An Introduction to Minimalist Grammars:

Formalism

(July 20, 2009)

Gregory Kobele
Humboldt Universität zu Berlin
University of Chicago
kobele@rz.hu-berlin.de

Jens Michaelis
Universität Bielefeld
jens.michaelis@uni-bielefeld.de

Introduction

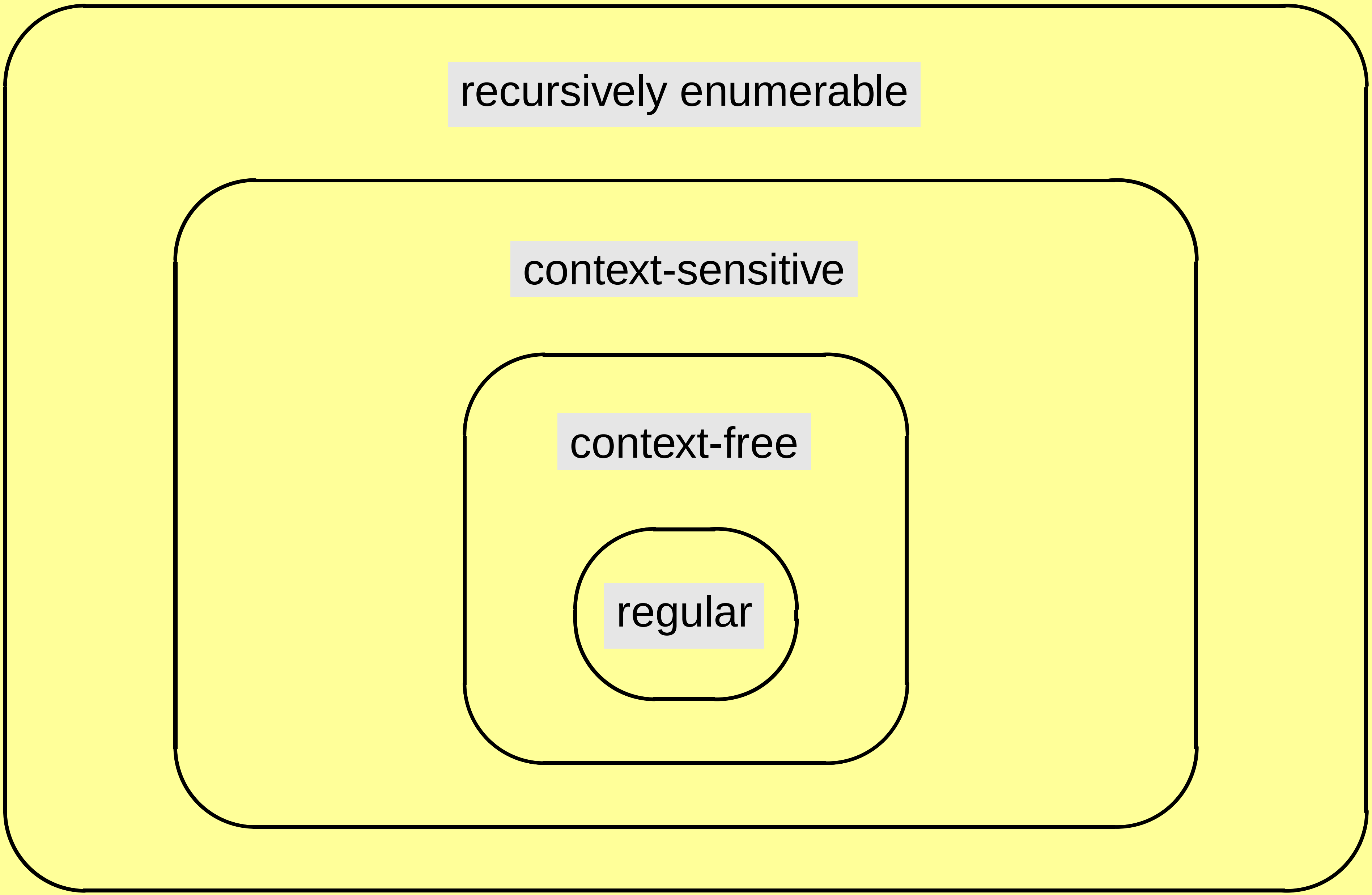
- Research on natural language syntax in terms of transformational grammar (TG) has always been accompanied by questions on the complexity of the individual grammars allowed by the general theory.

- From the perspective of formal language theory, special emphasis has more generally been placed on two specific aspects:

  a) the location within the Chomsky hierarchy of any grammars supposed to be adequate models for natural languages,

  b) the complexity of the parsing problem for such grammars.

Chomsky hierarchy



Introduction

- Peters and Ritchie (1971, 1973) proved the *Aspects*-model of TG (Chomsky 1965) to be Turing equivalent.

  ⇒ For every recursively enumerable set (i.e., type 0-language), there is a particular *Aspects*-grammar deriving it.

- Subsequently, locality conditions (LCs) — established in Ross 1967 and Chomsky 1973, 1977 — were studied intensively in work by many others searching for ways to reduce expressive power.

♦ See, e.g., Huang (1982), Chomsky (1986), Rizzi (1990), Cinque (1991), Manzini (1992), Müller & Sternefeld (1993), Szabolcsi & Zwarts (1993).

Introduction

- Complexity results, however, have been largly absent for those grammars with LC-add-ons. (Notable exception: Rogers 1998.)

  The picture changed with minimalist grammars (MGs) (Stabler 1997, 1999) as a formalization of “minimalism” (Chomsky 1995).

  MGs in that format constitute a mildly context-sensitive grammar formalism in the sense of Joshi 1985 (Michaelis 1998, 2001).

- Two crucial features of MGs helped achieving this result:
  - the resource sensitivity (encoded in the checking mechanism),
  - the implementation of the shortest move condition (SMC).

Mild context-sensitivity (Joshi 1985)

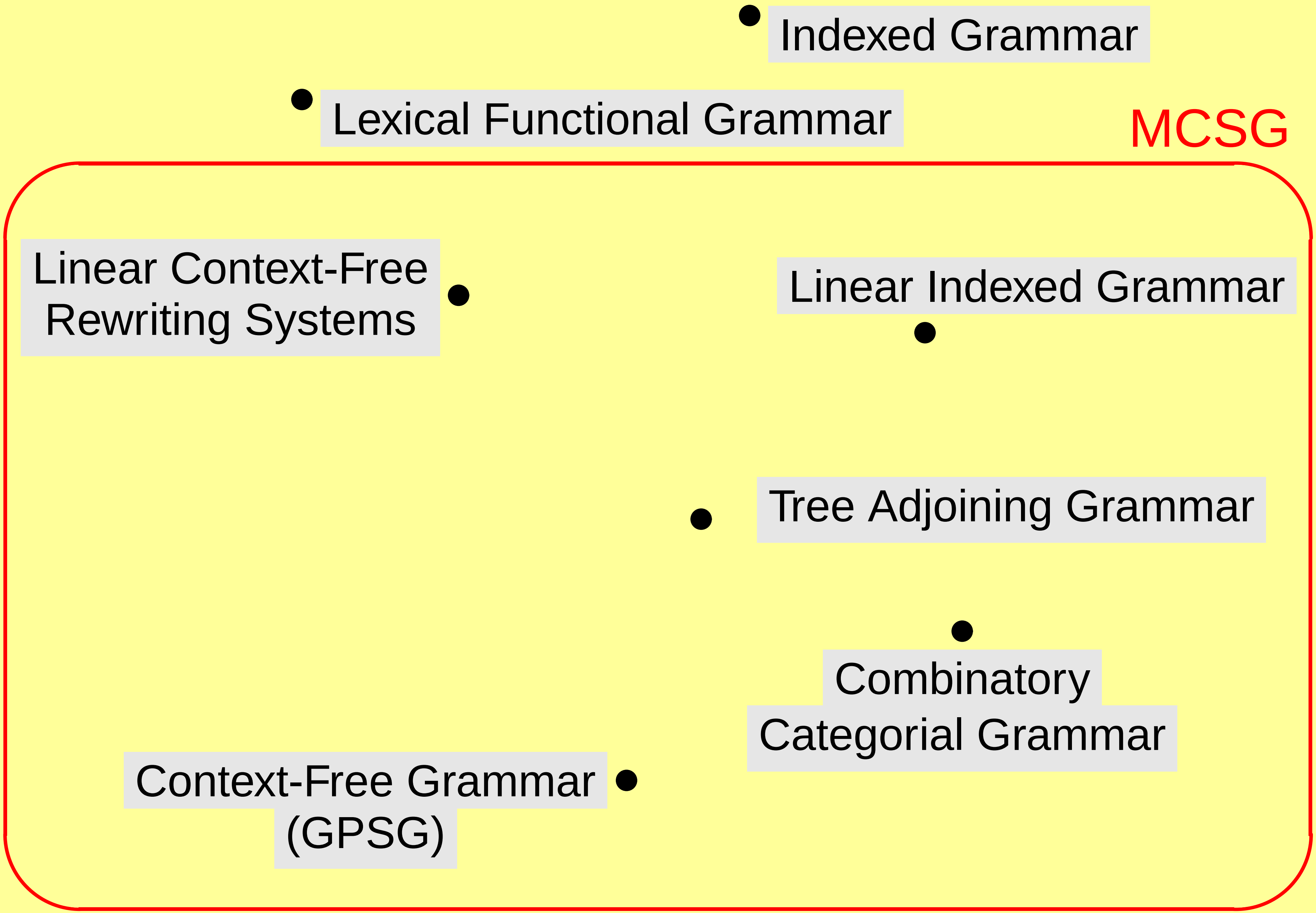
- A concept motivated by the intention of characterizing a narrow class of formal grammars which are
  - "only slightly more powerful than context-free grammars,"
  - nevertheless allowing for natural language descriptions in a linguistically significant way.

Mild context-sensitivity (Joshi 1985)

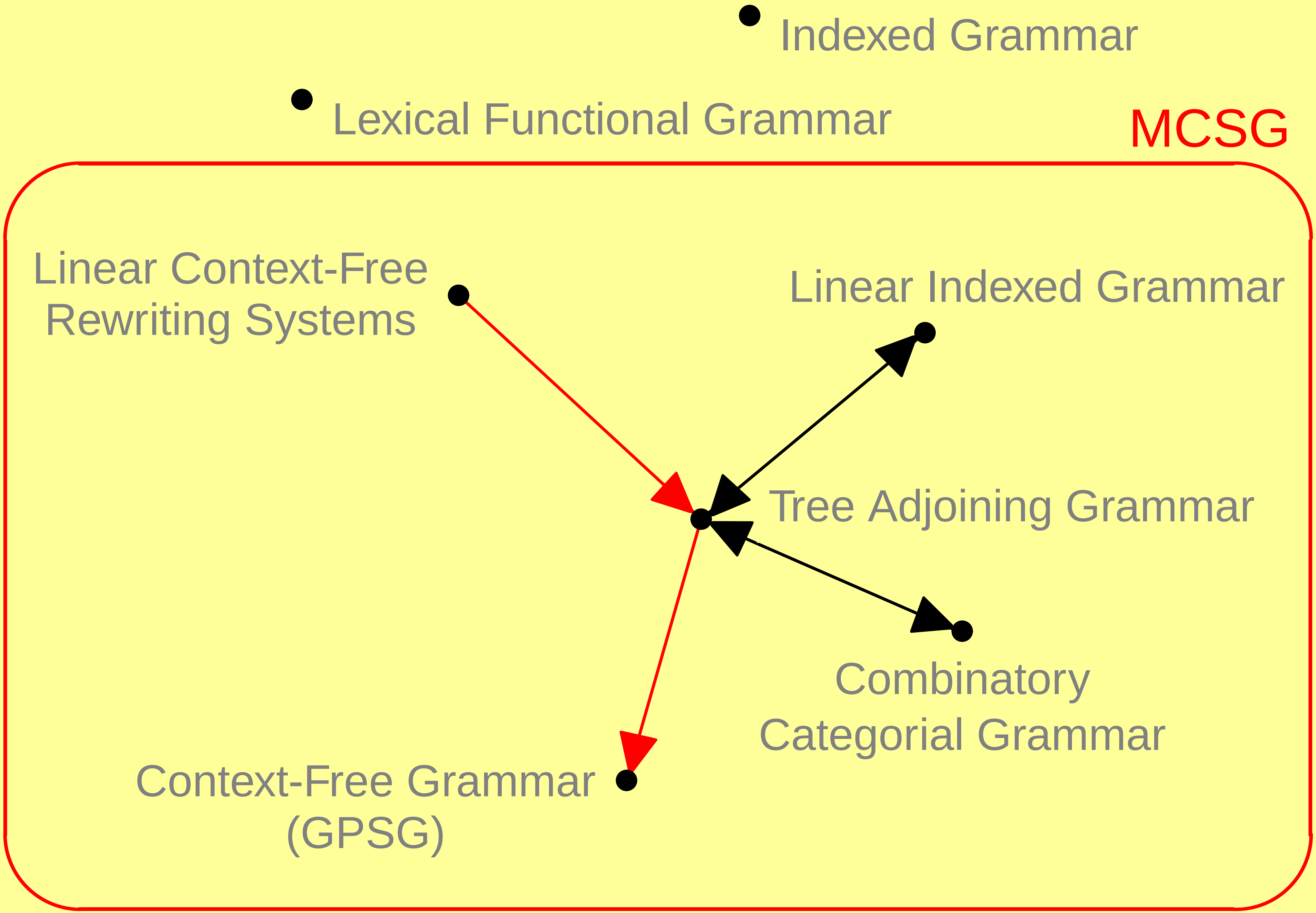
- A concept motivated by the intention of characterizing a narrow class of formal grammars which are
  - "only slightly more powerful than context-free grammars,"
  - nevertheless allowing for natural language descriptions in a linguistically significant way.

- A mildly context-sensitive grammar (MCSG) fulfills three criteria, understood as a "rough characterization" (cf. Joshi 1985, p. 225).
  1) Parsing problem is solvable in polynomial time.
  2) Language has the constant growth property.
  3) Finite upper bound on the number of different instantiations of factorized cross-serial dependencies occurring in any sentence.

MCSG-landscape



MCSG-landscape



Minimalist grammars (1)

- Minimalist grammars (MGs) provide an attempt at a rigorous algebraic formalization (of some) of the perspectives adopted in the minimalist branch of generative grammar.

  Work on MGs defined in this sense can be seen as having led to a realignment of “grammars found ‘useful’ by linguists” and formal complexity theory.

Two types of locality conditions (LCs)

- In particular, a study in terms of MGs can enhance our understanding of the complexity/restrictiveness of LCs.

  In fact, such a study shows that, though the addition of an LC may reduce complexity in an appropriate and intuitively natural way, it does not necessarily do so, and may even increase complexity.

- One can formally distinguish two types of LCs.

Two types of locality conditions (LCs)

- Intervention-based LCs (ILCs)
  - often in terms of minimality constraints, such as minimal link, minimal chain, shortest move, attract closest etc.

    in MGs: shortest move condition (SMC) (Stabler 1997, 1999)

- Containment-based LCs (CLCs)
  - often in terms of (generalized) grammatical functions, such as adjunct islands, specifier islands, subject island etc.

    in MGs: specifier island condition (SPIC) (Stabler 1999)

    in MGs: adjunct island condition (AIC)

    (Frey & Gärtner 2002, Gärtner & Michaelis 2003)

Two types of locality conditions (LCs)

- Intervention-based LCs (ILCs)
  - often in terms of minimality constraints

    essential structure: $[\ldots \alpha \ldots [\ldots \beta \ldots \gamma \ldots]]$

- Containment-based LCs (CLCs)
  - often in terms of (generalized) grammatical functions

    essential structure: $[\ldots \alpha \ldots [_{\beta} \ldots \gamma \ldots]]$

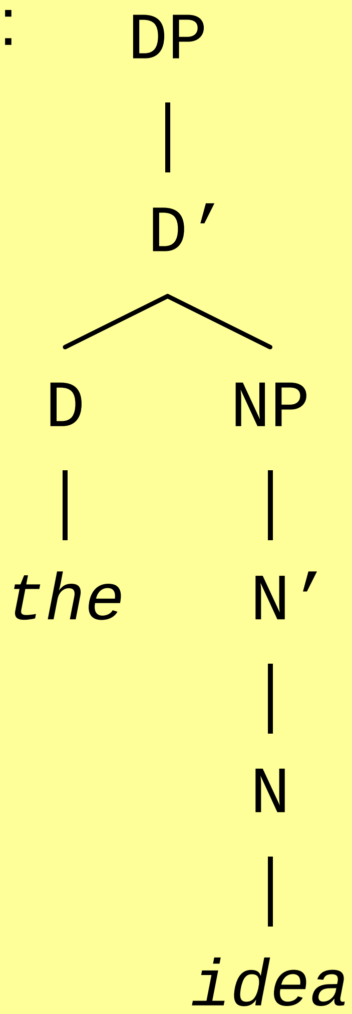
Minimalist grammars (1)

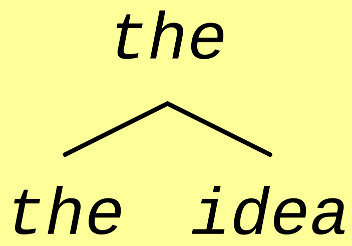
- More generally, MGs are capable of integrating (if needed) a variety of (arguably) “odd” items from the syntactician’s toolbox such as:
  - head movement (Stabler 1997, 2001)
  - (strict) remnant movement (Stabler 1997, 1999)
  - affix hopping (Stabler 2001)
  - adjunction and scrambling (Frey & Gärtner 2002)
  - late adjunction and extraposition (Gärtner & Michaelis 2003)
    — to some extent without rise in generative power
  - copy-movement (Kobele 2006)
  - wh-clustering (Gärtner & Michaelis 2007)

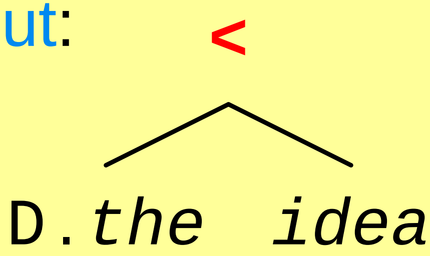
Minimalist expressions

- The objects generated by an MG are called minimalist expressions.

