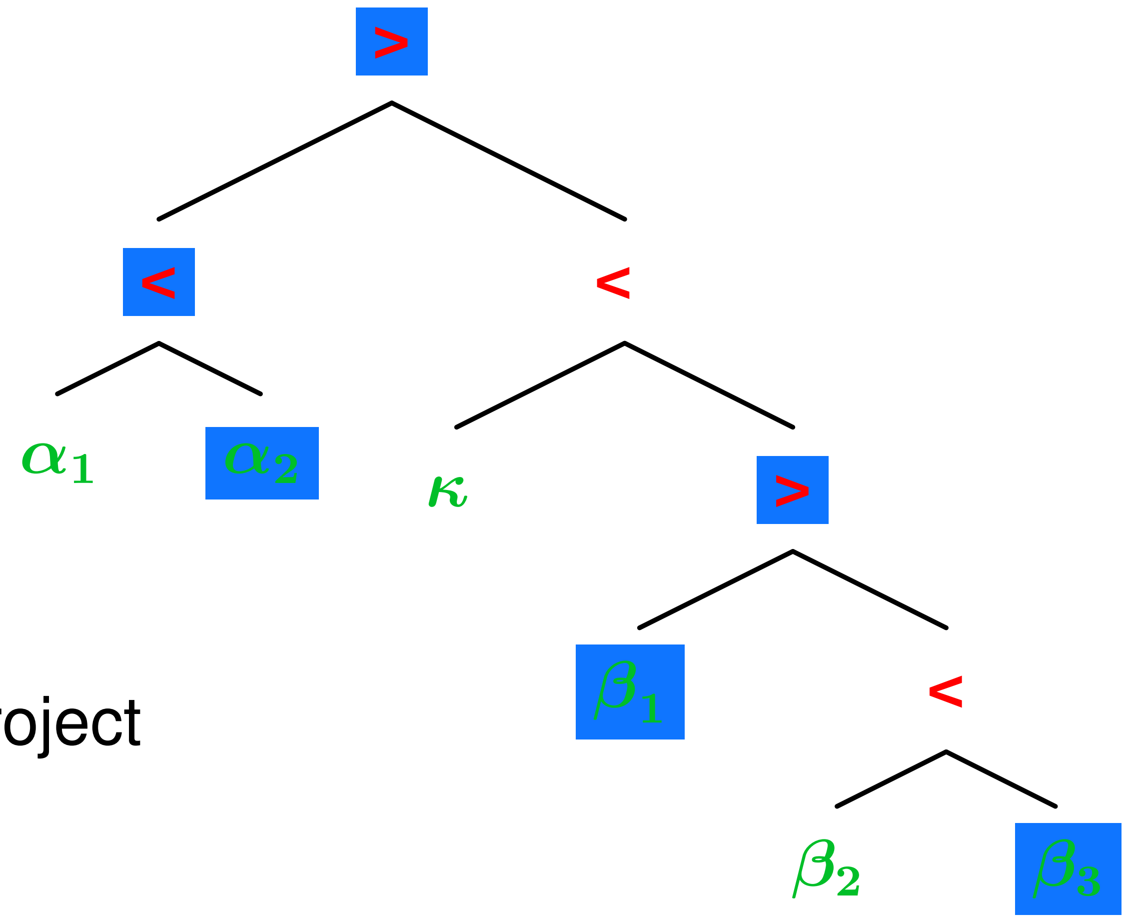
subtrees whose roots don’t project

[“relation” of projection]

Expressions over a feature set

< “left daughter projects”

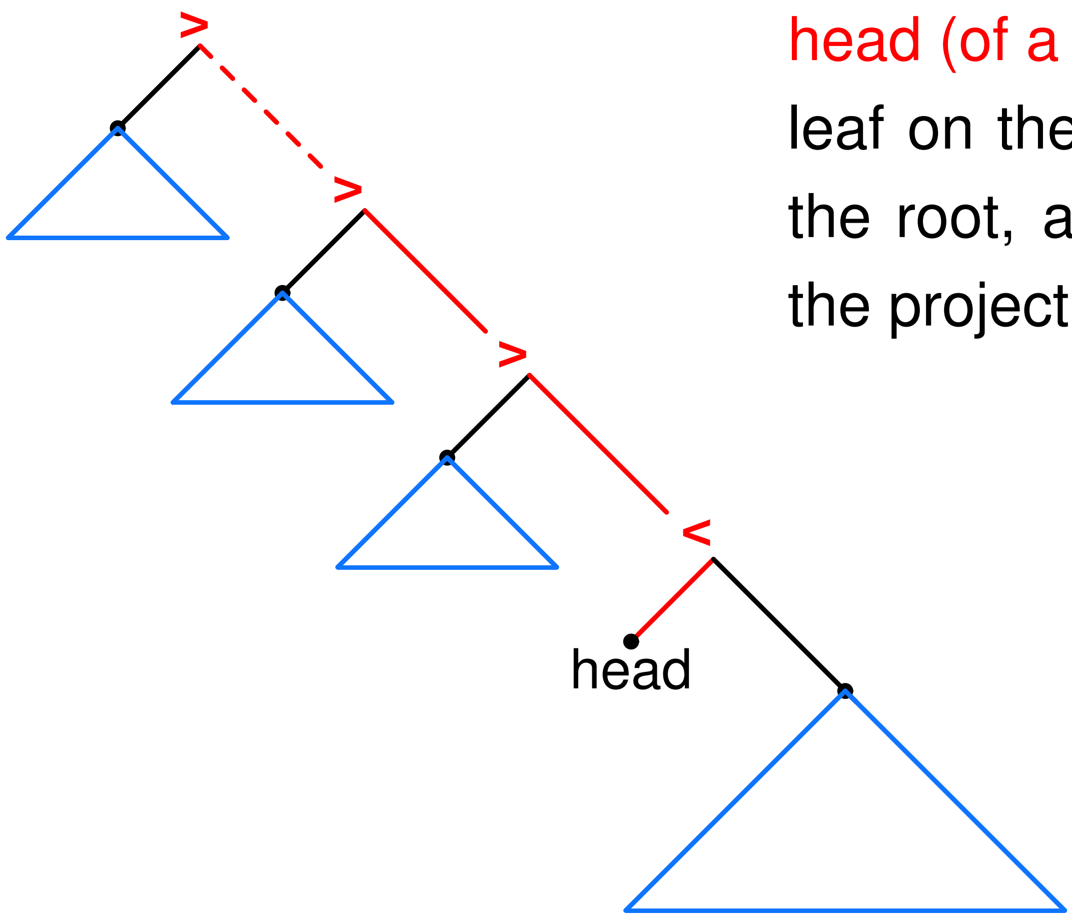
> “right daughter projects”



maximal projections :

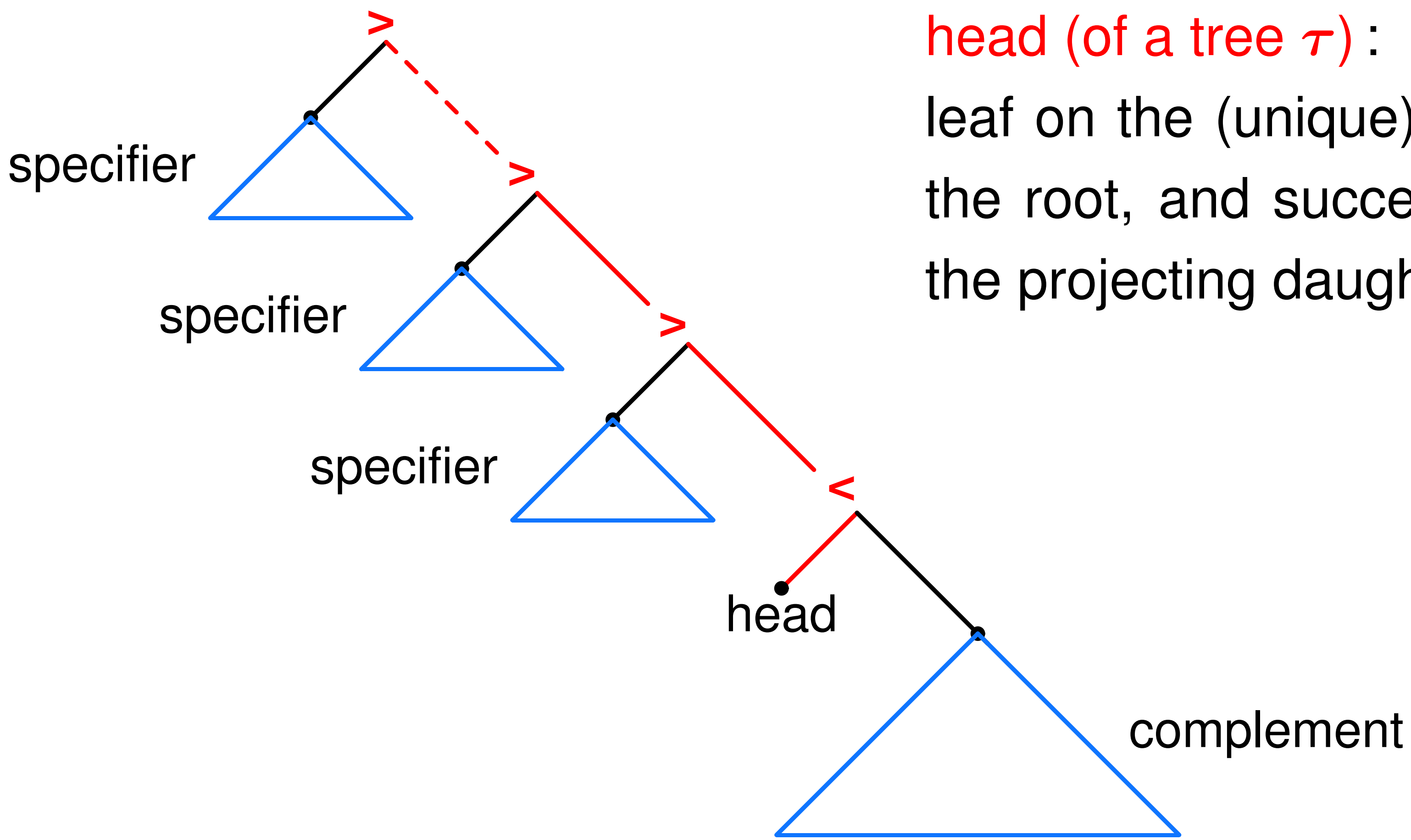
subtrees whose roots don’t project

[“relation” of projection]