Minimalist expressions

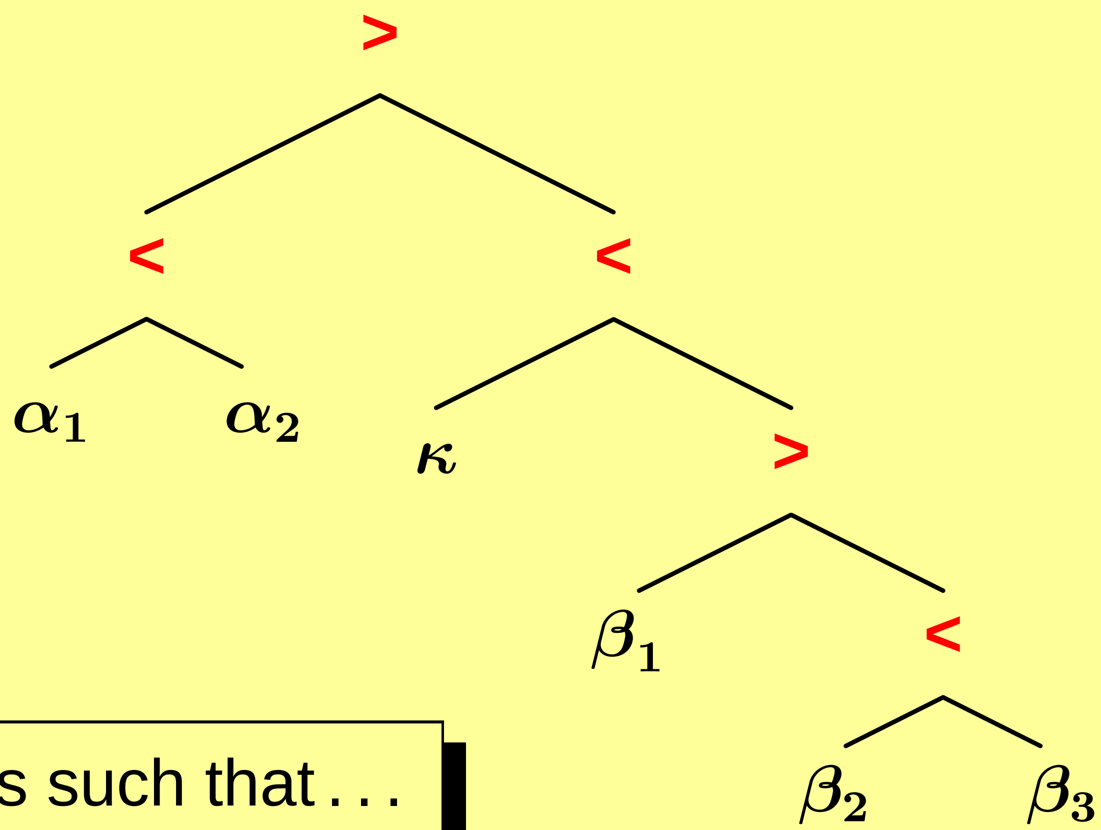
- Not: 

  Not: 

  But: 

The < “points towards” the projecting daughter, and thus — by means of transitivity — towards the head of the phrase.

Minimalist expressions



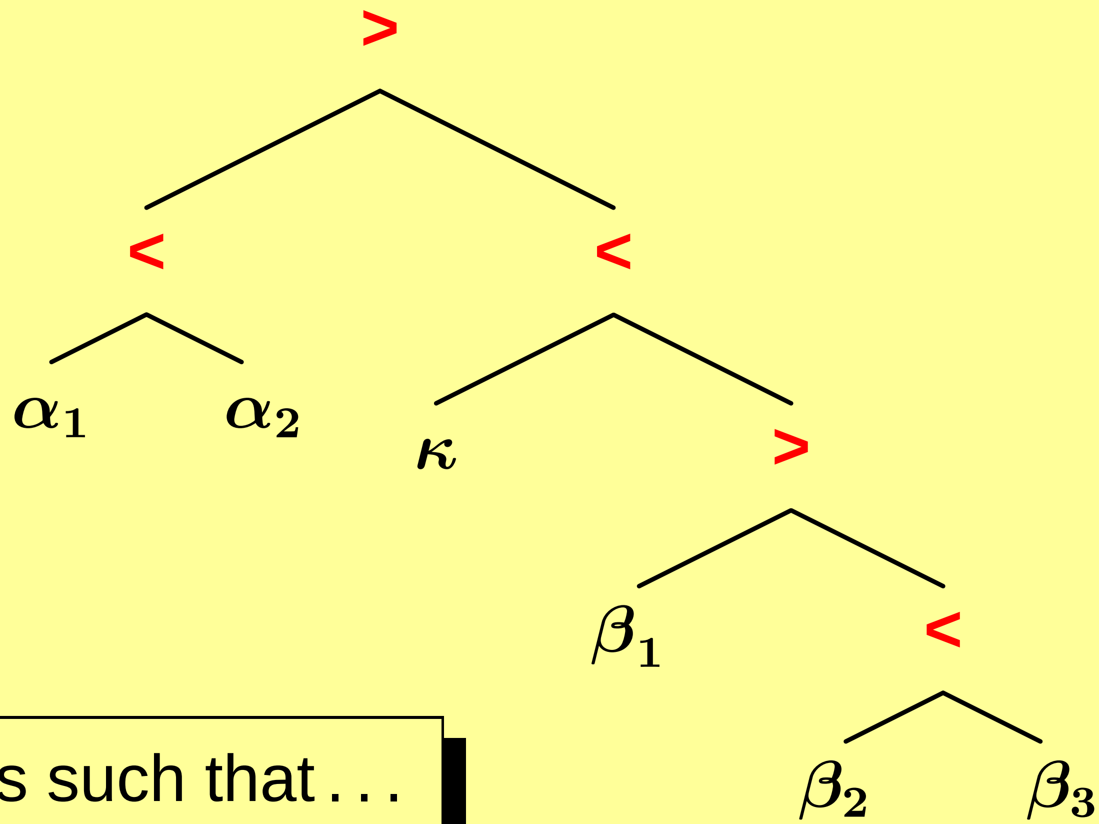
finite, binary labeled trees such that . . .

- non-leaf-labels are from $\{<, >\}$ [“projection”]

Minimalist expressions

< “left daughter projects”

> “right daughter projects”



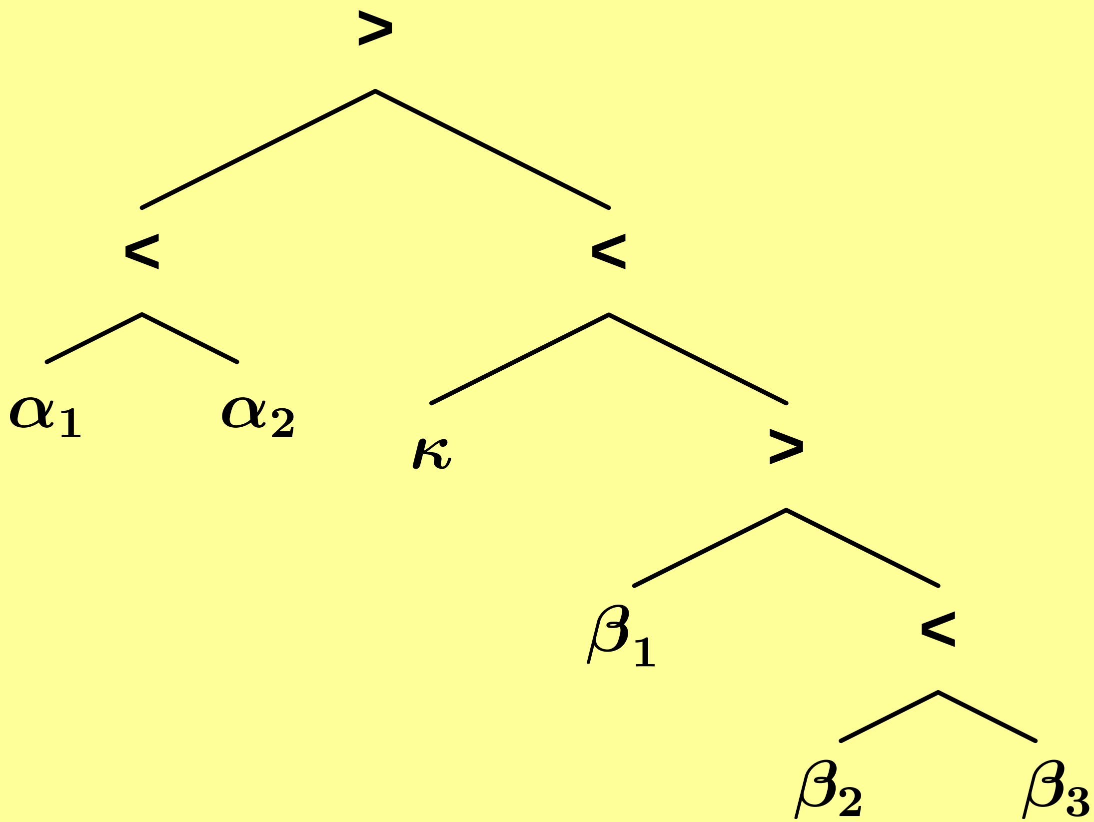
finite, binary labeled trees such that . . .

- non-leaf-labels are from $\{$ **<** , **>** $\}$ [“projection”]

Minimalist expressions

< “left daughter projects”

> “right daughter projects”



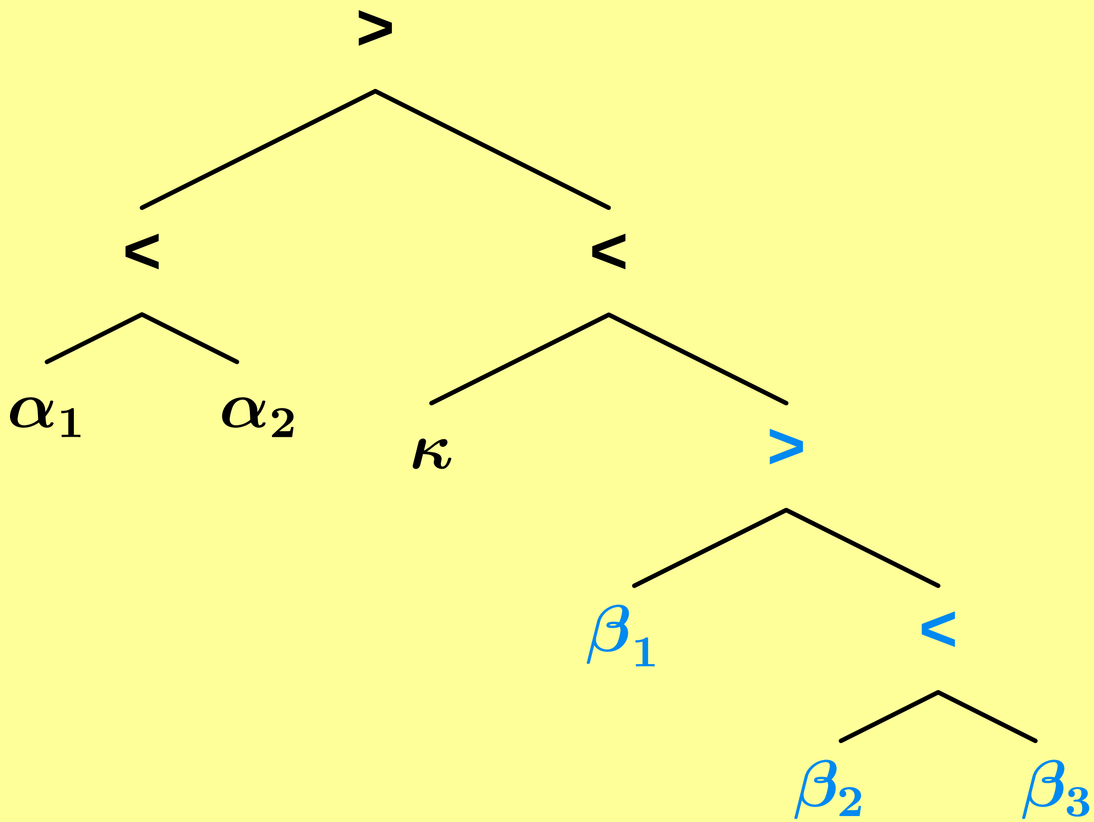
maximal projections :

each subtree whose root does not project

Minimalist expressions

< “left daughter projects”

> “right daughter projects”



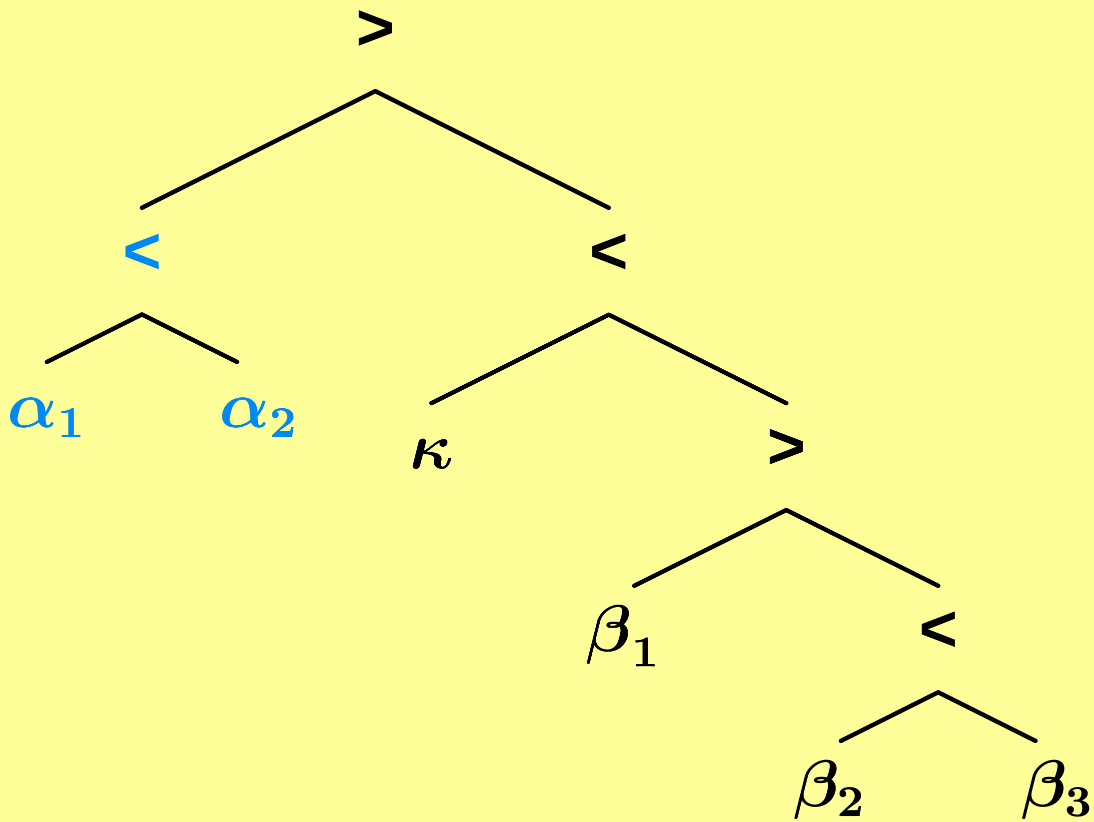
maximal projections:

each subtree whose root does not project

Minimalist expressions

< “left daughter projects”

> “right daughter projects”



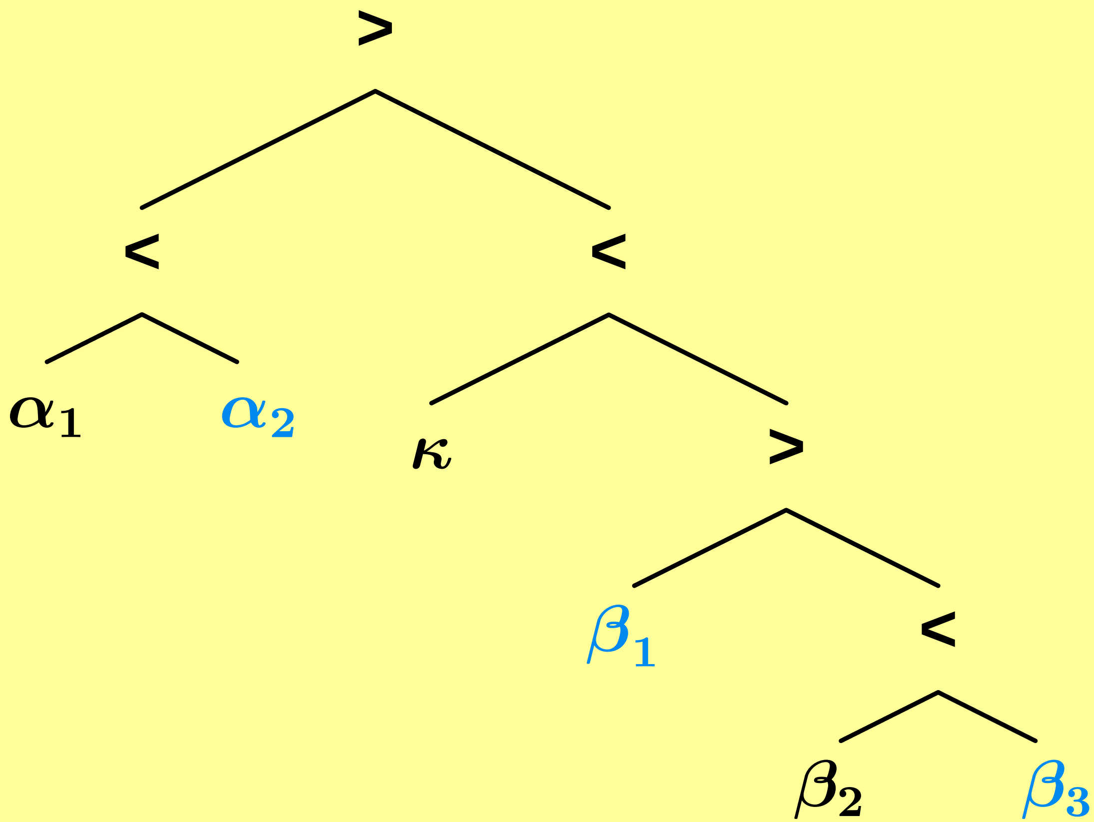
maximal projections :

each subtree whose root does not project

Minimalist expressions

< “left daughter projects”