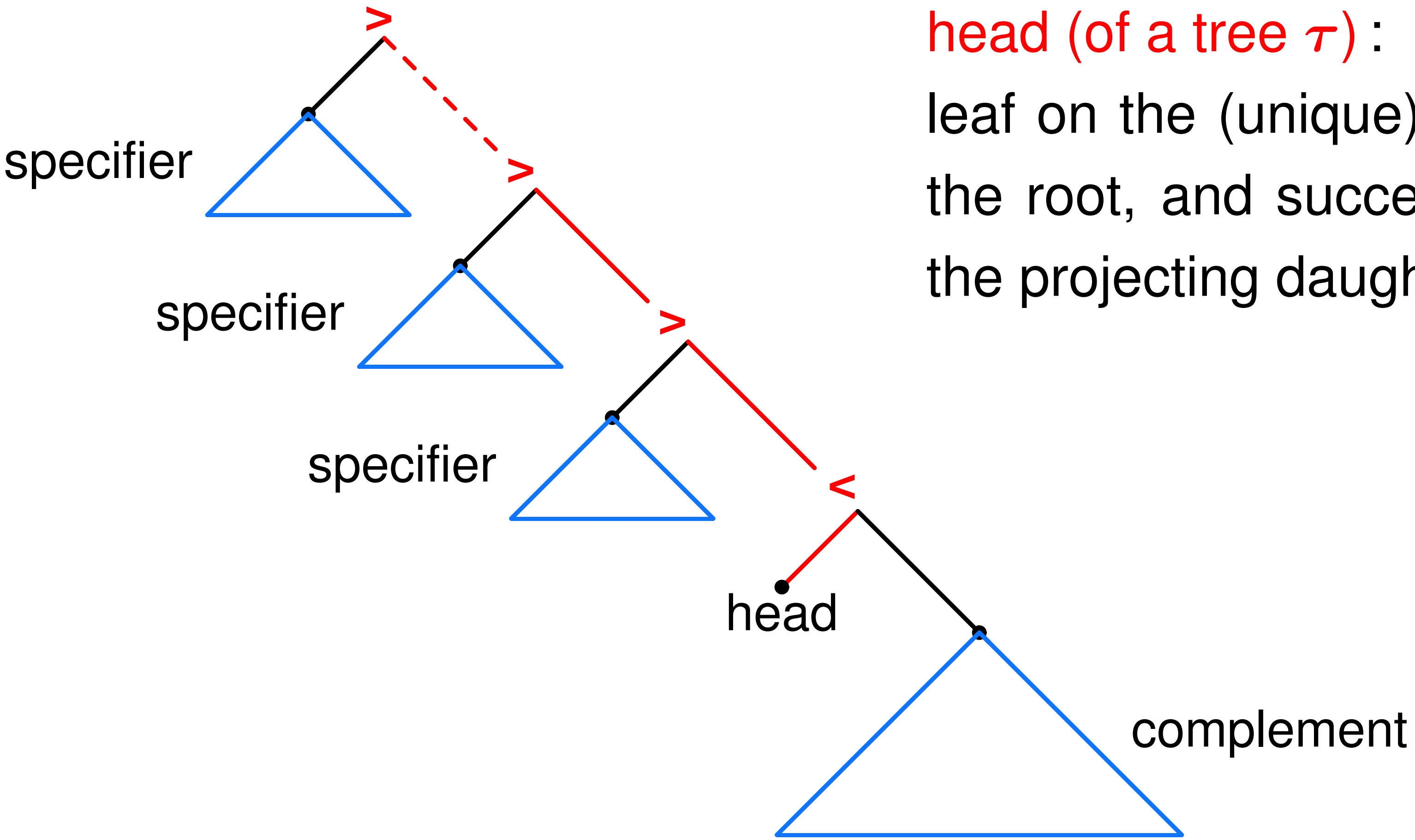
head (of a tree τ) :

leaf on the (unique) path starting at the root, and successively following the projecting daughters



head (of a tree τ) :

leaf on the (unique) path starting at the root, and successively following the projecting daughters



head (of a tree τ) :
leaf on the (unique) path starting at the root, and successively following the projecting daughters

τ displays feature f $:\Longleftrightarrow$ τ's head-label is of the form $f\alpha$

Building expressions

- Expressions can be built up from other expressions by applying structure building functions.

- The applications of these functions are triggered by particular instances of syntactic features appearing in the leaf-labels of the trees to which the functions are applied.

- After having been applied the triggering instances are deleted and count as checked.

- Different structure building operations are triggered by different types of syntactic features.

Syntactic features (the set Syn)

- Syn is partitioned into . . .

Base	$= \{\mathtt{x}, \mathtt{y}, \mathtt{z}, \ldots\}$	*(basic) categories*
Select	$= \{\mathtt{=x}, \mathtt{=y}, \mathtt{=z}, \ldots\}$	*(merge-)selectors*
Licensees	$= \{\mathtt{-x}, \mathtt{-y}, \mathtt{-z}, \ldots\}$	*(move)-licensees*
Licensors	$= \{\mathtt{+x}, \mathtt{+y}, \mathtt{+z}, \ldots\}$	*(move)-licensors*

Syntactic features (the set Syn)

- Syn is partitioned into . . .

Base	$= \{ \mathtt{x}, \mathtt{y}, \mathtt{z}, \ldots \}$	*(basic) categories*
Select	$= \{ \mathtt{=x}, \mathtt{=y}, \mathtt{=z}, \ldots \}$	*(merge-)selectors*
Licensees	$= \{ \mathtt{-x}, \mathtt{-y}, \mathtt{-z}, \ldots \}$	*(move)-licensees*
Licensors	$= \{ \mathtt{+x}, \mathtt{+y}, \mathtt{+z}, \ldots \}$	*(move)-licensors*

- NonSyn* $= \{$ *book* , *which* , *Mary_ read* , . . . , $\emptyset$, . . . $\}$

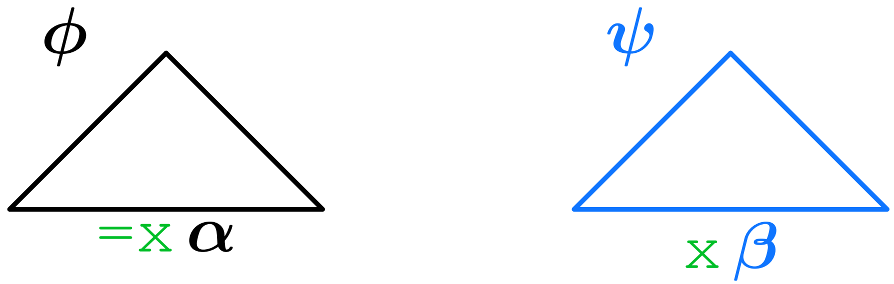
Structure building functions

$$\text{merge} : \text{Exp(Feat)} \times \text{Exp(Feat)} \xrightarrow{\text{part}} \text{Exp(Feat)}$$

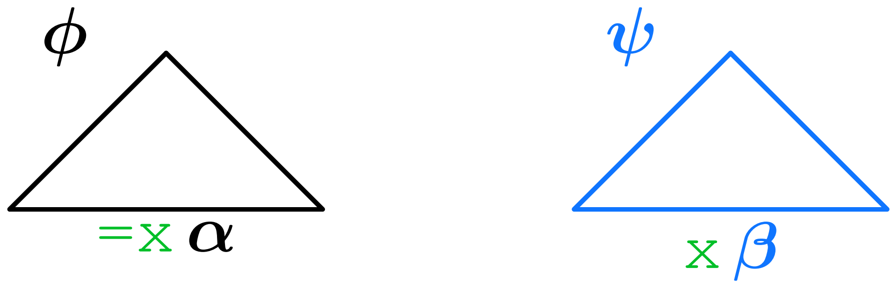
$\langle \phi , \psi \rangle \in$ Domain(merge) $:\Longleftrightarrow$

- ψ displays feature $\mathtt{x} \in$ Base
- ϕ displays feature $\mathtt{=x} \in$ Select

$$\text{merge} : \text{Exp(Feat)} \times \text{Exp(Feat)} \xrightarrow{\text{part}} \text{Exp(Feat)}$$



$$\text{merge} : \text{Exp(Feat)} \times \text{Exp(Feat)} \xrightarrow{\text{part}} \text{Exp(Feat)}$$