> “right daughter projects”



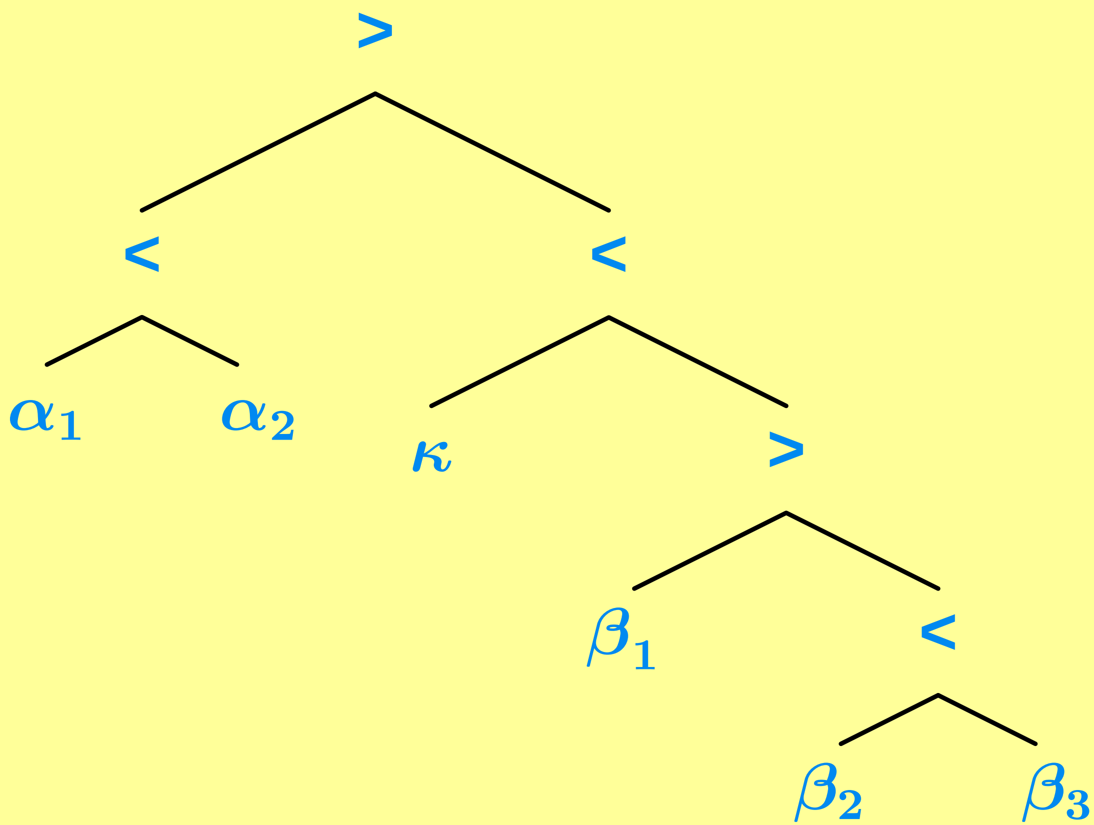
maximal projections :

each subtree whose root does not project

Minimalist expressions

< “left daughter projects”

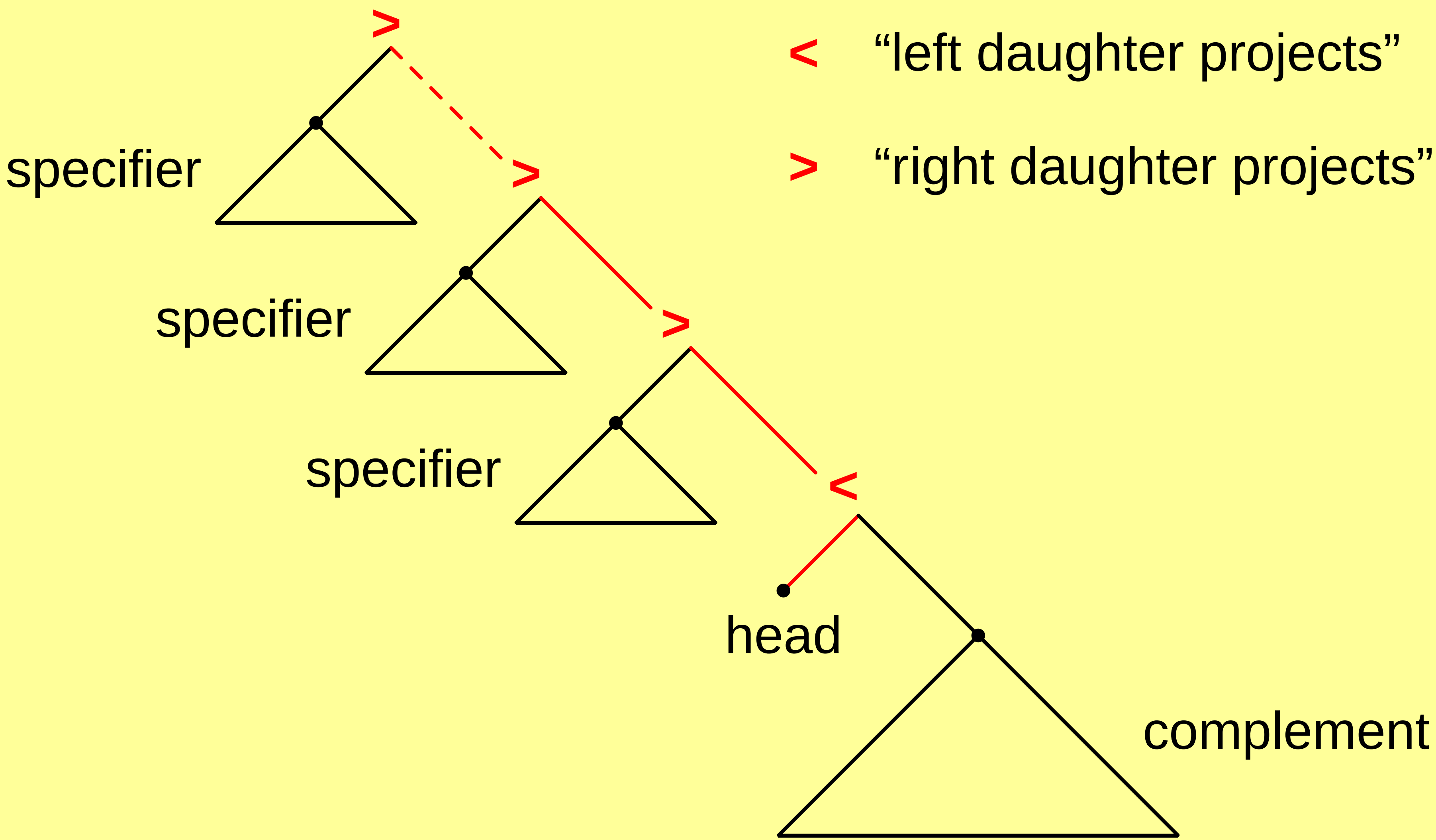
> “right daughter projects”



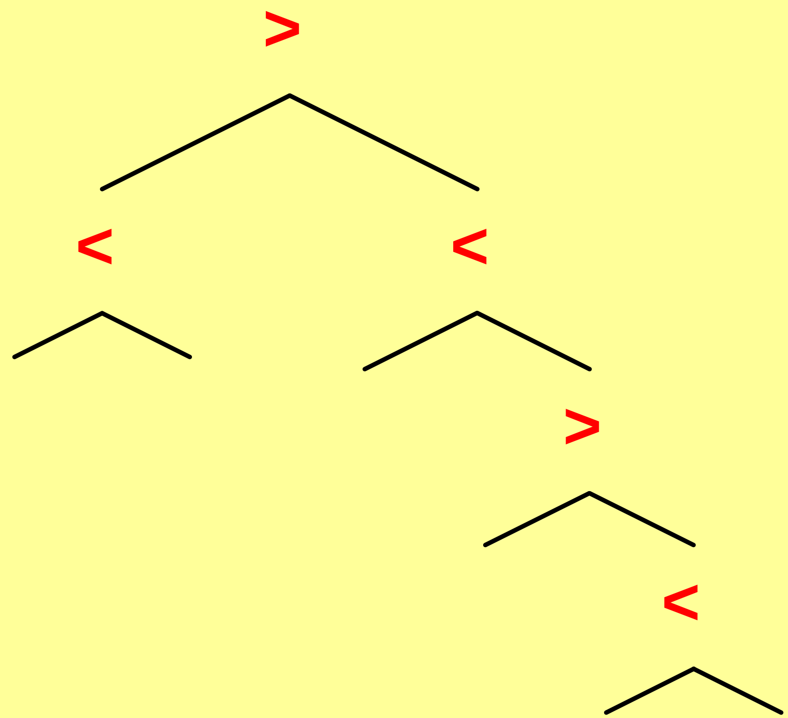
maximal projections :

each subtree whose root does not project

Minimalist expressions



Minimalist expressions



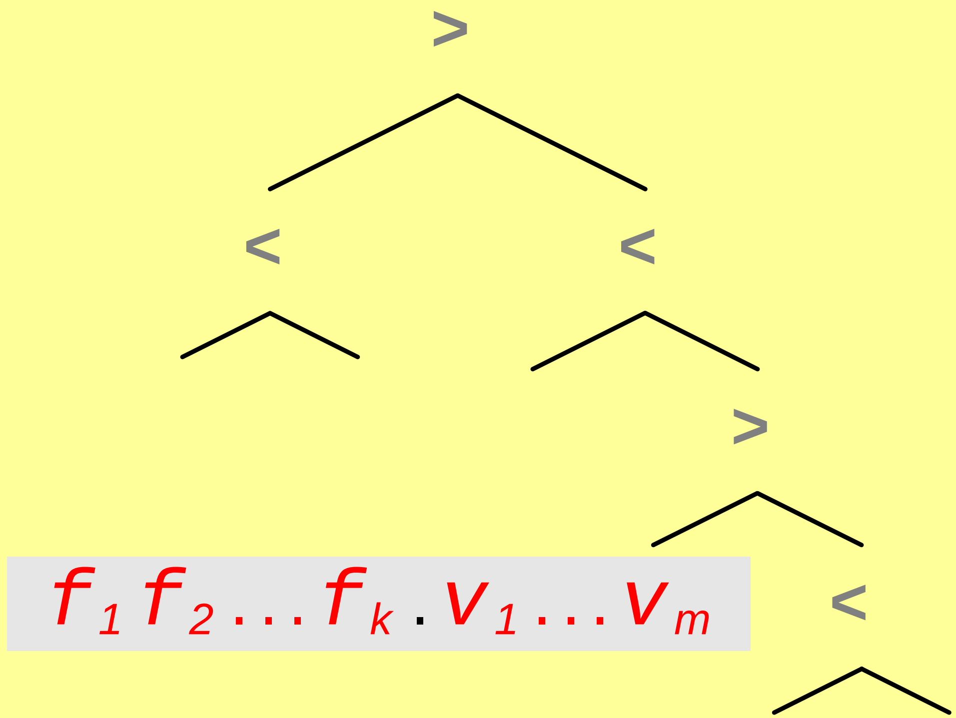
finite, binary labeled trees such that . . .

- non-leaf-labels are from { < , > } [“projection”]

Minimalist expressions

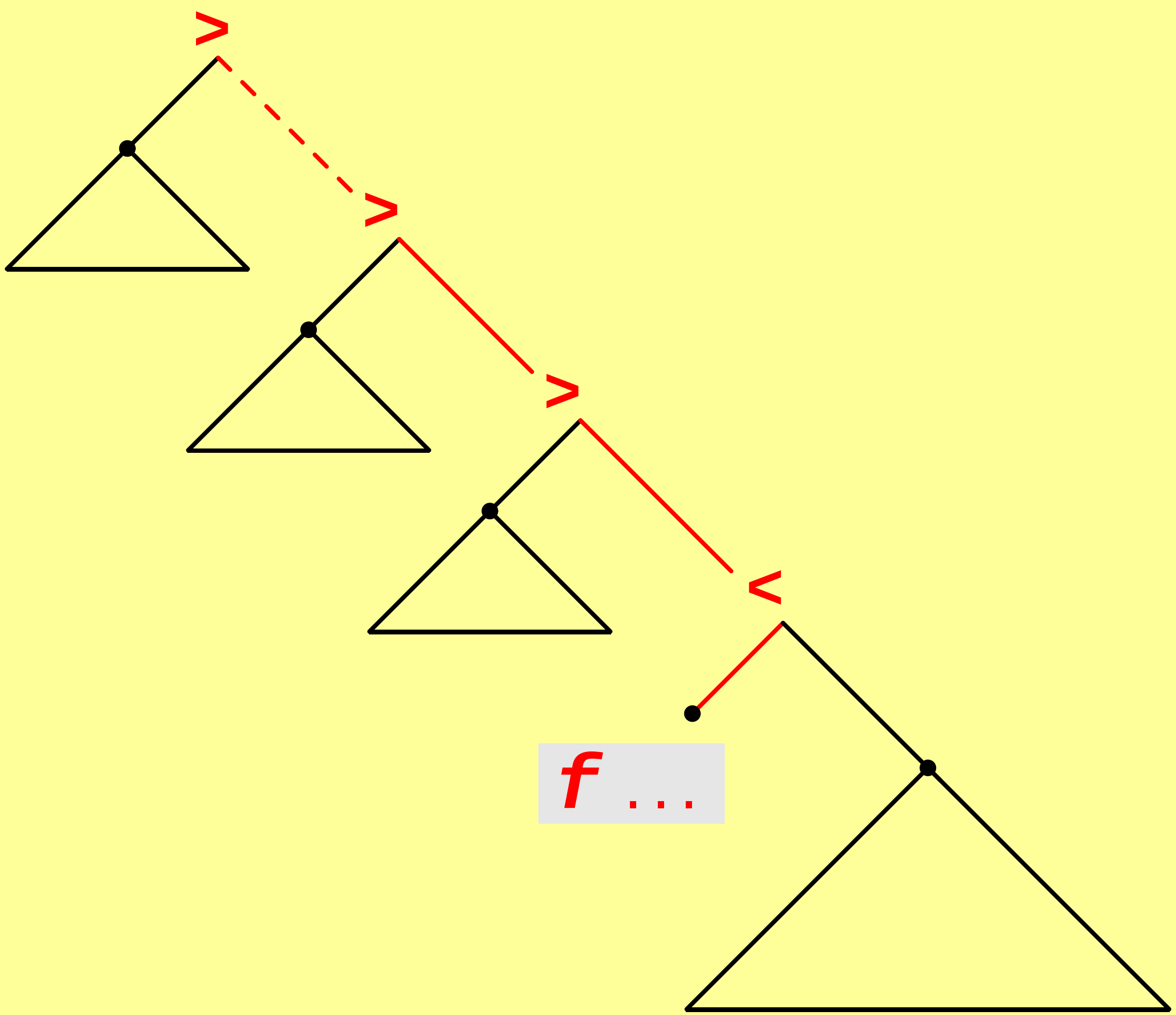
Vocabulary (terminals)

SynFeatures (syntactic features)



finite, binary labeled trees such that . . .

- leaf-labels are from SynFeatures*.Vocabulary*



tree displays feature *f* $:\Longleftrightarrow$ head-label is of the form *f* . . .

- There are different types of syntactic features.

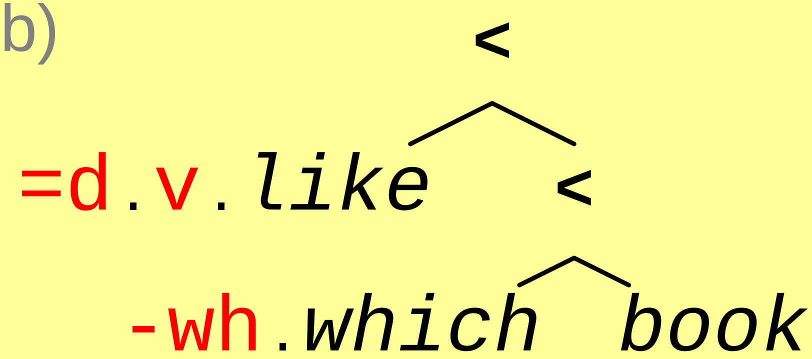
(basic) categories:	x , y , z , . . .	[Base]
(merge-) selectors:	=x , =y , =z , . . .	[Selectors]
(move-) licensees:	-x , -y , -z , . . .	[Licensees]
(move-) licensors:	+<u>x</u> , +<u>y</u> , +<u>z</u> , . . .	[Licensors]

. . .

Minimalist expresssions (examples)

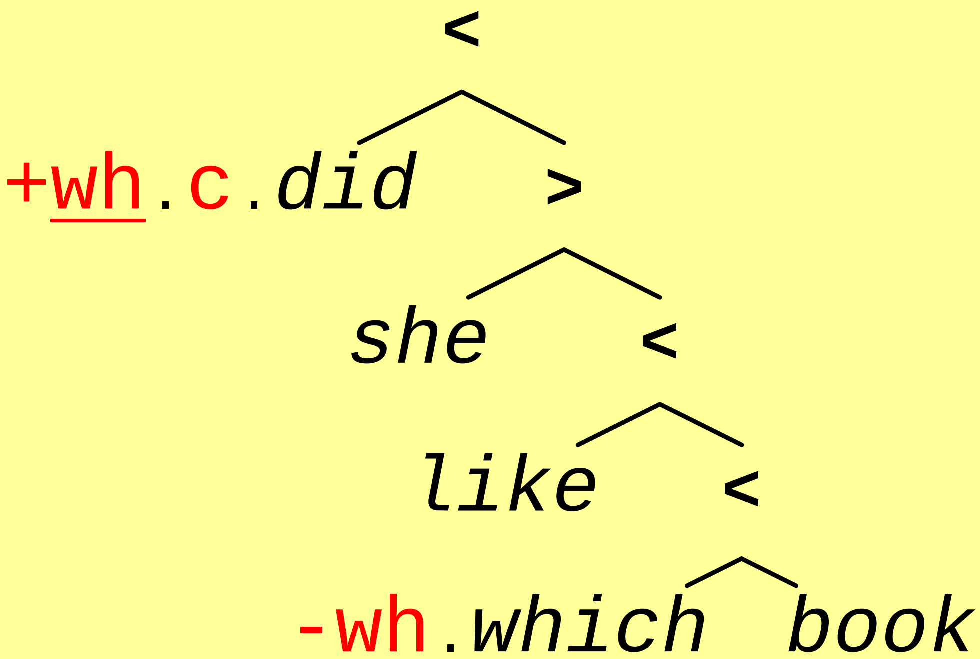
(a) =d.=d.v.*like*

(b)



(c) d.*she*

(d)



Building minimalist expressions

- Starting from a finite set of simple expressions (a lexicon), minimalist expressions can be built up recursively
  - by applying structure building functions checking off instances of syntactic features "from left to right,"

    where, after having applied a structure building function, the triggering feature instances are canceled.

- Different types of syntactic features trigger different structure building functions.

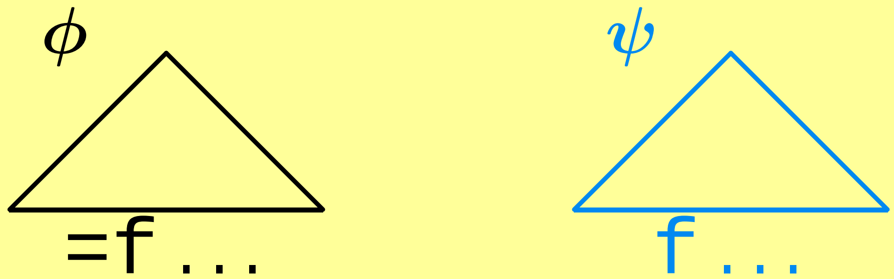
Structure building functions

$$\text{merge} : \text{Trees} \times \text{Trees} \xrightarrow{\text{part}} \text{Trees}$$

- $\langle \phi , \psi \rangle \in$ Domain(merge) $:\Longleftrightarrow$
  - ψ displays feature `f` $\in$ Base
  - ϕ displays feature `=f` $\in$ Selectors

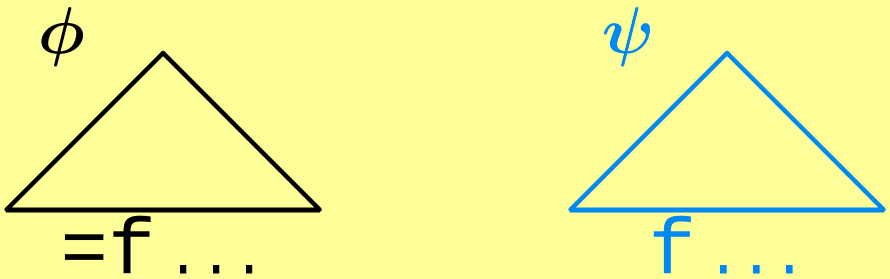
Structure building functions

$$\text{merge} : \text{Trees} \times \text{Trees} \xrightarrow{\text{part}} \text{Trees}$$



Structure building functions

$$\mathsf{merge} : \mathsf{Trees} \times \mathsf{Trees} \xrightarrow{\mathsf{part}} \mathsf{Trees}$$

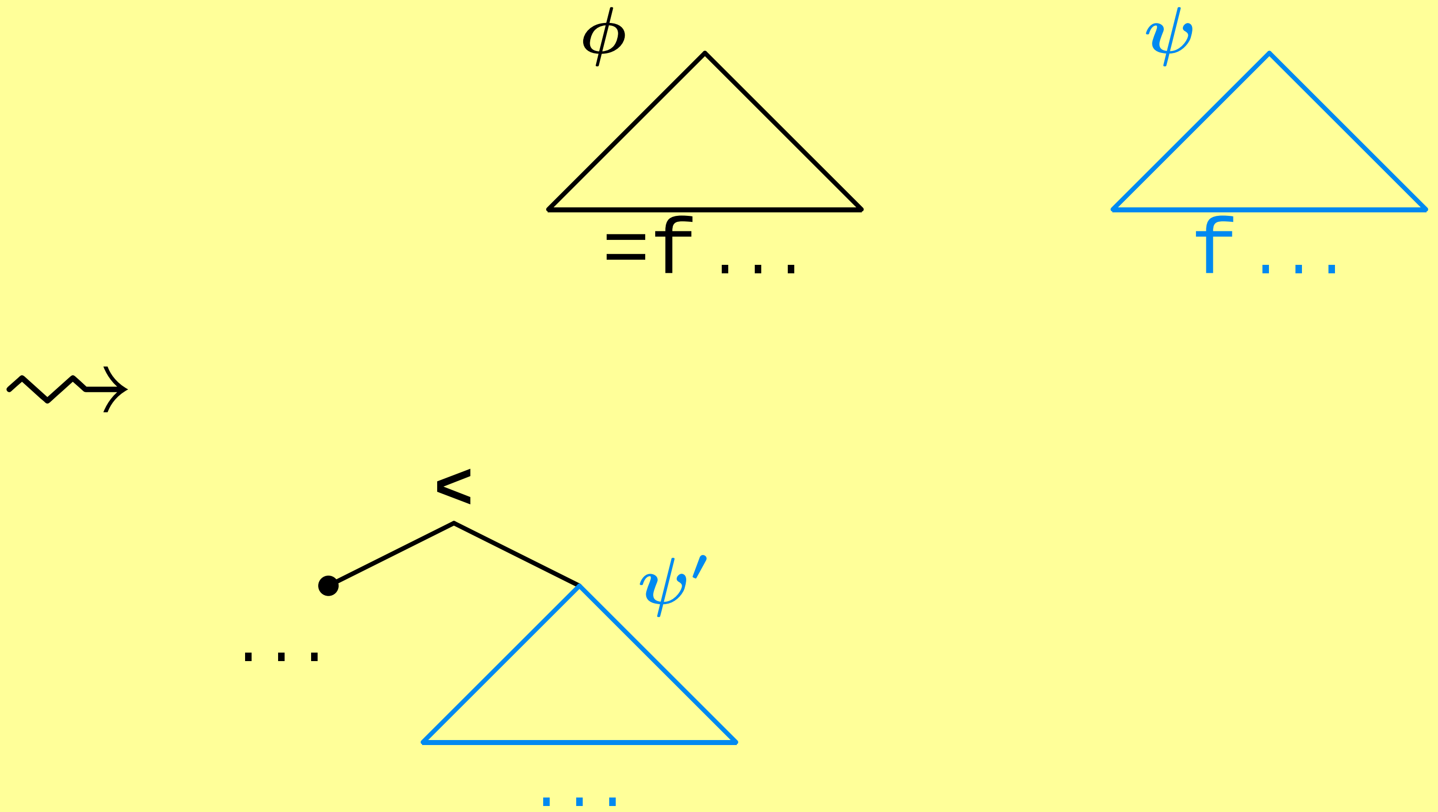


$\rightsquigarrow$

selecting ϕ simple

selecting ϕ complex

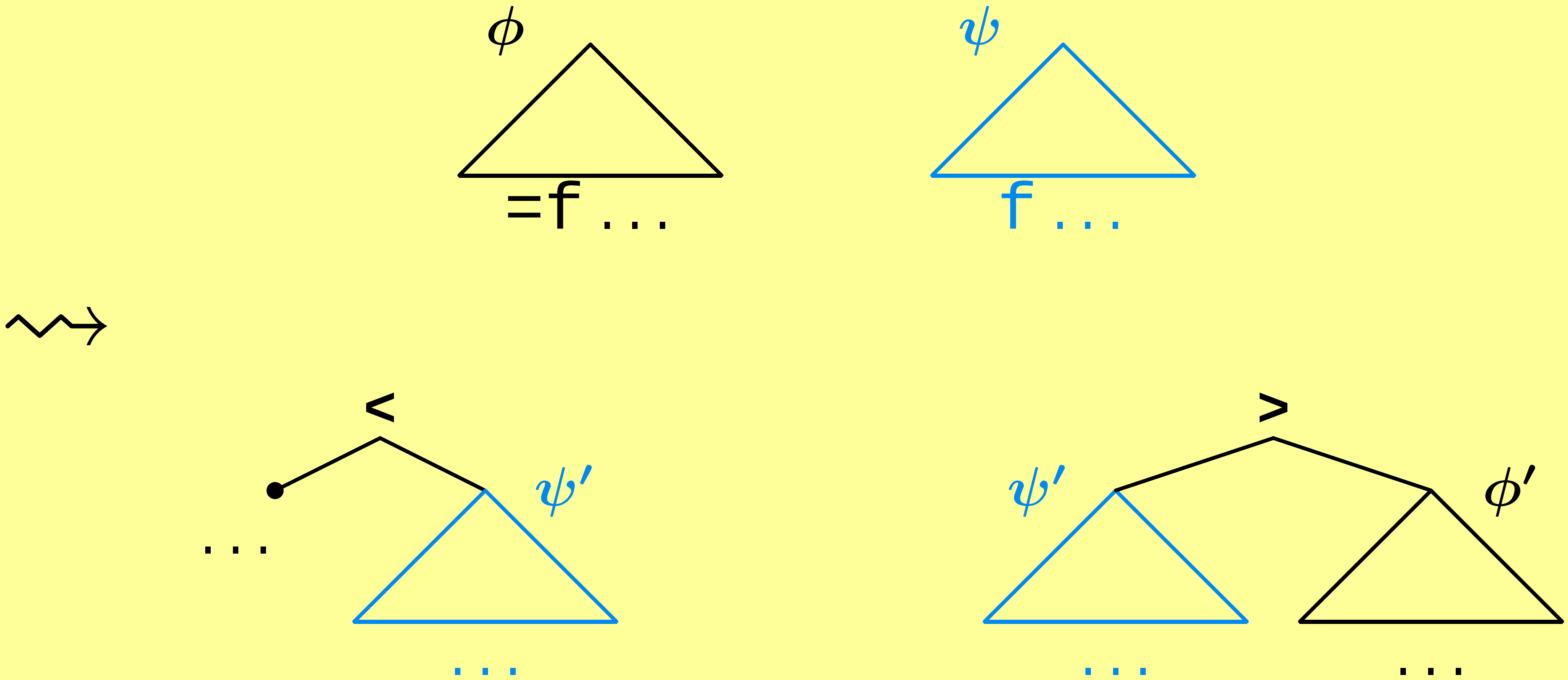
merge : Trees × Trees $\xrightarrow{\text{part}}$ Trees



selecting φ simple

selecting φ complex

$$\text{merge} : \text{Trees} \times \text{Trees} \xrightarrow{\text{part}} \text{Trees}$$

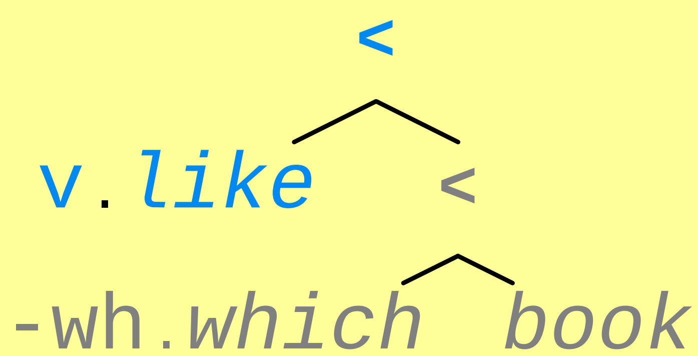


selecting ϕ simple

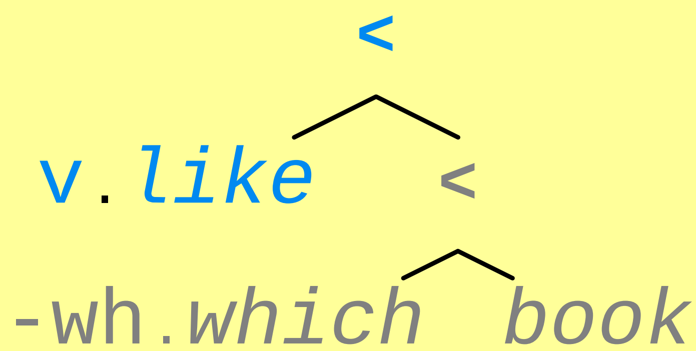
selecting ϕ complex

merge (selecting tree is simple)