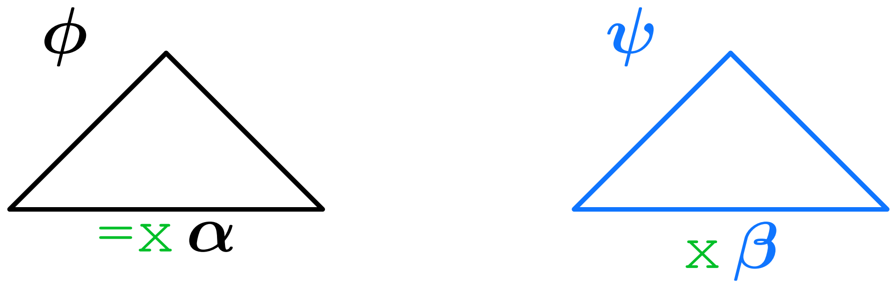


$\rightsquigarrow$

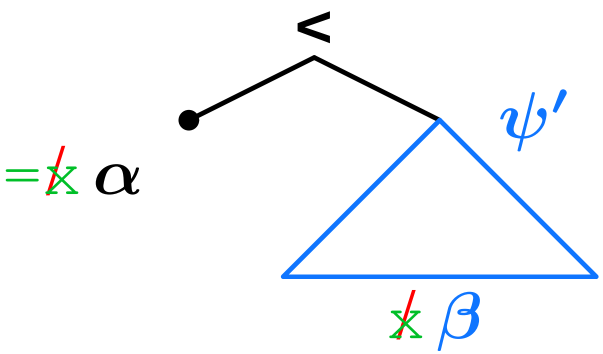
ϕ simple

ϕ complex

$$\text{merge} : \text{Exp(Feat)} \times \text{Exp(Feat)} \xrightarrow{\text{part}} \text{Exp(Feat)}$$



$\rightsquigarrow$

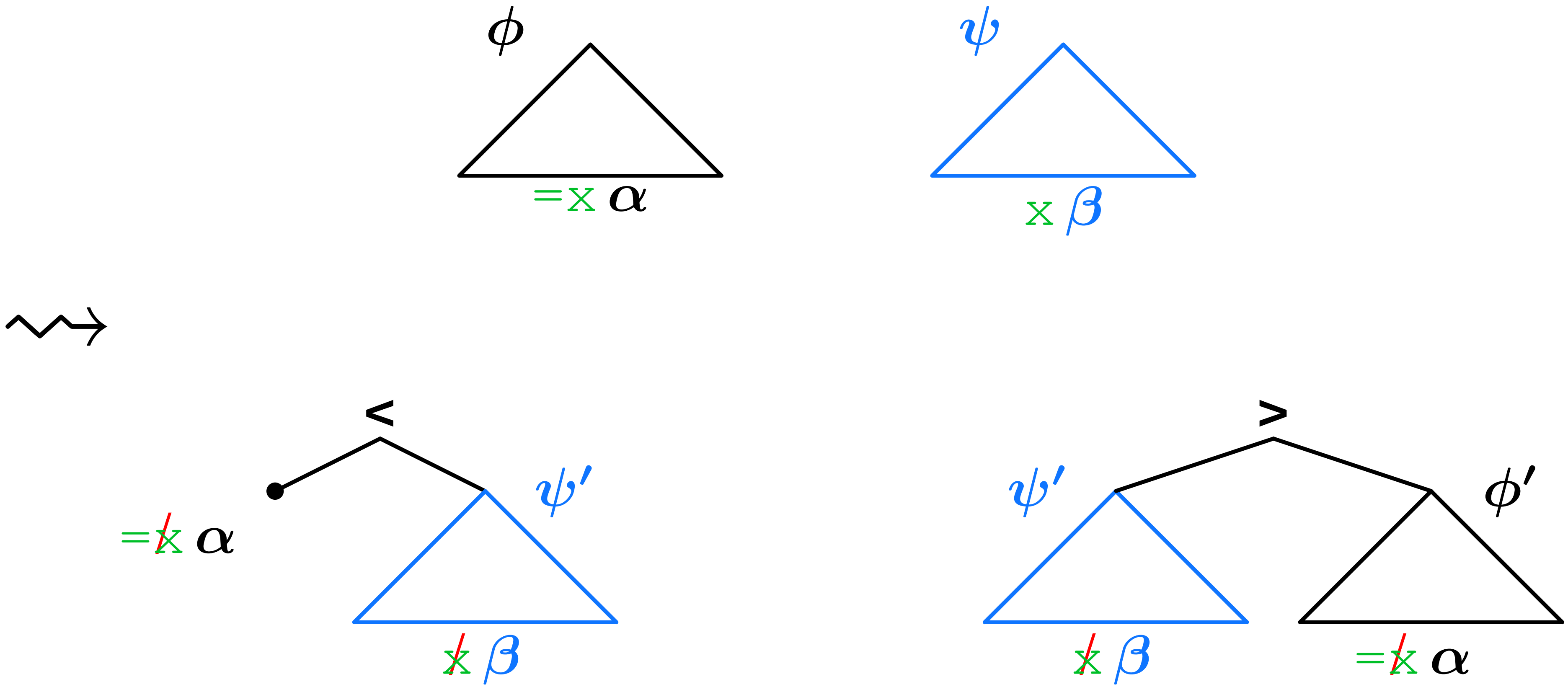


ϕ simple

ϕ complex

Structure building functions

$$\text{merge} : \text{Exp}(\text{Feat}) \times \text{Exp}(\text{Feat}) \xrightarrow{\text{part}} \text{Exp}(\text{Feat})$$