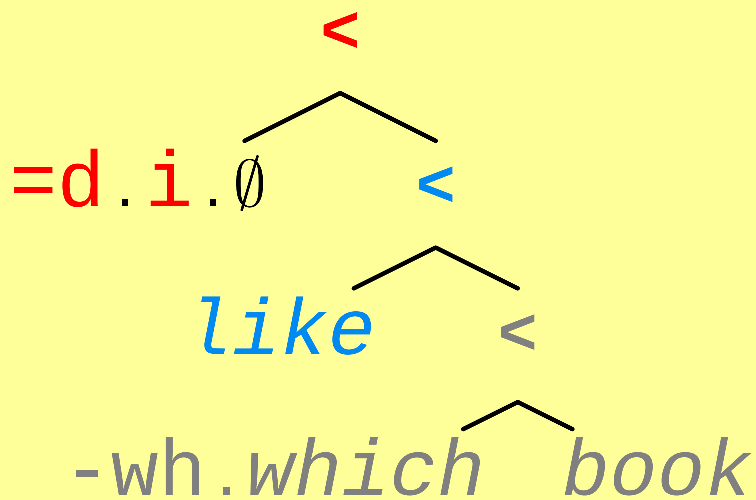
=v.=d.i.∅ +



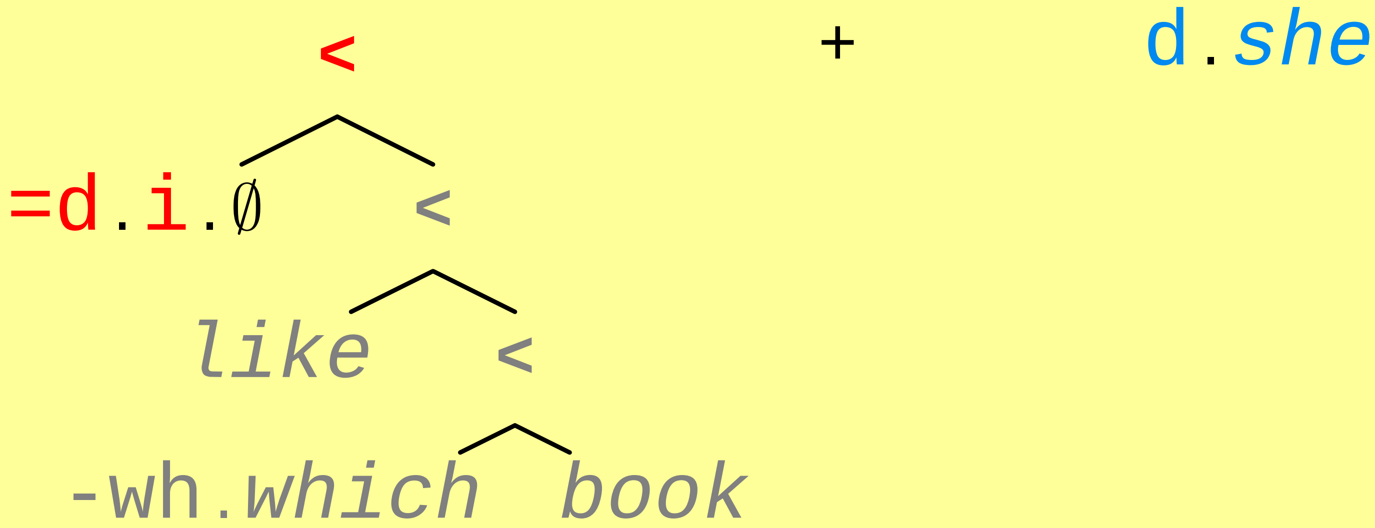
=v.=d.i.∅ +

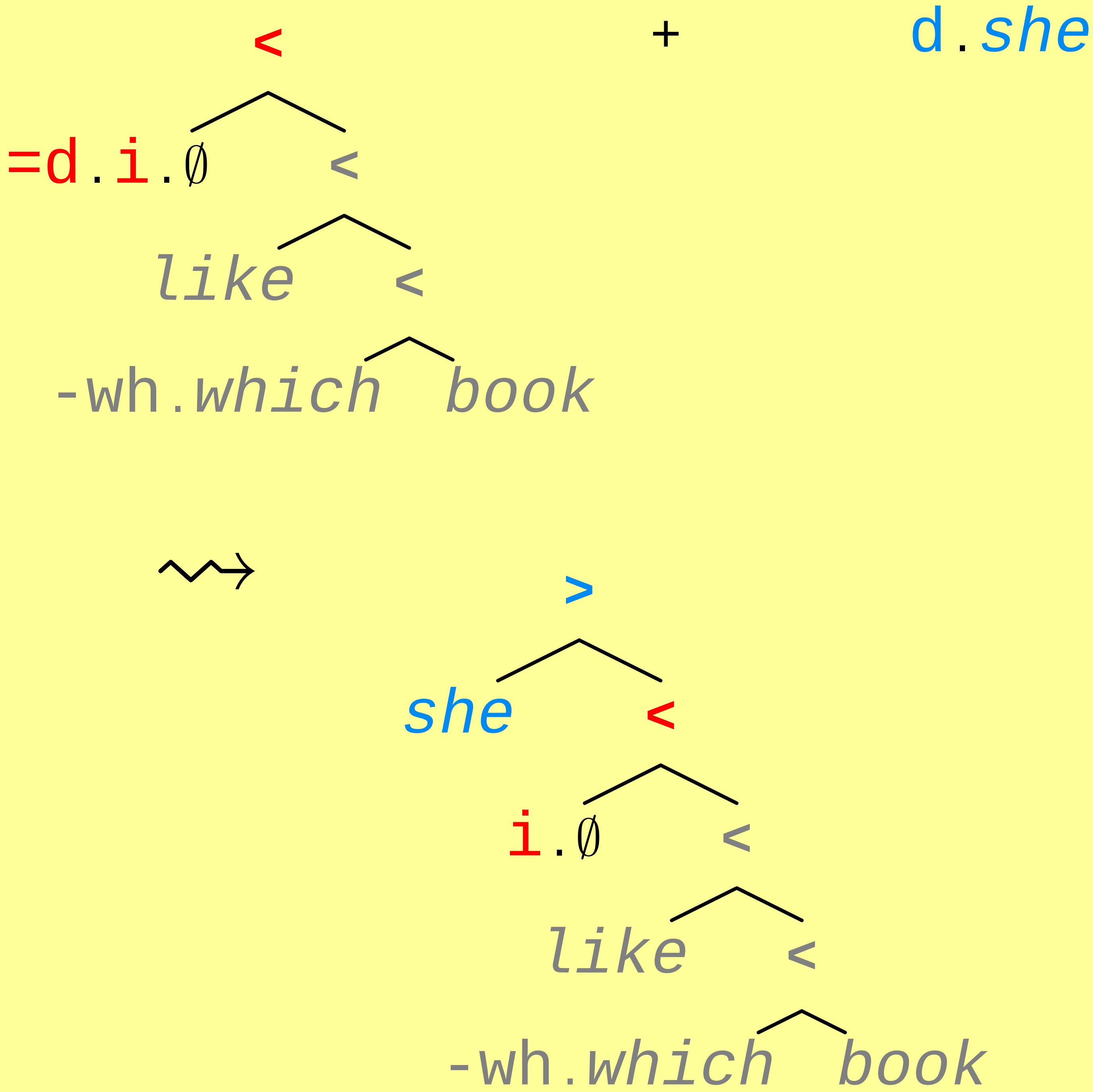


⇝



merge (selecting tree is complex)




<
+
d.she
=d.i.∅
<
like
<
-wh.which
book
⇝
>
she
<
i.∅
<
like
<
-wh.which
book

Structure building functions (overt phrasal movement)

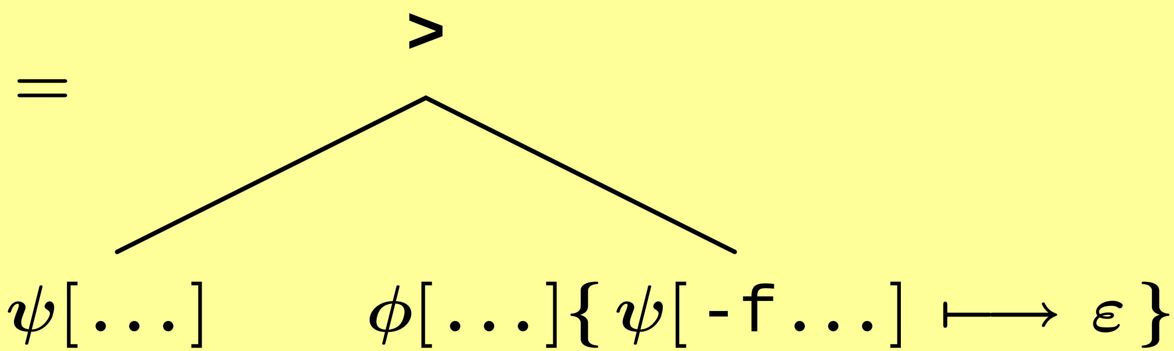
$\text{move} : \text{Trees} \xrightarrow{\text{part}} 2^{\text{Trees}}$

- $\phi \in \text{Domain(move)} :\Longleftrightarrow$

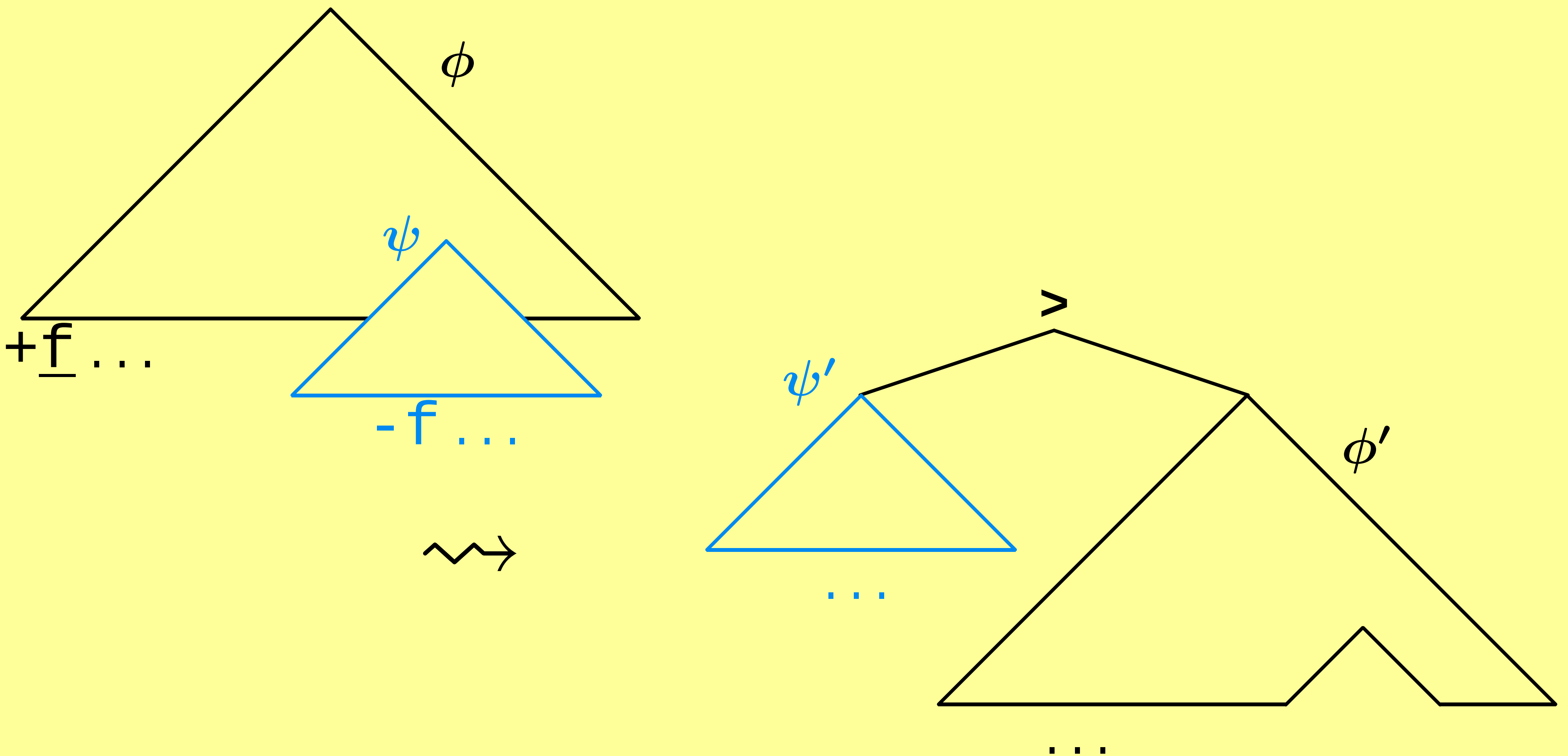
  - ϕ displays feature $+\underline{\texttt{f}} \in$ Licensors

  - there is a maximal projection ψ within ϕ that displays feature $\texttt{-f} \in$ Licensees

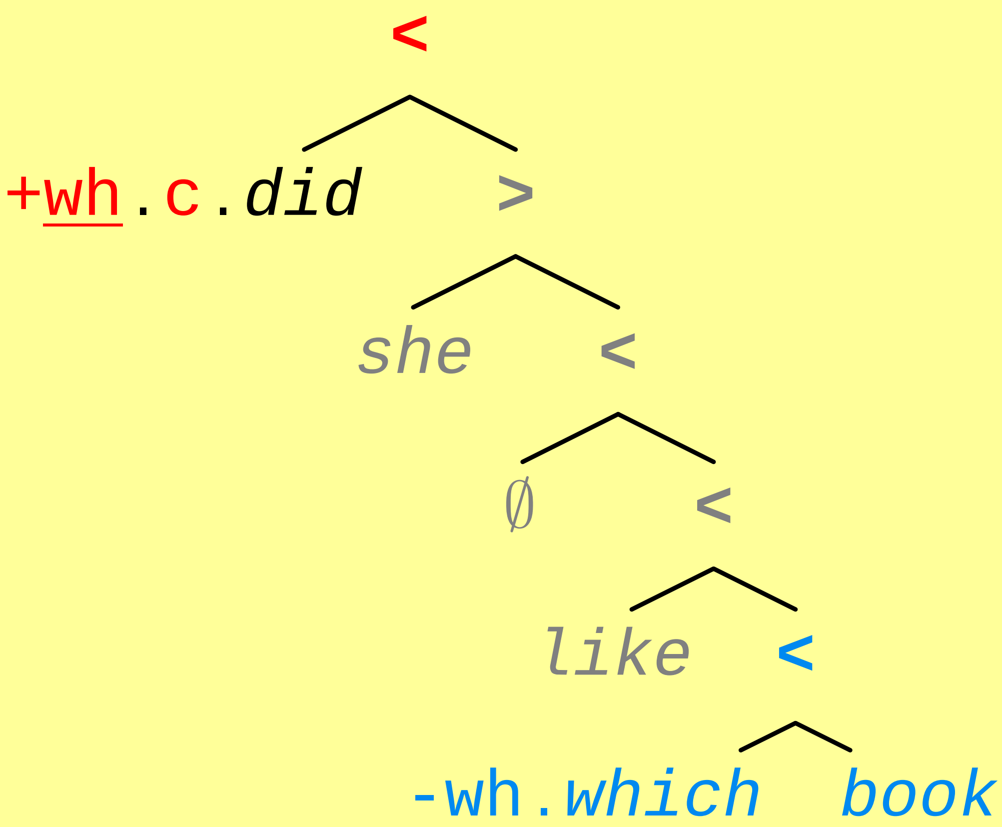
- $\text{move}(\phi[+\underline{\texttt{f}}\ldots]) \quad =$



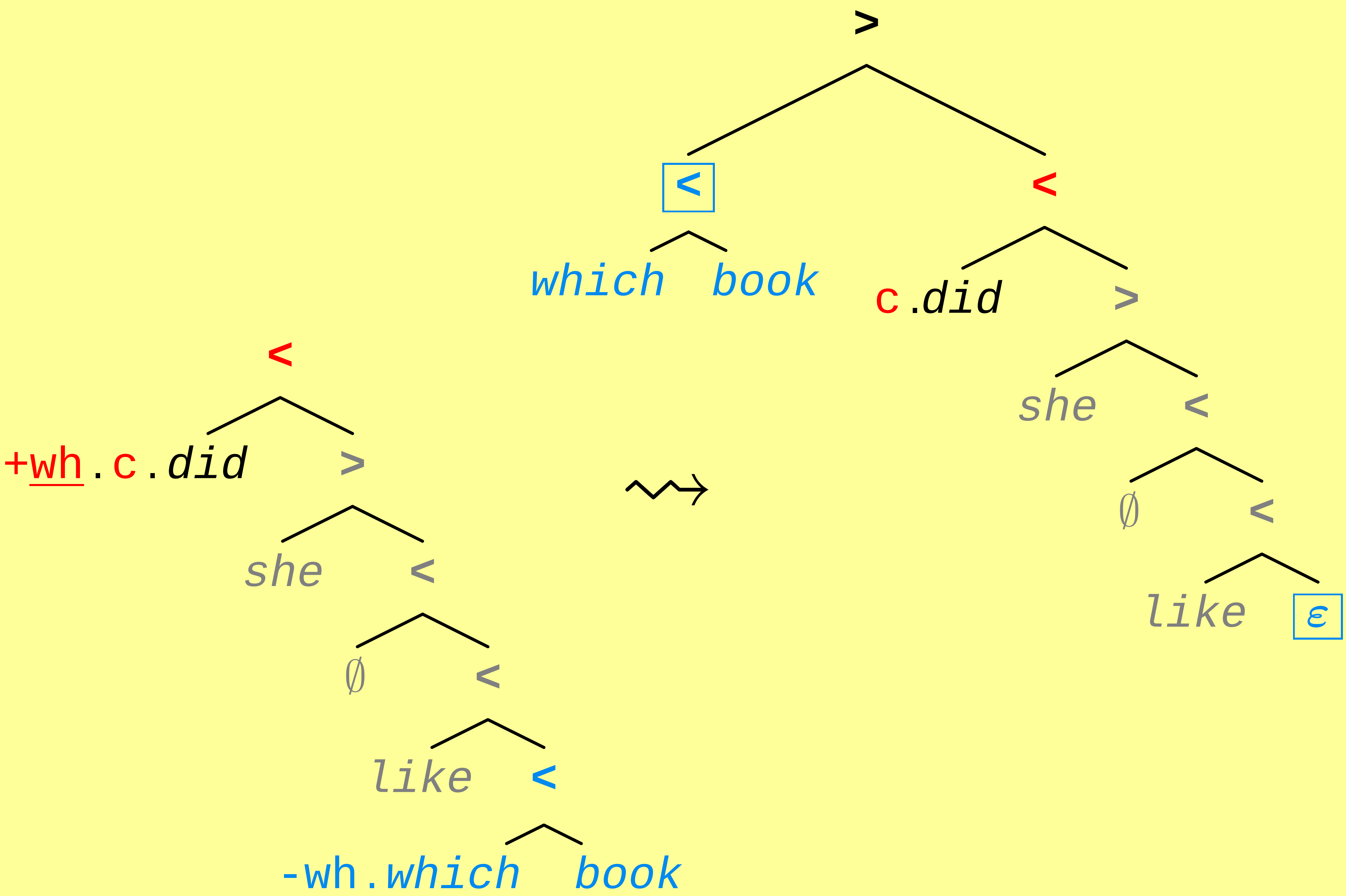
$$\text{move} : \text{Trees} \xrightarrow{\text{part}} 2^{\text{Trees}}$$



move



move



- There are different types of syntactic features.

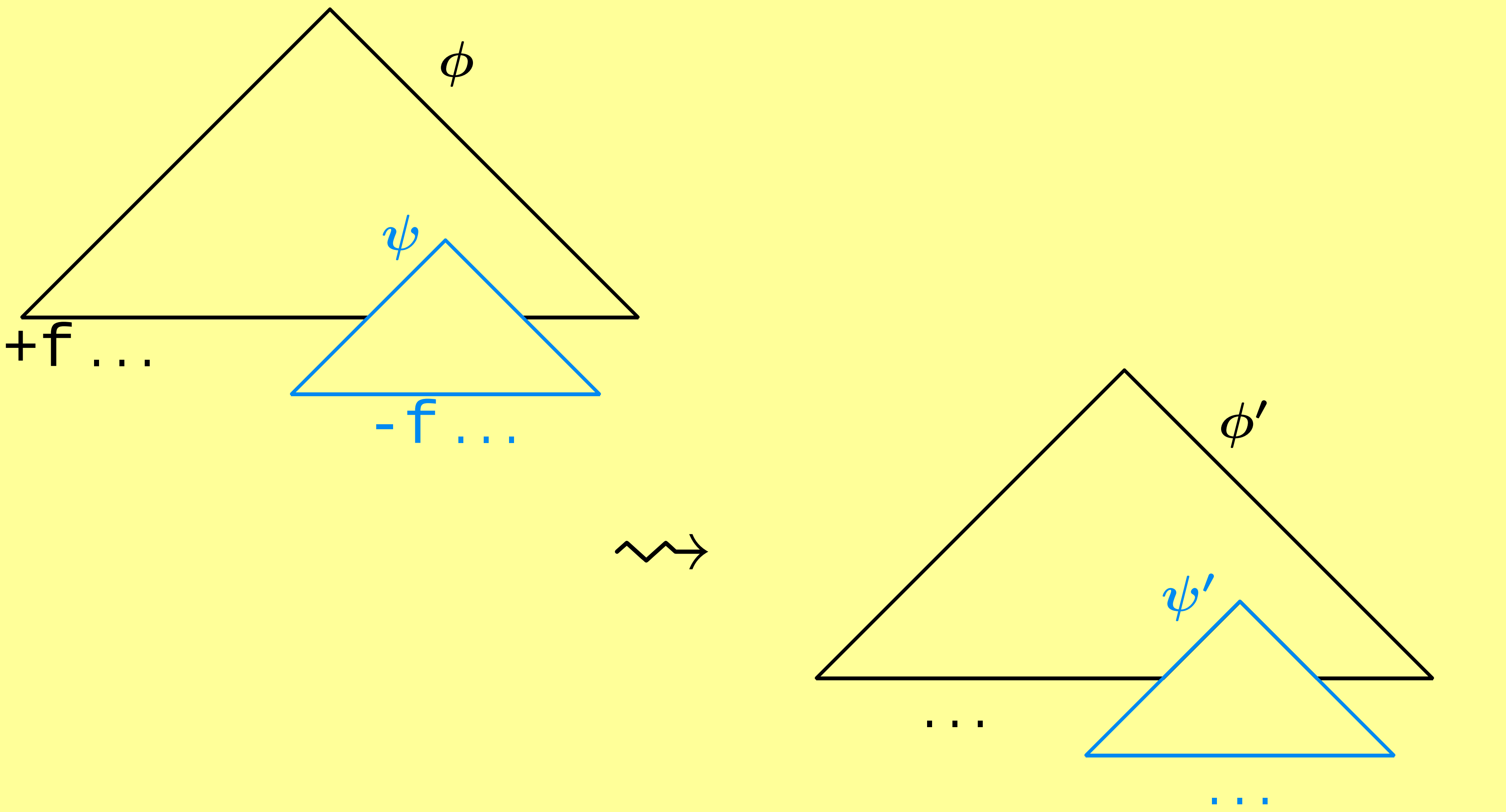
(basic) categories:	x , y , z , . . .	[Base]
(merge-) selectors:	. . .	[Selectors]
(move-) licensees:	-x , -y , -z , . . .	[Licensees]
(move-) licensors:	+<u>x</u> , +<u>y</u> , +<u>z</u> , . . .	[Licensors , strong]
	+x , +y , +z , . . .	[Licensors , weak]

. . .

$\text{agree} : \text{Trees} \xrightarrow{\text{part}} 2^{\text{Trees}}$

- $\phi \in \text{Domain(agree)} :\Longleftrightarrow$
  - ϕ displays feature $\texttt{+f} \in \text{Licensors}$
  - there is a maximal projection ψ within ϕ that displays feature $\texttt{-f} \in \text{Licensees}$
- $\text{agree}(\phi[\texttt{+f}\ldots]) = \phi[\ldots]\{\psi[\texttt{-f}\ldots] \longmapsto \psi[\ldots]\}$

agree : Trees $\xrightarrow{\text{part}}$ 2^{Trees}



- There are different types of syntactic features.

(basic) categories:	x , y , z , . . .	[Base]
(merge-) selectors:	=x , =y , =z , . . .	[Selectors , weak]
	=>x , =>y , =>z , . . .	[Selectors , strong]
	x<= , y<= , z<= , . . .	
(move-) licensees:	-x , -y , -z , . . .	[Licensees]
(move-) licensors:	+x , +y , +z , . . .	[Licensors]

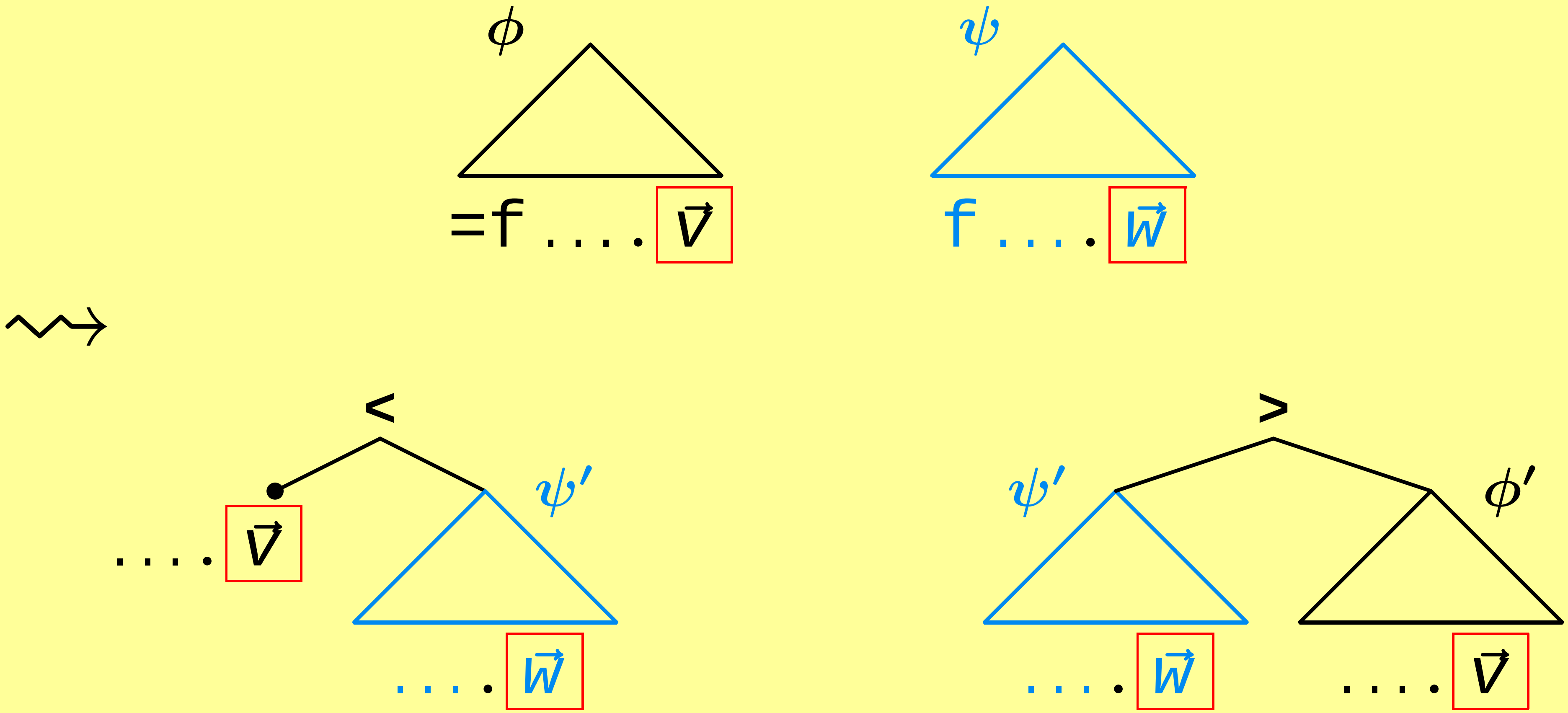
. . .

Structure building functions

$$\text{merge} : \text{Trees} \times \text{Trees} \xrightarrow{\text{part}} \text{Trees}$$

- $\langle\phi,\psi\rangle \in$ Domain(merge) $:\Longleftrightarrow$
  - ψ displays feature `f` $\in$ Base
  - ϕ displays feature `=f`, `=>f`, or `f<=` $\in$ Selectors

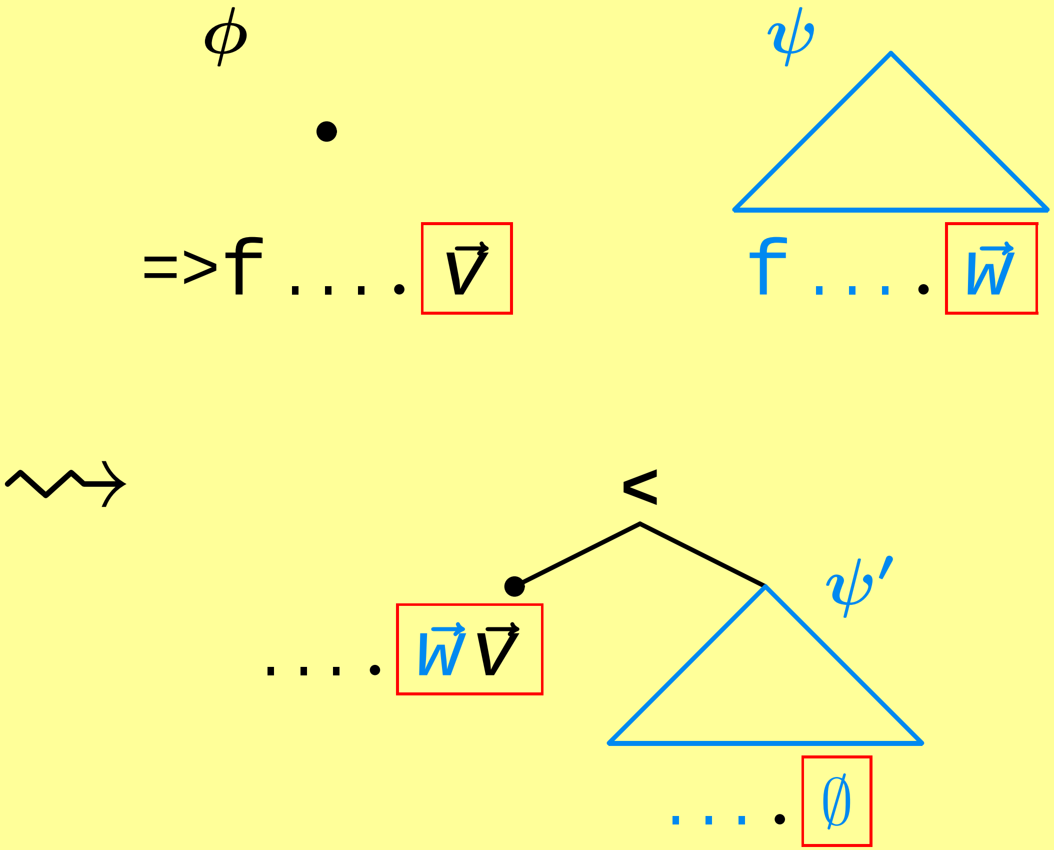
merge : Trees $\times$ Trees $\xrightarrow{\text{part}}$ Trees



selecting ϕ simple

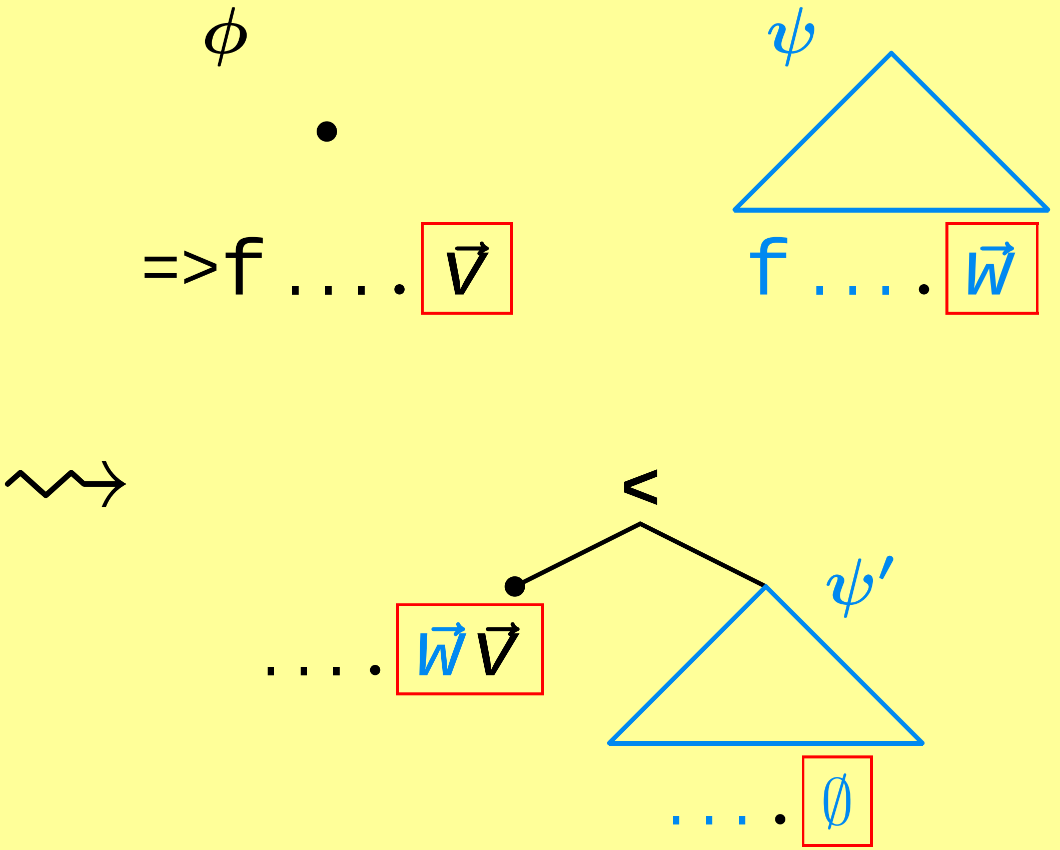
selecting ϕ complex

$\text{merge} : \text{Trees} \times \text{Trees} \xrightarrow{\text{part}} \text{Trees}$



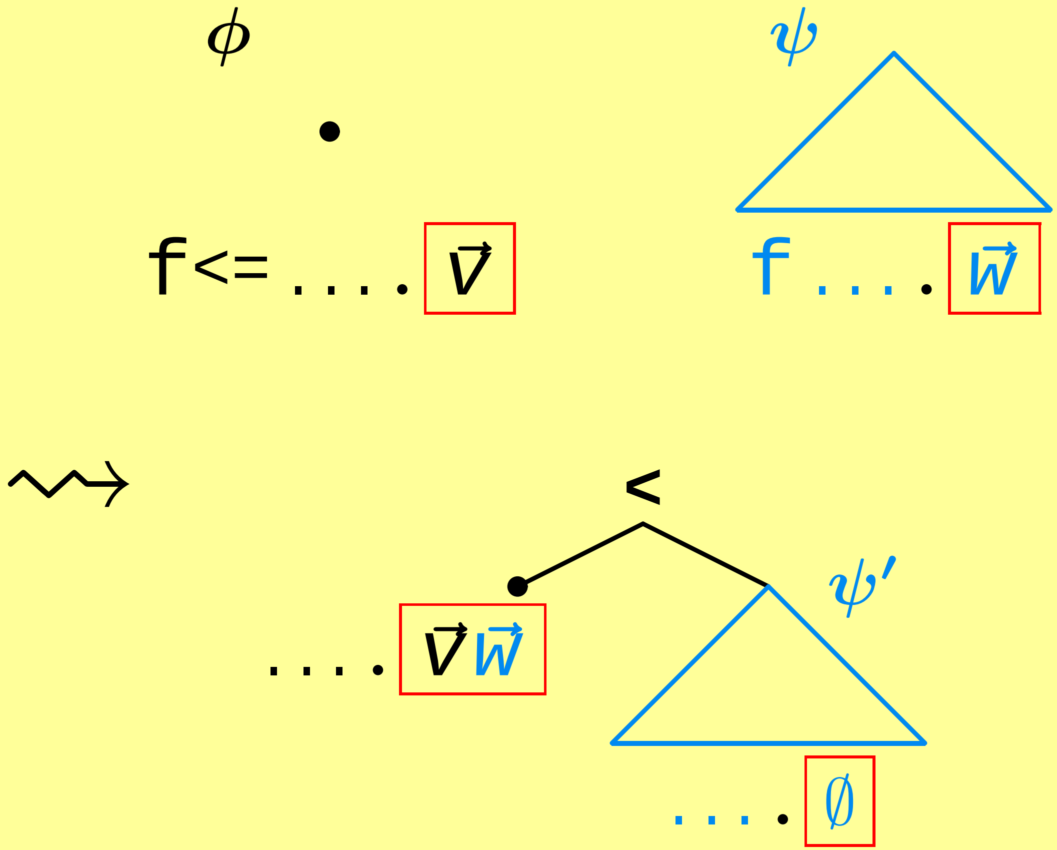
selecting ϕ simple , head-incorporation left

$$\text{merge} : \text{Trees} \times \text{Trees} \xrightarrow{\text{part}} \text{Trees} \qquad \text{(HMC)}$$

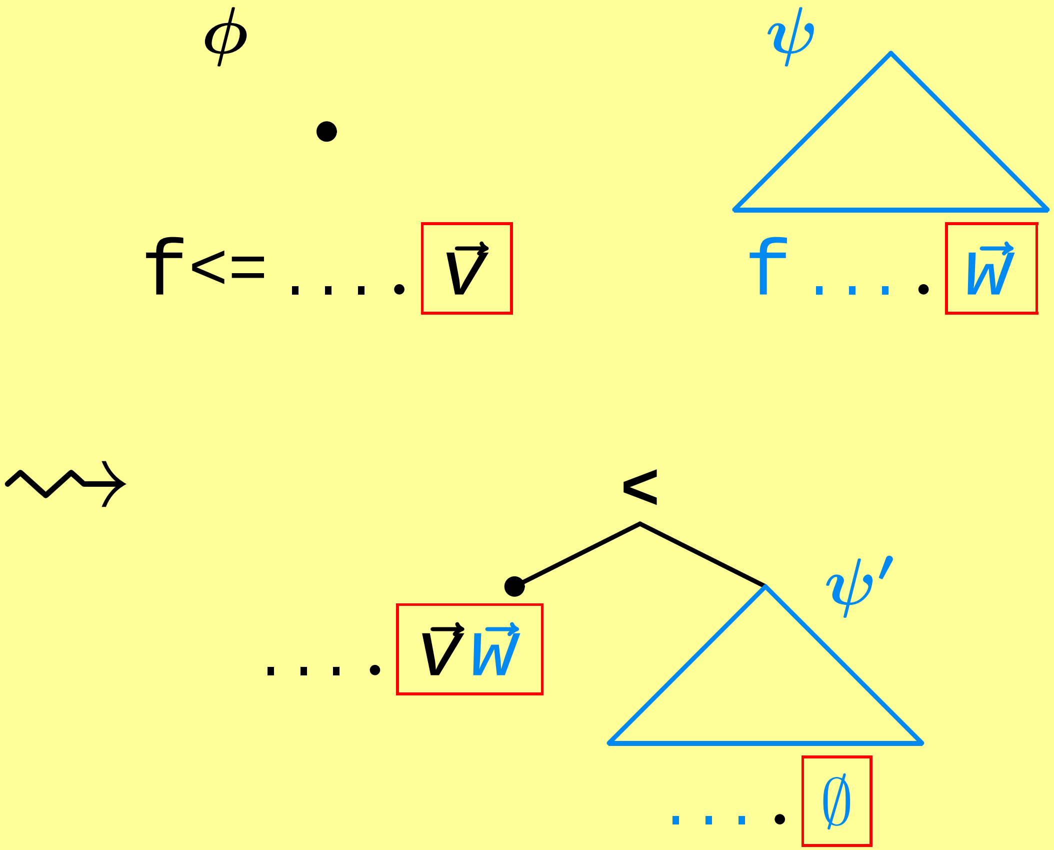


selecting ϕ simple , head-incorporation left

merge : Trees × Trees $\xrightarrow{\text{part}}$ Trees

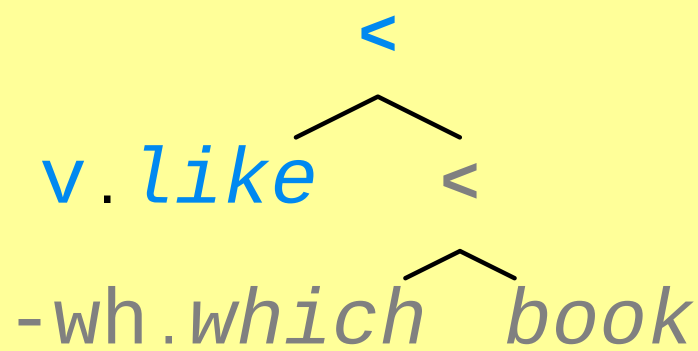


selecting ϕ simple , head-incorporation right merge : Trees × Trees $\xrightarrow{\text{part}}$ Trees (HMC)