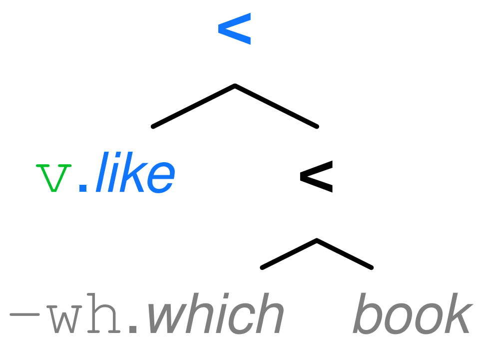


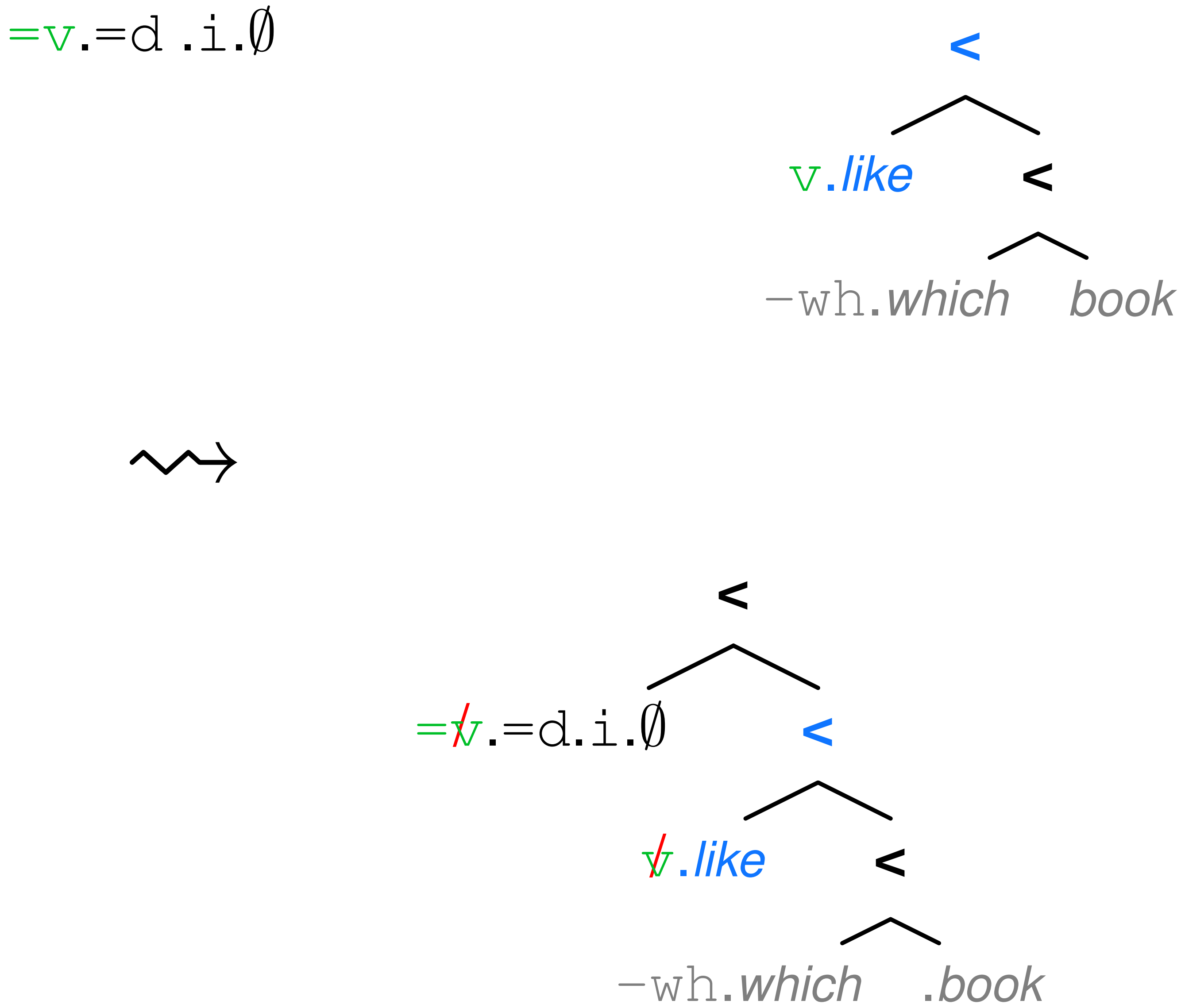
ϕ simple

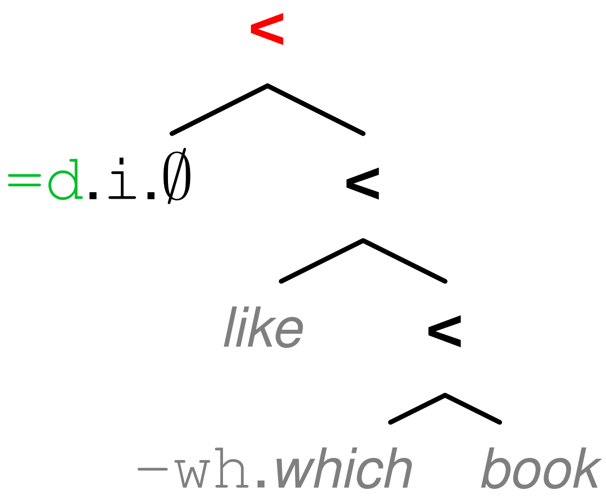
ϕ complex

merge (selecting tree is simple)

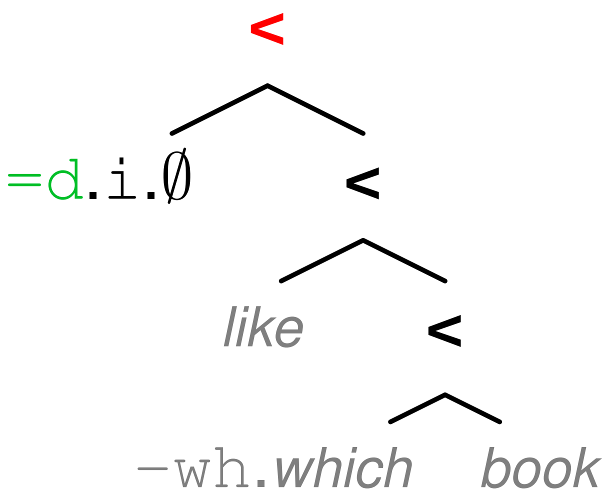
=v.=d.i.∅




=v.=d .i.∅
<
v.like
<
−wh.which
book
⇝
<
=v.=d.i.∅
<
v.like
<
−wh.which
.book

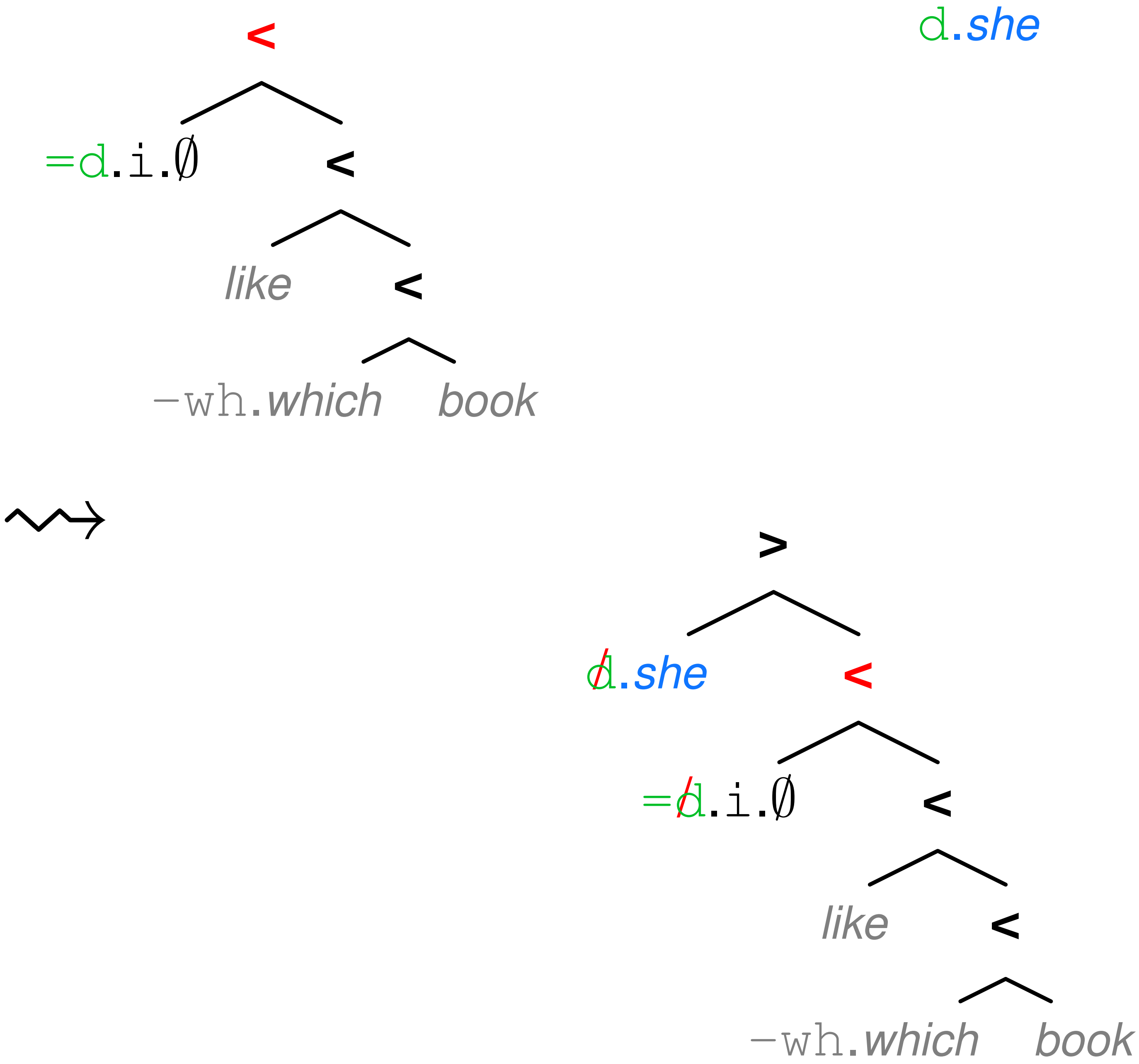

<
=d.i.∅
<
like
<
−wh.which
book

merge (selecting tree is complex)



d.*she*

merge (selecting tree is complex)



Structure building functions

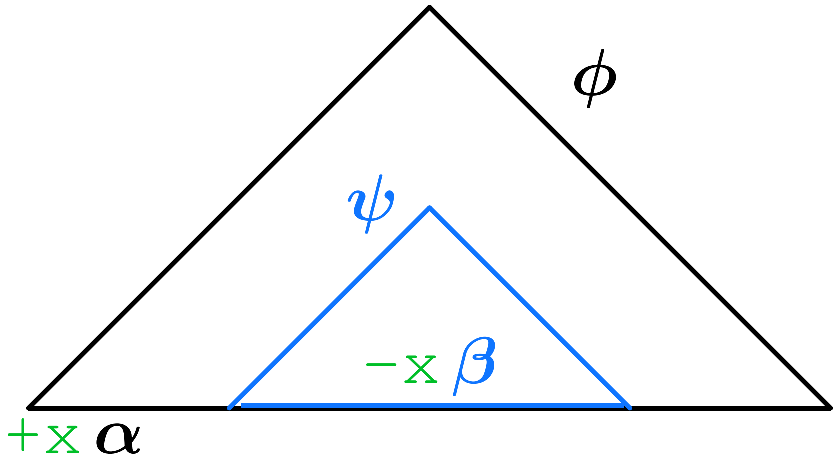
$\mathsf{move} : \mathsf{Exp}(\mathsf{Feat}) \xrightarrow{\text{part}} \mathsf{Exp}(\mathsf{Feat})$

$\phi \in \mathsf{Domain}(\mathsf{move}) \; :\Longleftrightarrow$

- ϕ displays feature $+\mathtt{x} \in$ Licensors
- there is exactly one maximal projection ψ within ϕ that displays feature $-\mathtt{x} \in$ Licensees (SMC)