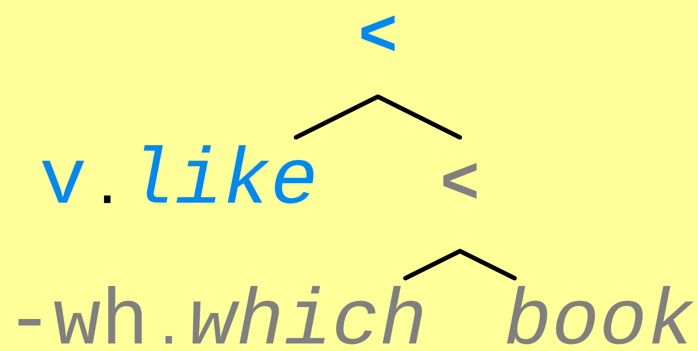


selecting ϕ simple , head-incorporation right

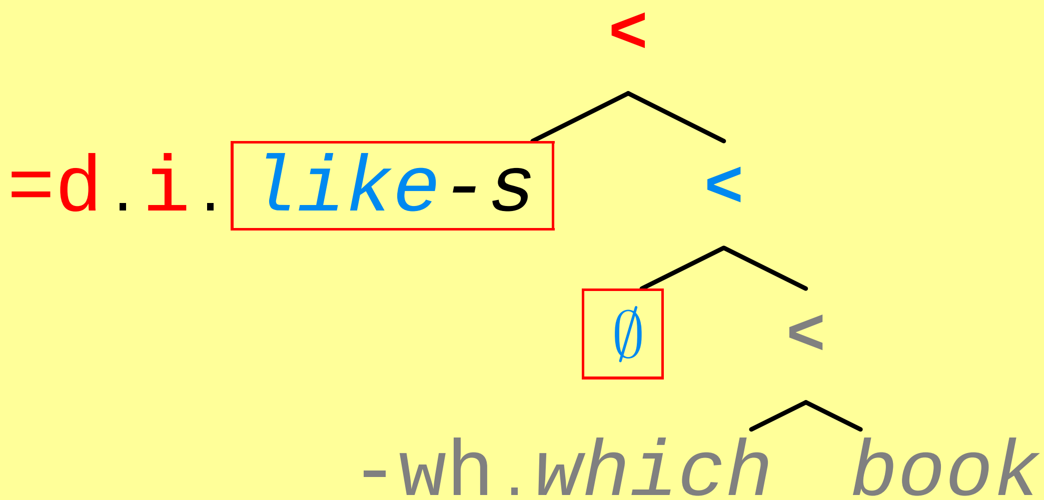
=>v.=d.i.-*s* +



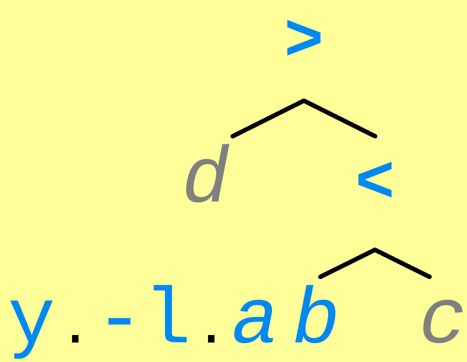
=>v.=d.i.-*s* +

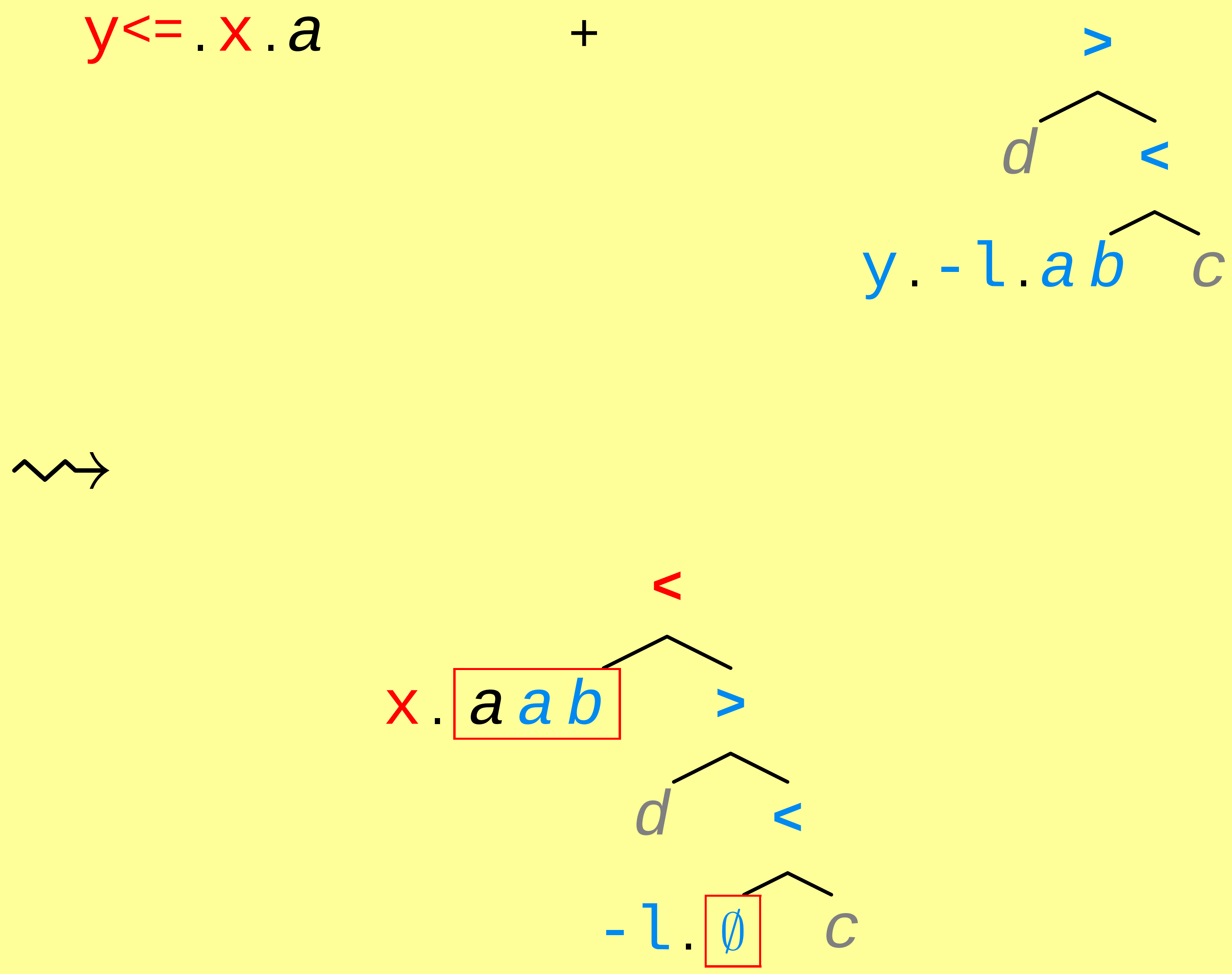


⇝



y<=.x.*a* +




y<=.x.a
+
>
d
<
y.-l.ab
c
⇝
<
x.aab
>
d
<
-l.∅
c

$G = \langle$ Features , Lexicon , Ω , c $\rangle$

- Features = SynFeatures ∪ Vocabulary [features]

  SynFeatures = Base ∪ Selectors ∪ Licensees ∪ Licensors

  x   =x , =>x , x<=   -x   +$\underline{x}$, +x

- Lexicon   a finite set of simple expressions [lexicon]

- Ω = { merge , move , agree } [structure building functions]

- c ∈ Base [distinguished category]

The closure of G [Closure(G)] $:\Longleftrightarrow$

closure of the lexicon under finite applications of the functions in Ω.

The tree language of G [T(G)] $:\Longleftrightarrow$

trees in the closure with essentially no unchecked syntactic features — only head-label contains exactly one unchecked instance of c.

The string language of G [L(G)] $:\Longleftrightarrow$

(terminal) yields of the trees belonging to the tree language.

A simple MG-lexicon

n.*book*	d.*she*
=d.v.*like*	=n.d.-wh.*which*
=v.=d.i.$\emptyset$	=i.+wh.c.*did*

Vocabulary $= \{$*book*, *did*, *like*, *she*, *which*$\}$

Head movement constraint (HMC) (Stabler 1997)

- The implementation of

  head movement in MGs is in accordance with the HMC

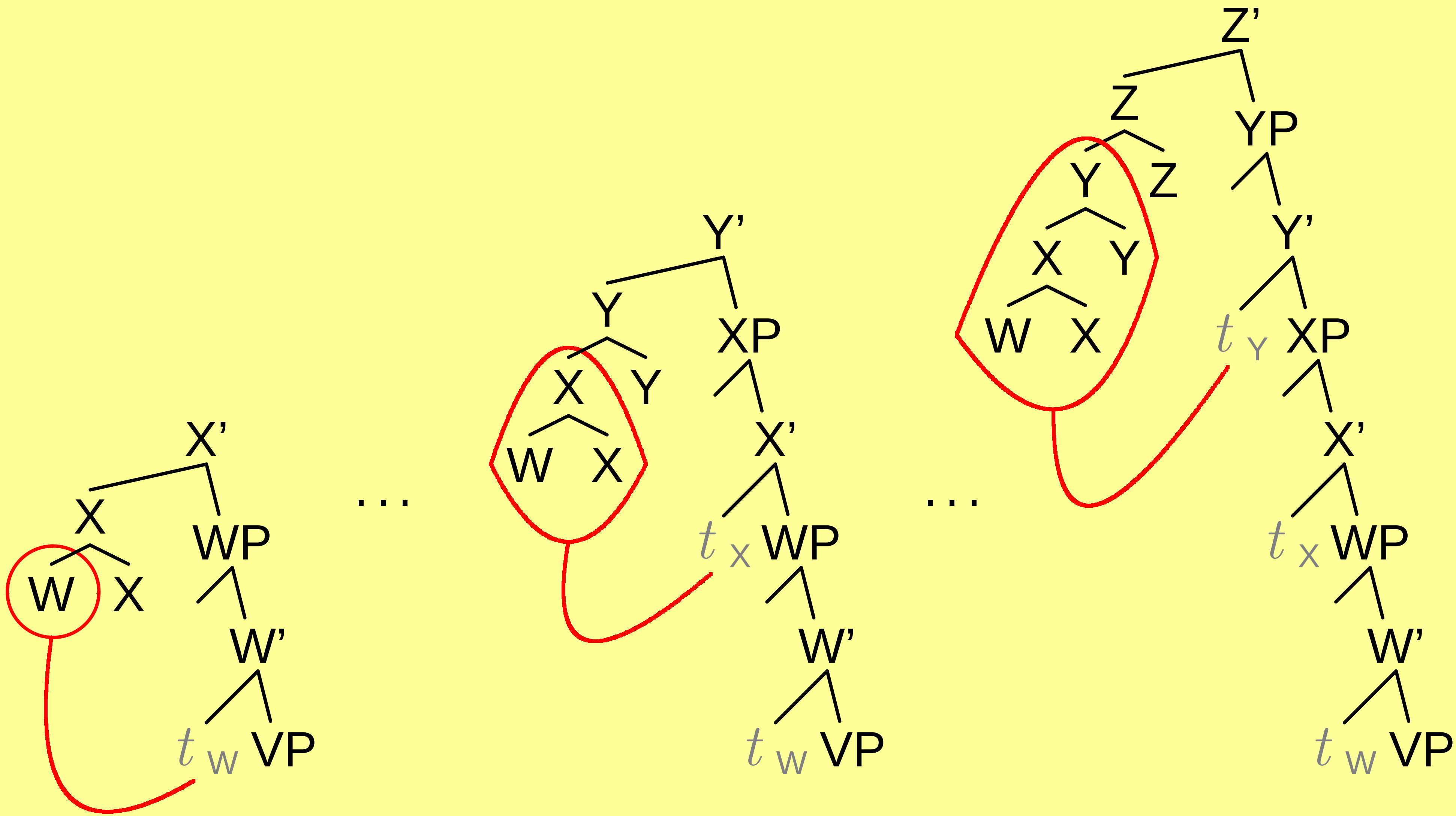
  – demanding

    a moving head not to pass over the closest c-commanding head.

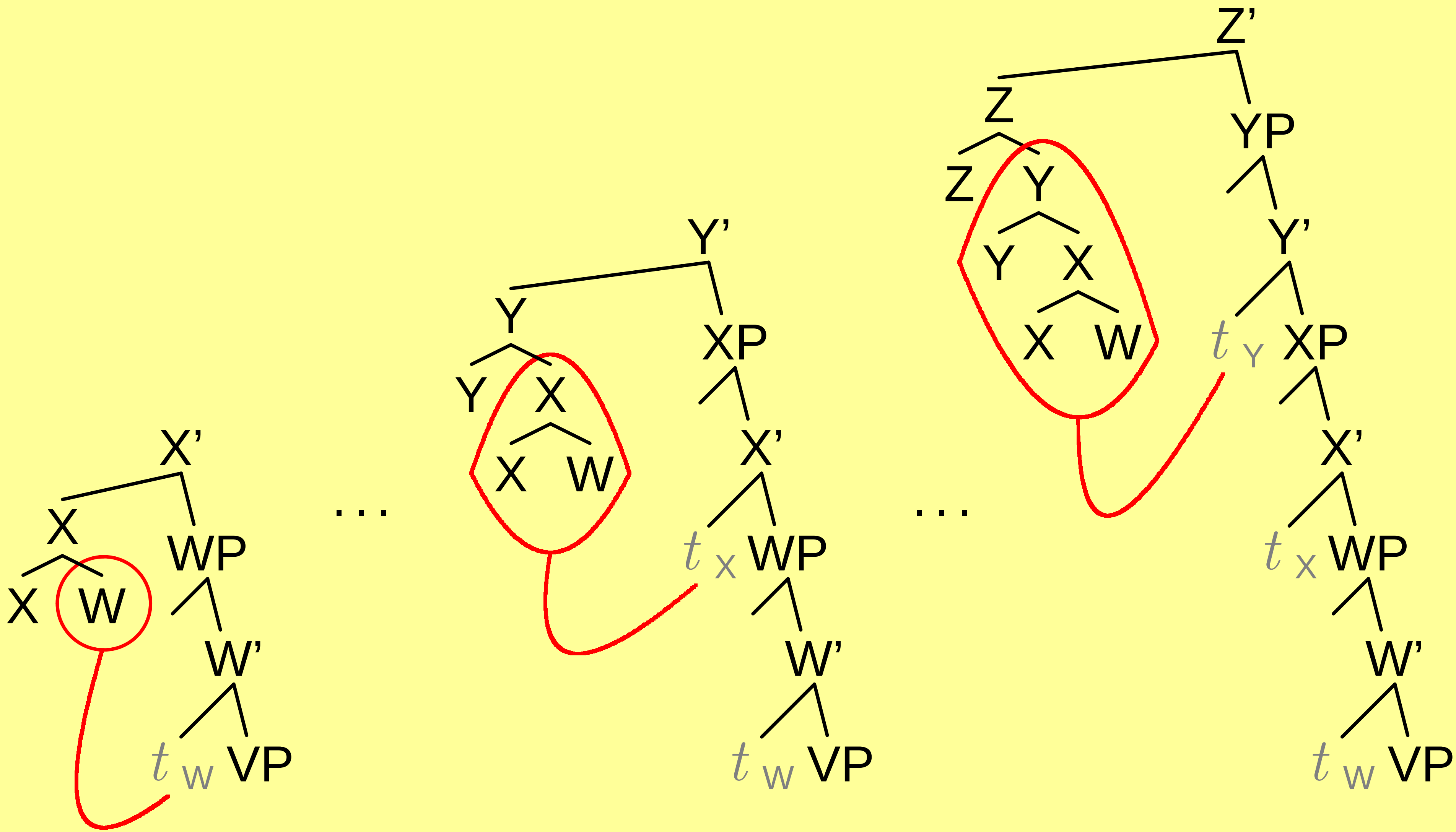
To put it differently,

whenever we are concerned with a case of successive head movement, i.e. recursive adjunction of a (complex) head to a higher head, it obeys strict cyclicity.