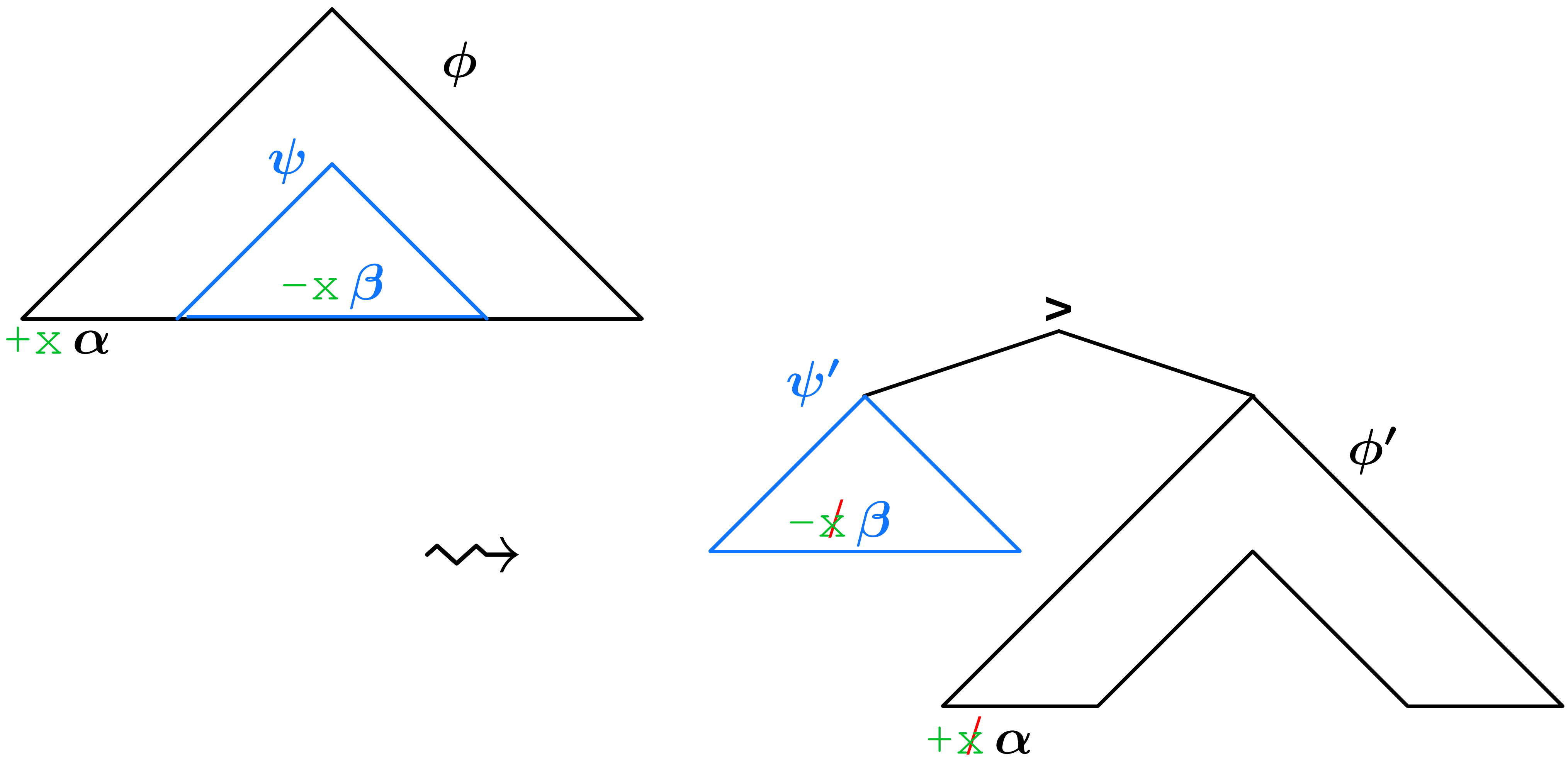
Structure building functions

$$\text{move} : \text{Exp(Feat)} \xrightarrow{\text{part}} \text{Exp(Feat)}$$

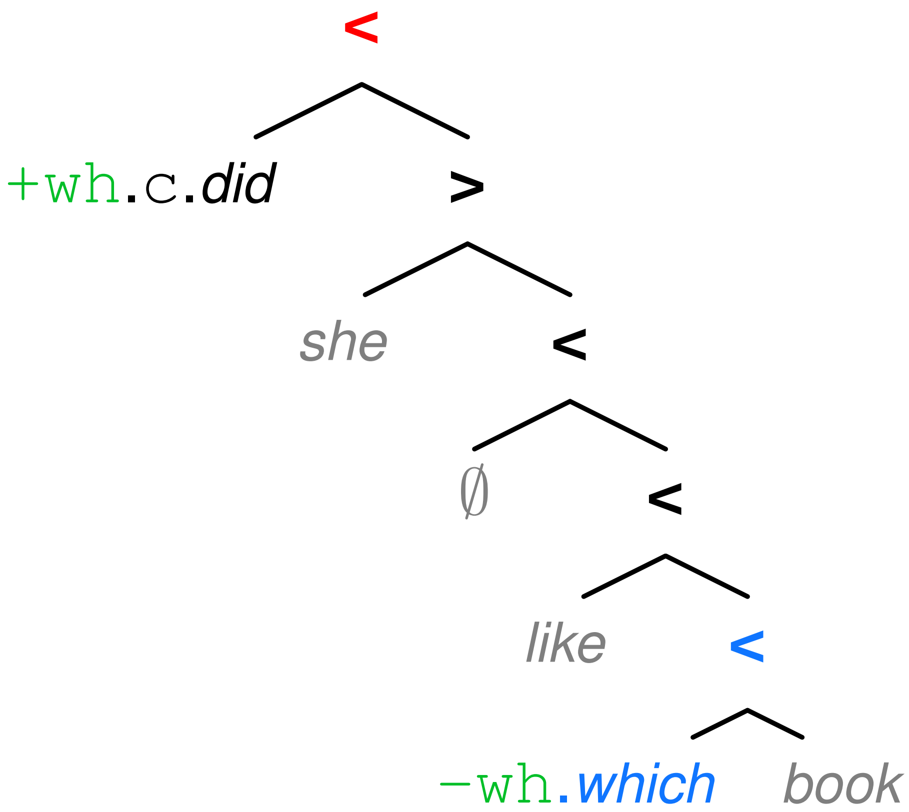


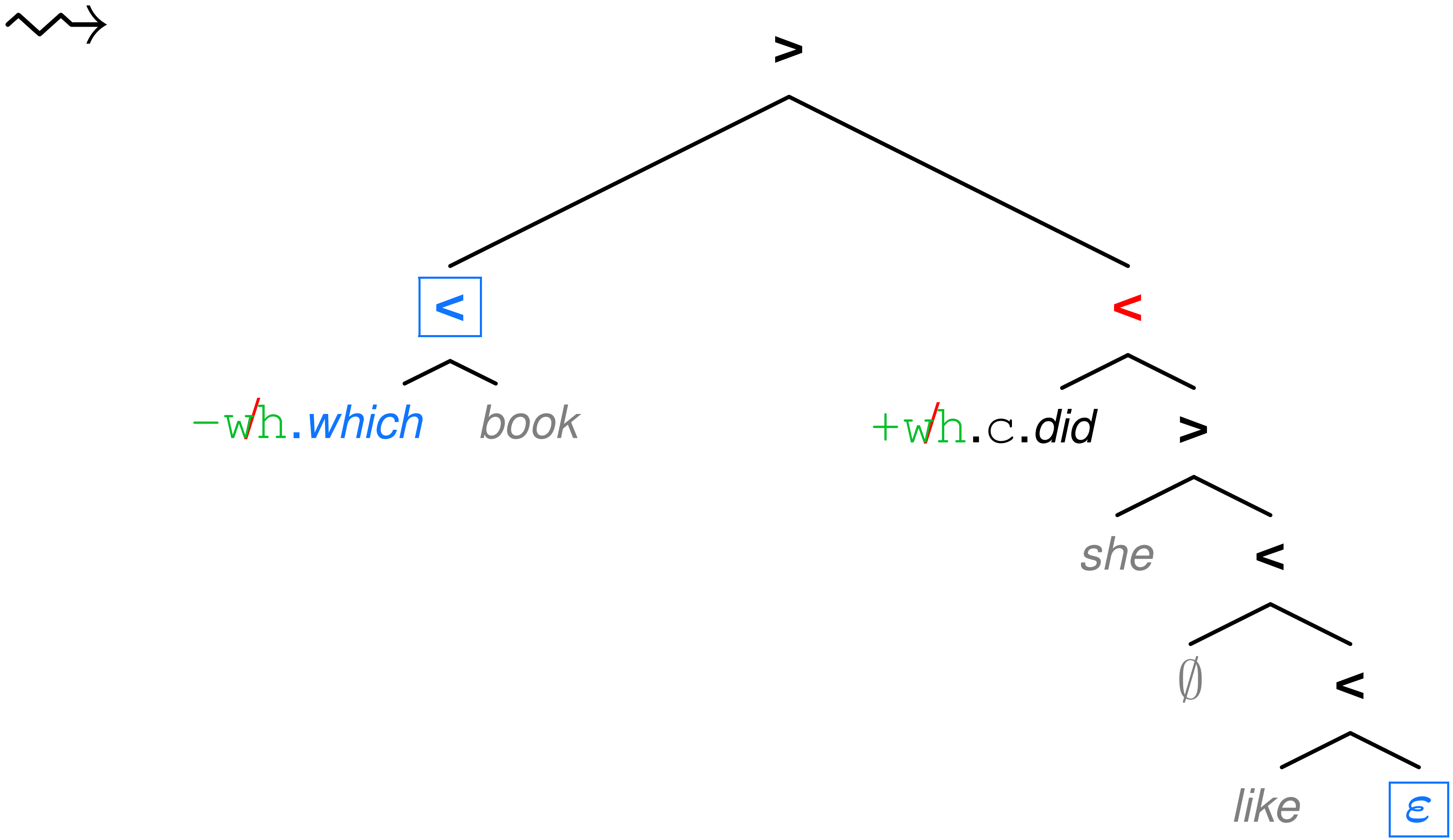
Structure building functions

move : Exp(Feat) $\xrightarrow{\text{part}}$ Exp(Feat)



move




⇝
>
<
<
−wh.which
book
+wh.c.did
>
she
<
∅
<
like
ε

$$\mathsf{G} = \langle \mathsf{Feat}, \mathsf{Lex}, \boldsymbol{\Omega}, \mathtt{c} \rangle$$

$$G = \langle \text{Feat}, \text{Lex}, \boldsymbol{\Omega}, \texttt{c} \rangle$$

- $\text{Feat} = \text{Syn} \cup \text{NonSyn}$

$$\text{Syn} = \text{Base} \cup \text{Select} \cup \text{Licensees} \cup \text{Licensors}$$

Base	Select	Licensees	Licensors
x	=x	−x	+x

$$G = \langle \text{Feat}, \text{Lex}, \Omega, c \rangle$$

- $\text{Feat} = \text{Syn} \cup \text{NonSyn}$

  $$\text{Syn} = \underset{x}{\text{Base}} \cup \underset{=x}{\text{Select}} \cup \underset{-x}{\text{Licensees}} \cup \underset{+x}{\text{Licensors}}$$

- Lex is a finite set of simple expressions with labels from $\text{Syn}^*\text{NonSyn}^*$ [lexicon]

An MG-lexicon

n.*book*	d.*she*	=d.v.*like*
=n.d.−wh.*which*	=v.=d.i.∅	=i.c.*that*
=i.+wh.c.*did*	i.*Mary_ read*	

$$\text{NonSyn}^* = \{\, \textit{book}\,,\ \textit{which}\,,\ \textit{Mary\_ read}\,,\ \ldots\,,\ \emptyset\,,\ \ldots \}$$

$G = \langle \mathsf{Feat}, \mathsf{Lex}, \Omega, \mathrm{c} \rangle$

- $\mathsf{Feat} = \mathsf{Syn} \cup \mathsf{NonSyn}$

  $\mathsf{Syn} = \mathsf{Base} \cup \mathsf{Select} \cup \mathsf{Licensees} \cup \mathsf{Licensors}$

  $\mathtt{x}$ $\quad$ $=\!\mathtt{x}$ $\quad$ $-\mathtt{x}$ $\quad$ $+\mathtt{x}$

- Lex is a finite set of simple expressions with labels from $\mathsf{Syn}^*\mathsf{NonSyn}^*$ [lexicon]

$G = \langle \text{Feat}, \text{Lex}, \Omega, c \rangle$

- $\text{Feat} = \text{Syn} \cup \text{NonSyn}$

  $\text{Syn} = \text{Base} \cup \text{Select} \cup \text{Licensees} \cup \text{Licensors}$

Base	Select	Licensees	Licensors
x	=x	−x	+x

- Lex is a finite set of simple expressions with labels from $\text{Syn}^*\text{NonSyn}^*$ [lexicon]

- $\Omega = \{ \text{merge}, \text{move} \}$ [structure building functions]

$G = \langle \text{Feat}, \text{Lex}, \Omega, c \rangle$

- $\text{Feat} = \text{Syn} \cup \text{NonSyn}$

  $\text{Syn} = \text{Base} \cup \text{Select} \cup \text{Licensees} \cup \text{Licensors}$

  Base: x — Select: $=x$ — Licensees: $-x$ — Licensors: $+x$

- Lex is a finite set of simple expressions with labels from $\text{Syn}^*\text{NonSyn}^*$ [lexicon]

- $\Omega = \{\text{merge}, \text{move}\}$ [structure building functions]

- $c \in \text{Base}$

Closure(G), the closure of an MG $G = \langle \mathrm{Feat}, \mathrm{Lex}, \mathbf{\Omega}, \mathrm{c} \rangle$,

is the closure of Lex under the operators from $\mathbf{\Omega}$.

Closure(G), the closure of an MG $\mathrm{G} = \langle \mathrm{Feat}, \mathrm{Lex}, \boldsymbol{\Omega}, \mathrm{c} \rangle$,

is the closure of Lex under the operators from $\boldsymbol{\Omega}$.

$\boldsymbol{\tau} \in$ Closure(G) is complete $:\Longleftrightarrow$

no leaf-label has an unchecked instance of a syntactic feature, except for the head-label, which is from $\{\mathrm{c}\}\,\mathrm{NonSyn}^*$.

Minimalist languages

Closure(G), the closure of an MG $G = \langle \text{Feat}, \text{Lex}, \Omega, c \rangle$,

is the closure of Lex under the operators from Ω.

$\tau \in$ Closure(G) is complete $:\Longleftrightarrow$

no leaf-label has an unchecked instance of a syntactic feature, except for the head-label, which is from $\{c\}\,\text{NonSyn}^*$.

The tree and string language generated by G

$$T(G) = \{ \tau \mid \tau \in \text{Closure}(G) \text{ and complete} \}$$

$$L(G) = \{ \text{yield}_{\text{NonSyn}}(\tau) \mid \tau \in T(G) \}$$

Generative capacity

- MGs are weakly equivalent to LCFRSs

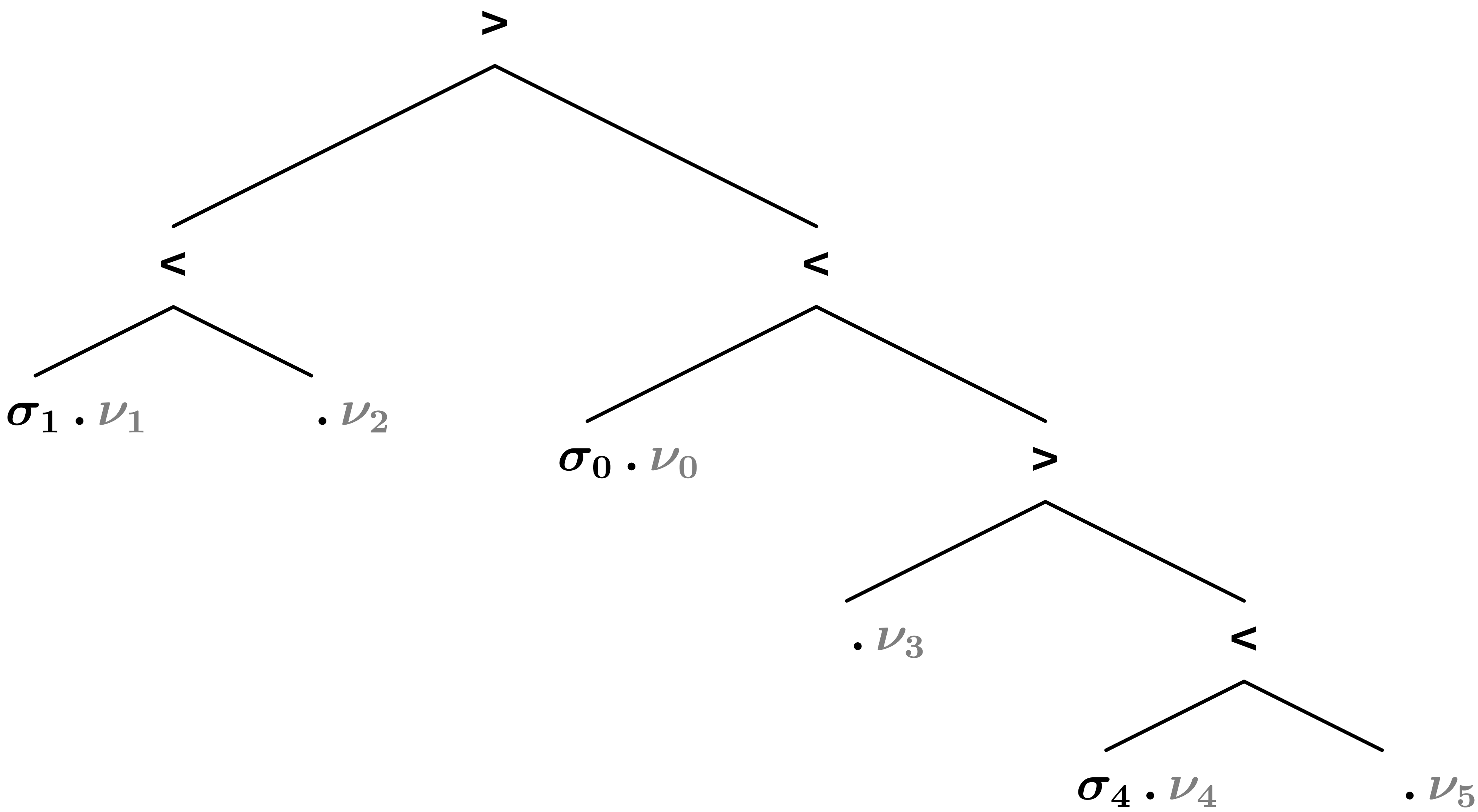
  (Michaelis 2001a, 2001b; Harkema 2001)

- MGs are weakly equivalent to LCFRSs

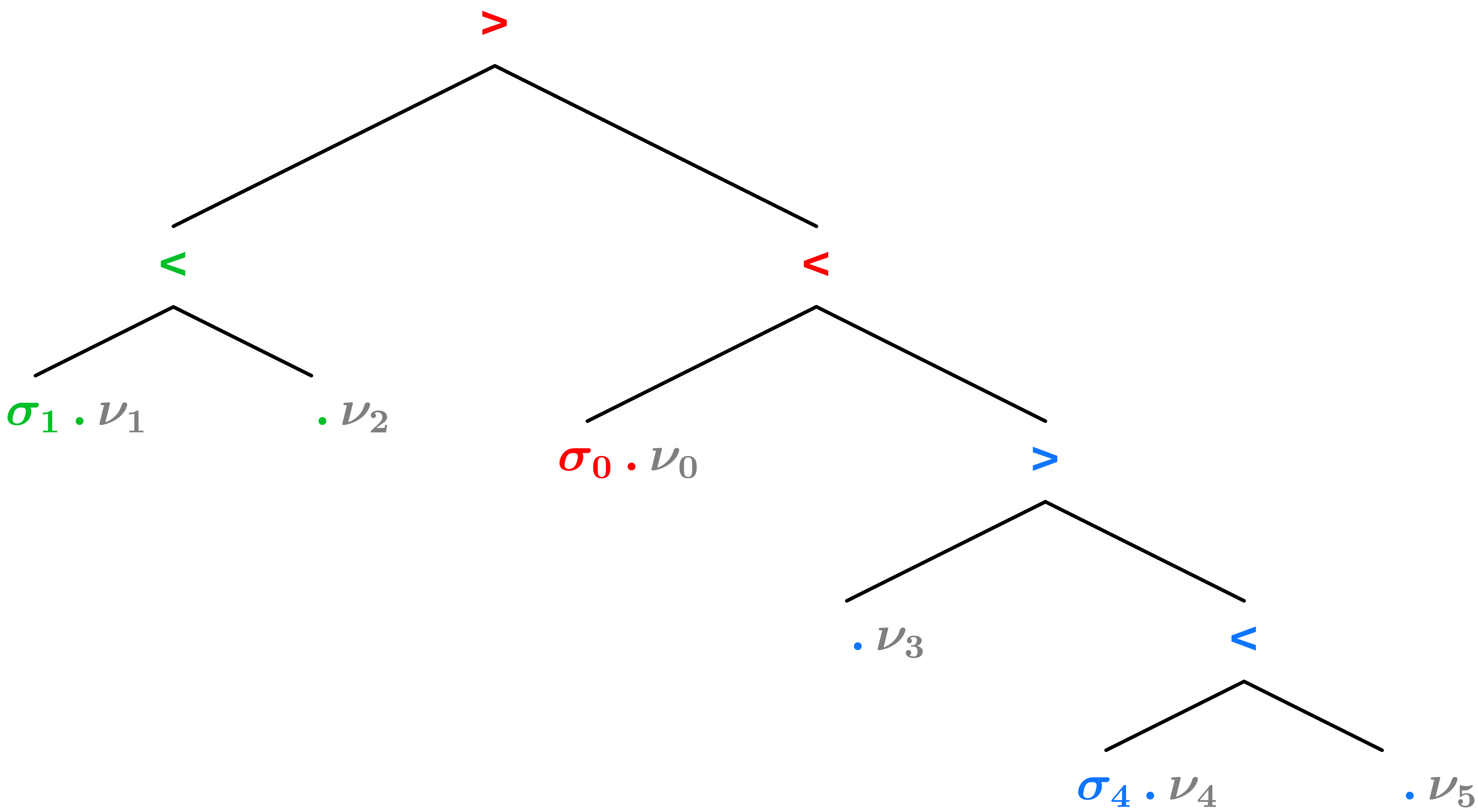
  (Michaelis 2001a, 2001b; Harkema 2001)

  Thus, in particular, for each MG there is an LCFRS deriving the same string language. This can be shown applying the methods which were developed in Michaelis 2001a for exactly this purpose, and which led to the succinct, chain-based MG-reformulation presented in Stabler & Keenan 2000 — reducing “classical” MGs to their “bare essentials.”

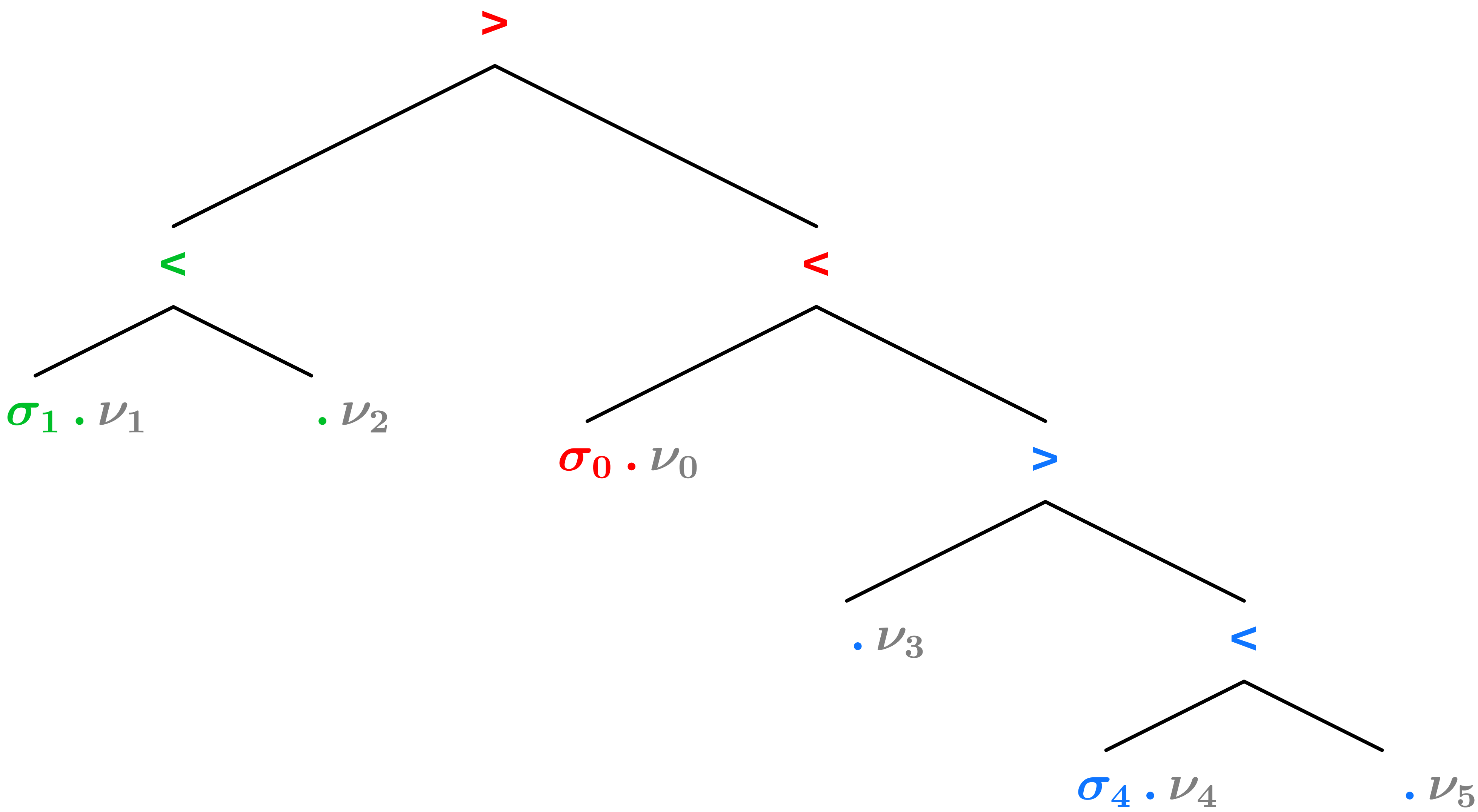
Reducing an MG



Reducing an MG

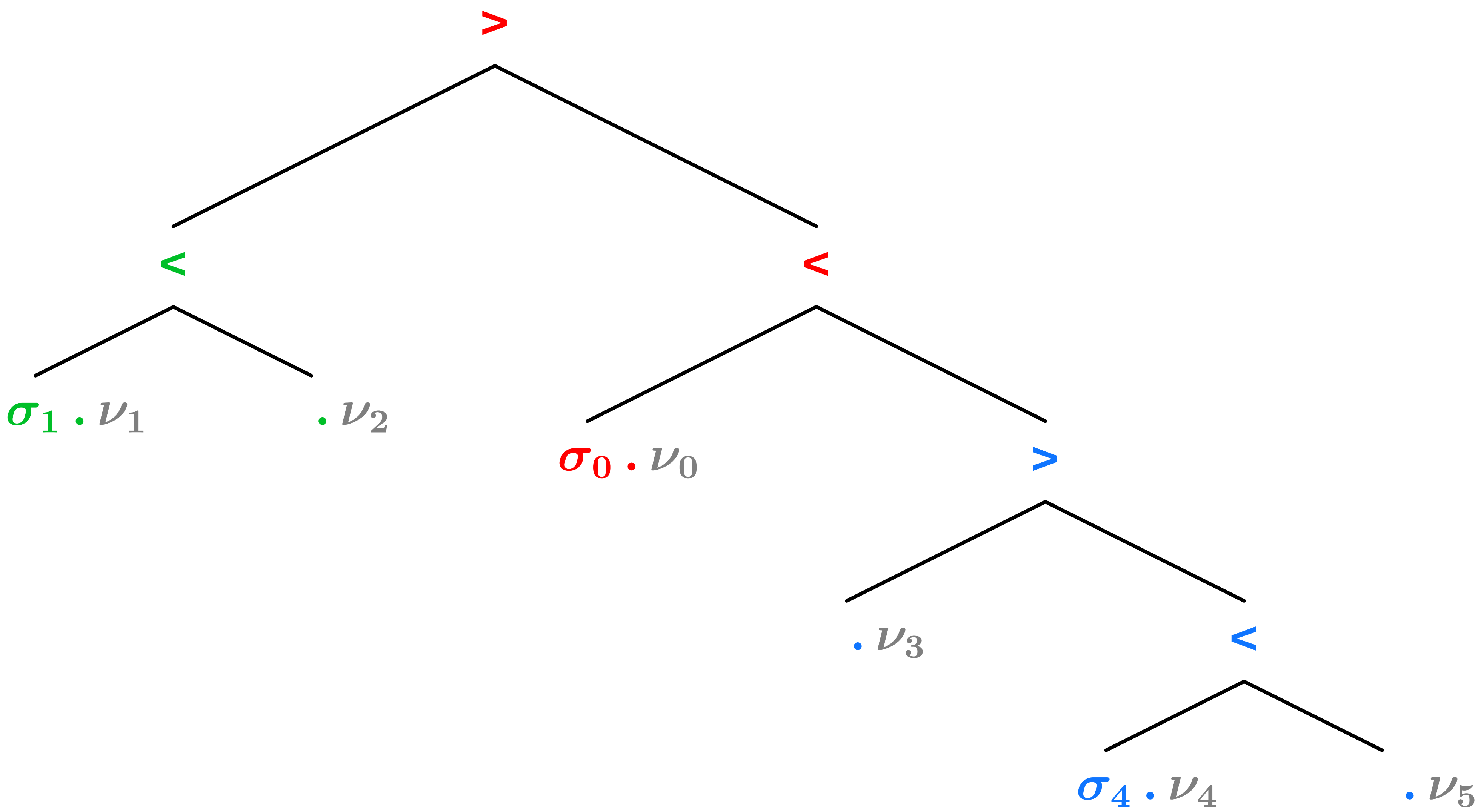


Reducing an MG



$$\left\langle \sigma_0 . \nu_0 \;,\; \sigma_1 . \nu_1 \nu_2 \;,\; \sigma_4 . \nu_3 \nu_4 \nu_5 \right\rangle$$

Reducing an MG



$$\left\langle\ \sigma_0 \cdot \quad ,\ \sigma_1 \cdot \quad ,\ \sigma_4 \cdot \quad \right\rangle$$

Reducing an MG

Let $\mathsf{G} = \langle \mathsf{Feat}, \mathsf{Lex}, \mathbf{\Omega}, \mathrm{c} \rangle$ be an MG

Reducing an MG

Let $\mathsf{G} = \langle \mathsf{Feat}, \mathsf{Lex}, \Omega, \mathtt{c} \rangle$ be an MG

$\tau \in \mathsf{Closure}(\mathsf{G})$ is relevant $:\Longleftrightarrow$

for each licensee $-\mathtt{x}$, there is at most one maximal projection in τ that displays $-\mathtt{x}$.

Reducing an MG

Let $\mathrm{G} = \langle \mathrm{Feat}, \mathrm{Lex}, \Omega, \mathrm{c} \rangle$ be an MG

$\tau \in \mathrm{Closure(G)}$ is relevant $:\Longleftrightarrow$

for each licensee $-\mathrm{x}$, there is at most one maximal projection in τ that displays $-\mathrm{x}$.

In fact, this kind of structure is characteristic of each $\tau \in \mathrm{Closure(G)}$ involved in creating a complete expression in G. Recall that 'move' is defined only in case that there is exactly one maximal projection in τ that has a particular licensee feature allowing the projection's "movement into a specifier position."

Reducing an MG

Let $G = \langle \text{Feat}, \text{Lex}, \Omega, c \rangle$ be an MG

$\tau \in \text{Closure}(G)$ is relevant $:\Longleftrightarrow$

for each licensee $-x$, there is at most one maximal projection in τ that displays $-x$.

In fact, this kind of structure is characteristic of each $\tau \in \text{Closure}(G)$ involved in creating a complete expression in G. Recall that 'move' is defined only in case that there is exactly one maximal projection in τ that has a particular licensee feature allowing the projection's "movement into a specifier position."

$\text{RClosure}(G)$ is the set of all relevant $\tau \in \text{Closure}(G)$

A finite partition of RClosure(G)

Basic idea : consider $\tau \in \text{RClosure}(G)$

- Reduce τ to a tuple such that for each maximal projection displaying an unchecked syntactic feature, there is exactly one component of the tuple consisting of the projection's head-label, but with the suffix of non-syntactic features truncated.

A finite partition of RClosure(G)

Basic idea : consider $\tau \in$ RClosure(G)

- Reduce τ to a tuple such that for each maximal projection displaying an unchecked syntactic feature, there is exactly one component of the tuple consisting of the projection's head-label, but with the suffix of non-syntactic features truncated.

$\rightsquigarrow$ only finitely many equivalence classes

Relevance :
The resulting tuple has at most m+1 components , $m = |\text{Licensees}|$.

Structure building by cancellation of features :
Each tuple component is the suffix of the syntactic prefix of the label of a lexical item.

A finite partition of RClosure(G)

Basic idea : consider $\tau \in$ RClosure(G)

- Reduce τ to a tuple such that for each maximal projection displaying an unchecked syntactic feature, there is exactly one component of the tuple consisting of the projection's head-label, but with the suffix of non-syntactic features truncated.

$\rightsquigarrow$ only finitely many equivalence classes

Relevance :
The resulting tuple has at most m+1 components , $m = |\text{Licensees}|$.

Structure building by cancellation of features :
Each tuple component is the suffix of the syntactic prefix of the label of a lexical item.

$\rightsquigarrow$ regarding the partition, applications of 'merge' and 'move' do not depend on the chosen representatives

A weakly equivalent LCFRS

- The nonterminating rules :

  (1) $\mathsf{T} \longrightarrow \mathsf{merge}_{\mathsf{U},\mathsf{V}}(\mathsf{U},\mathsf{V})$

  T , U and V the “representatives” of some $\boldsymbol{\tau}$, $\boldsymbol{\upsilon}$ and $\boldsymbol{\phi} \in$ RClosure , respectively, such that $\boldsymbol{\tau} =$ merge$(\boldsymbol{\upsilon}, \boldsymbol{\phi})$.

  (2) $\mathsf{T} \longrightarrow \mathsf{move}_{\mathsf{U}}(\mathsf{U})$

  T and U the “representatives” of some $\boldsymbol{\tau}$ and $\boldsymbol{\upsilon} \in$ RClosure , respectively, such that $\boldsymbol{\tau} =$ move$(\boldsymbol{\upsilon})$.

- The terminating rules :

  (3) $\mathsf{T} \longrightarrow \boldsymbol{\nu}$

  T the “representative” of $\boldsymbol{\tau} \in$ Lex with label $\boldsymbol{\sigma\nu}$, where $\boldsymbol{\sigma} \in$ Syn , and $\boldsymbol{\nu} \in$ NonSyn .

- "Reconstruction" of the non-syntactic material is possible by means of the regular functions of the LCFRS :

$$\mathrm{T} \overset{*}{\underset{G}{\Longrightarrow}} \langle \nu_0, \ldots, \nu_{\mathrm{m}} \rangle \in \mathrm{Strings(NonSyn)}^{\mathrm{m+1}}$$

(a) T the "representative" of $\tau \in$ RClosure(G) .

(b) For $\{ \mathrm{l}_1 , \ldots \mathrm{l}_{\mathrm{m}} \}$, an enumeration of Licensees,

ν_0 is the "non-extractable" part of the non-syntactic yield of τ , i.e. , that part of the non-sytactic yield of τ which by no means would be pied-piped if some proper subtree of τ became subject to movement.

For $1 \leq \mathrm{i} \leq \mathrm{m}$,

if there is no subtree of τ displaying licensee l_i , then $\nu_\mathrm{i} = \varepsilon$.

Otherwise , ν_i is the "non-extractable" part of the non-syntactic yield of τ_i , the subtree of τ displaying licensee l_i .

References

Philippe de Groote, Glyn Morrill and Christian Retoré (eds.). 2001. *Logical Aspects of Computational Linguistics (LACL '01)*, Lecture Notes in Artificial Intelligence Vol. 2099. Springer, Berlin, Heidelberg.

Henk Harkema. 2001. A characterization of minimalist languages. In: de Groote et al. 2001, pp. 193–211.

Jens Michaelis. 2001a. Derivational minimalism is mildly context–sensitive. In: M. Moortgat (ed.), *Logical Aspects of Computational Linguistics (LACL '98)*, Lecture Notes in Artificial Intelligence Vol. 2014, pp. 179–198. Springer, Berlin, Heidelberg. Also available at `http://www.ling.uni-potsdam.de/~michael/papers.html`.

Jens Michaelis. 2001b. Transforming linear context–free rewriting systems into minimalist grammars. In: de Groote et al. 2001, pp. 228–244. Also available at `http://www.ling.uni-potsdam.de/~michael/papers.html`.

Edward P. Stabler. 1997. Derivational minimalism. In: C. Retoré (ed.), *Logical Aspects of Computational Linguistics (LACL '96)*, Lecture Notes in Artificial Intelligence Vol. 1328, pp. 68–95. Springer, Berlin, Heidelberg.

Edward P. Stabler and Edward L. Keenan. 2000. Structural similarity. In: *Algebraic Methods in Language Processing.* Proceedings of the 16th Twente Workshop on Language Technology (TWLT 16) joint with the 2nd AMAST Workshop on Language Processing, Iowa City, IA, pp. 251–265.