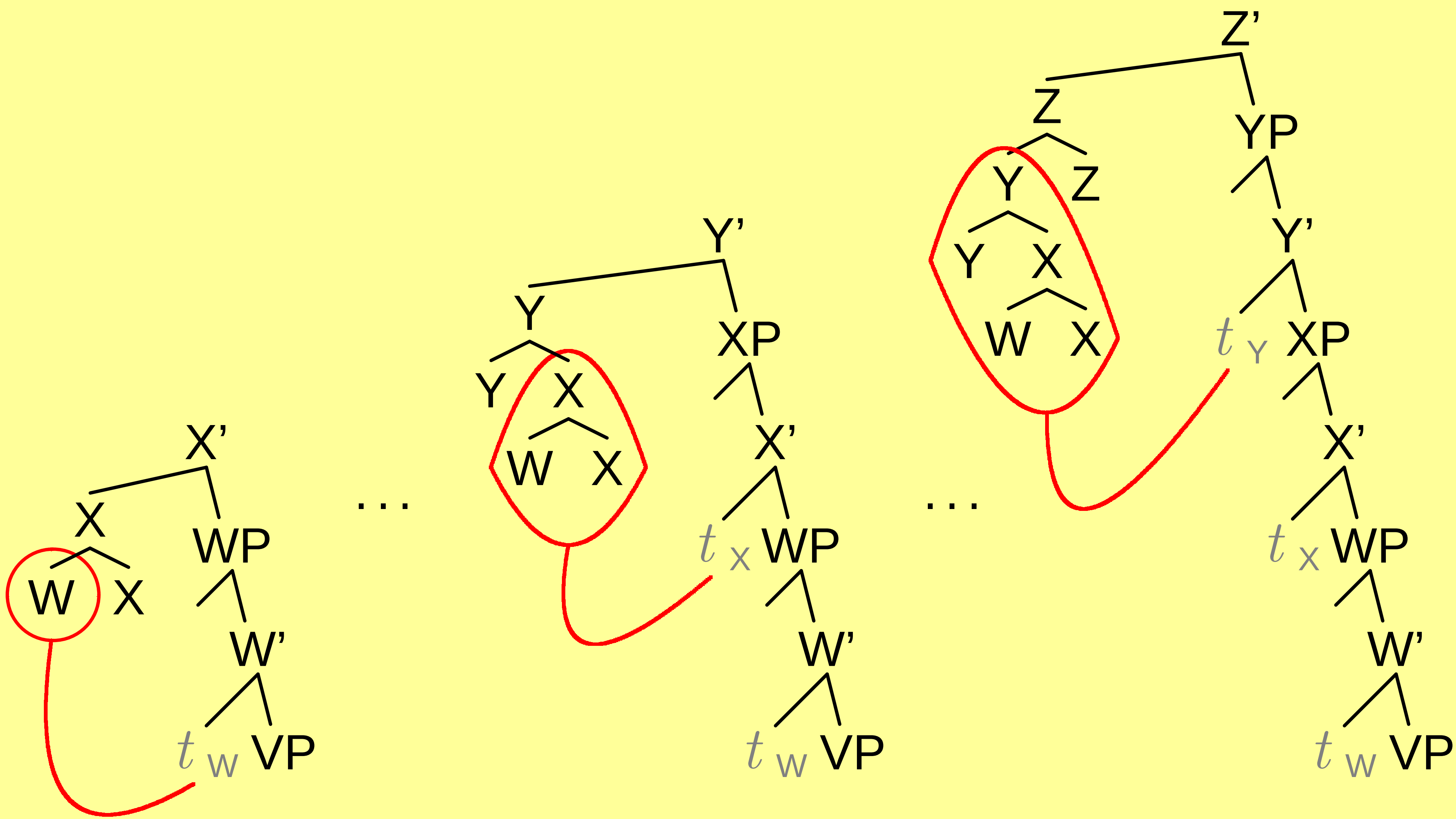
Successive cyclic left head adjunction



Successive cyclic right head adjunction



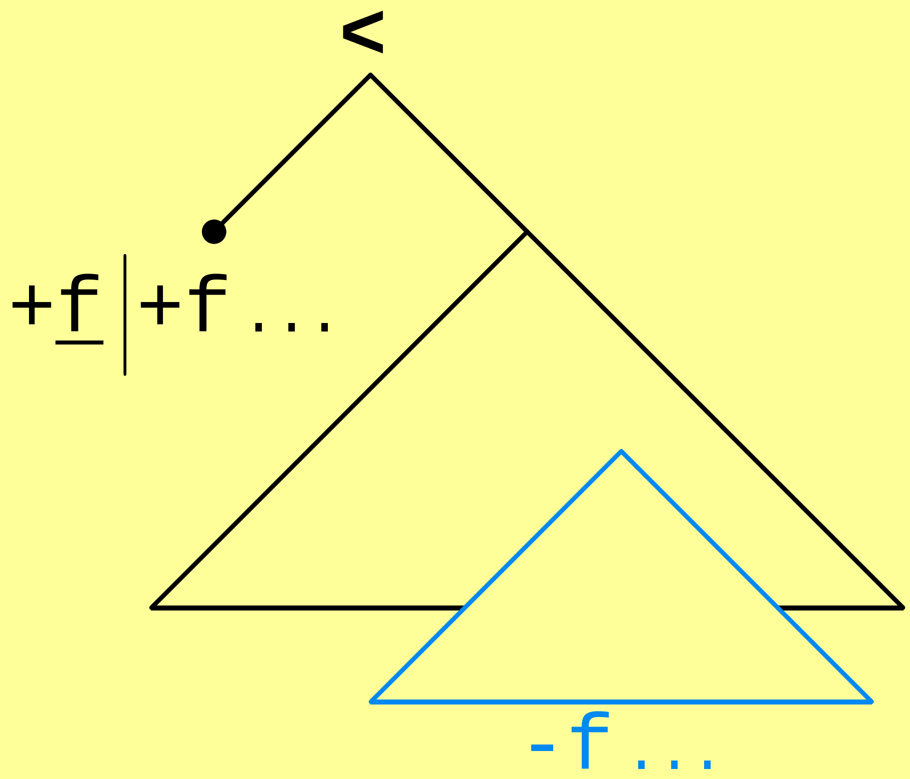
Successive cyclic (mixed) head adjunction



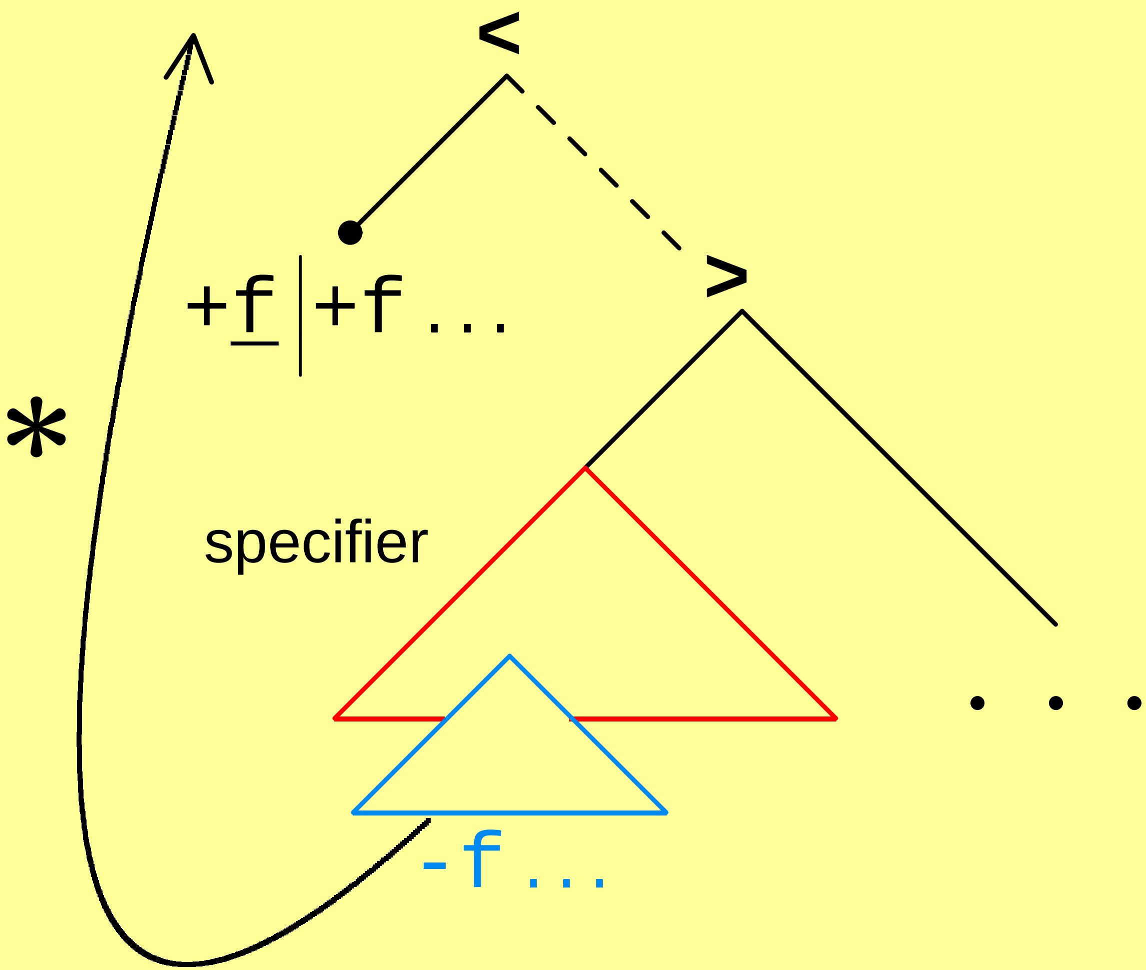
Shortest movement condition (SMC) (Stabler 1997, 1999)

- The number of competing licensee features triggering a movement is (finitely) bounded by n.

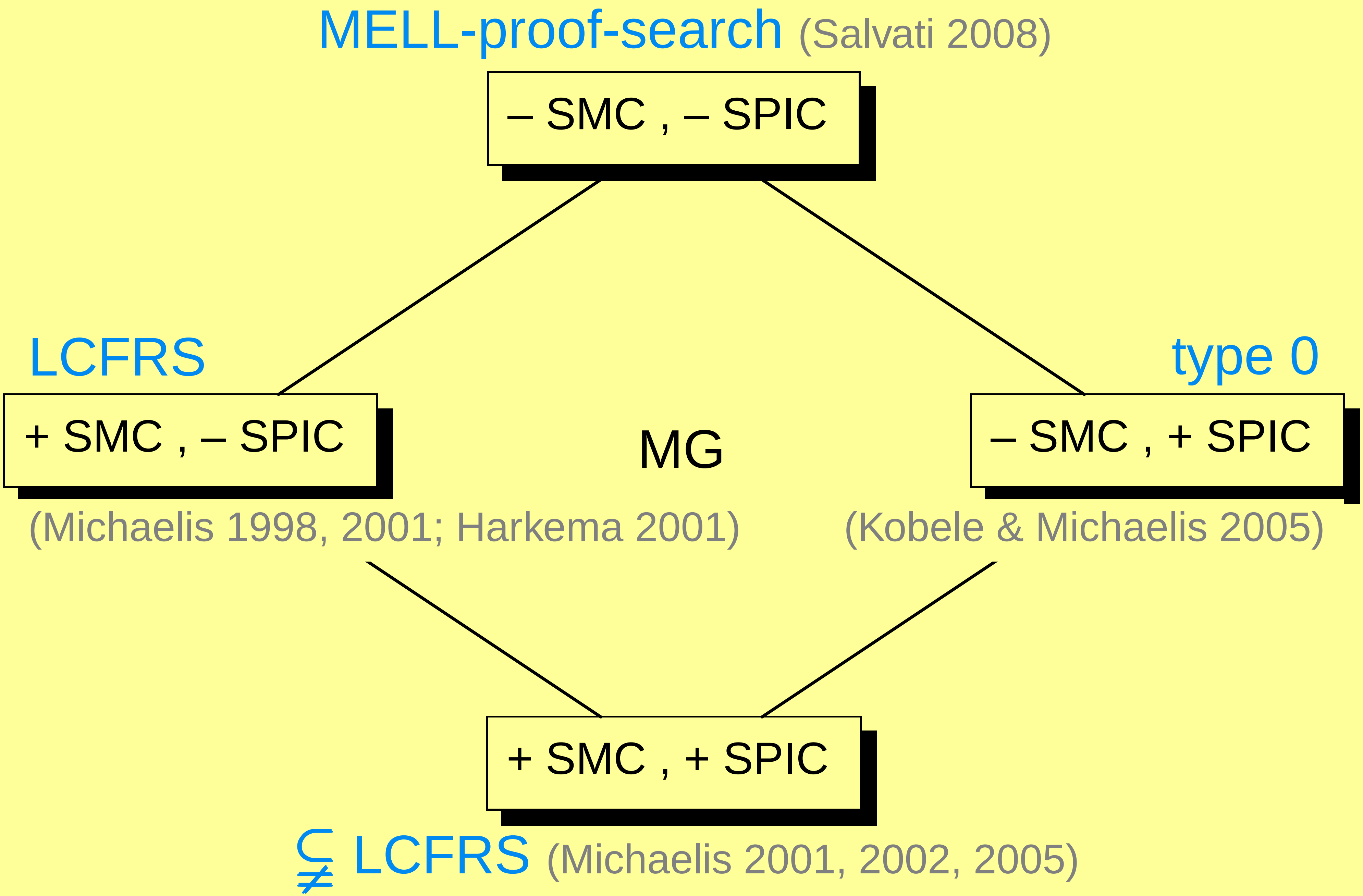
  In the strictest version $n = 1$, i.e., there is at most one maximal projection displaying a matching licensee feature:



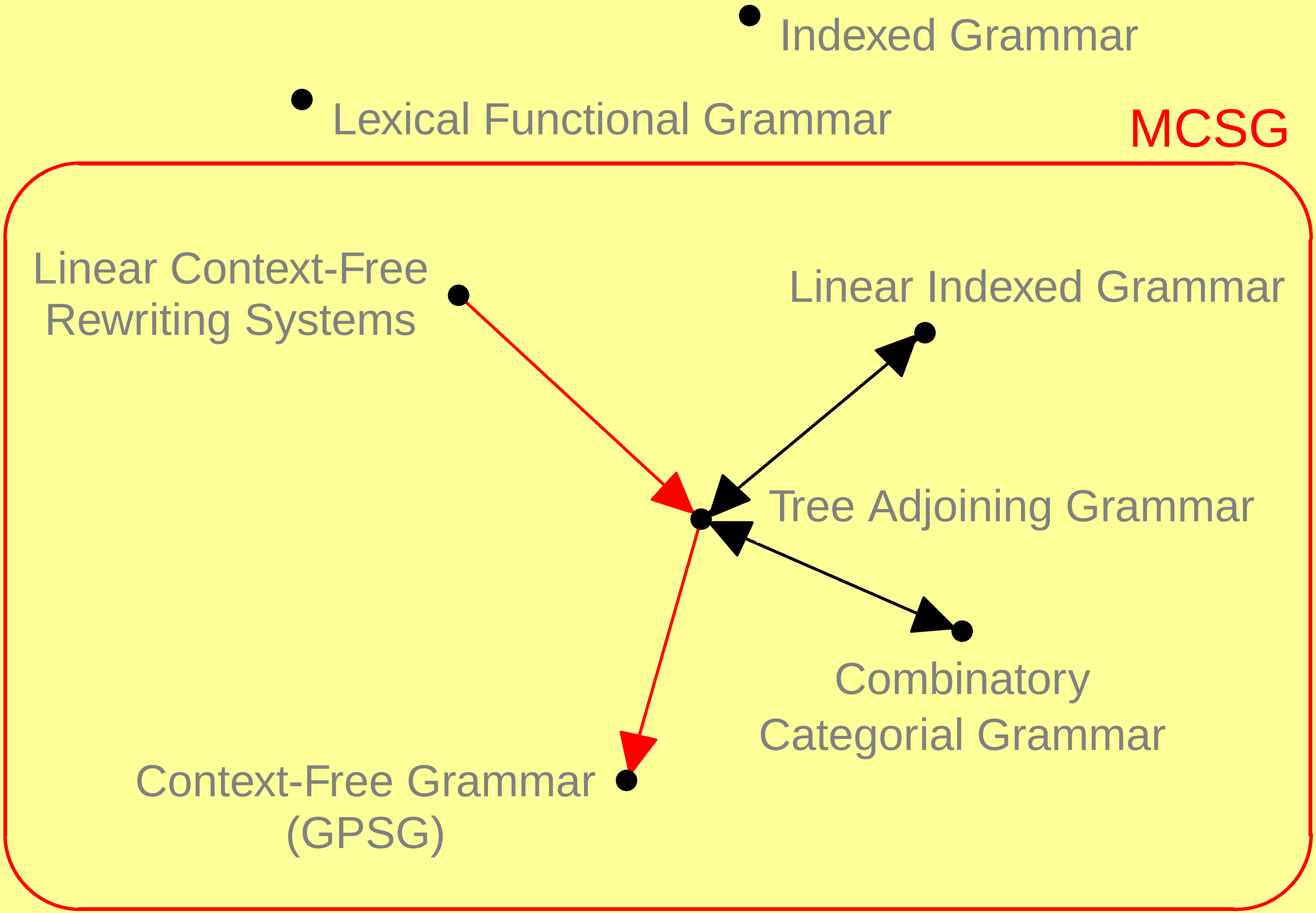
- Proper “extraction” from specifiers is blocked.

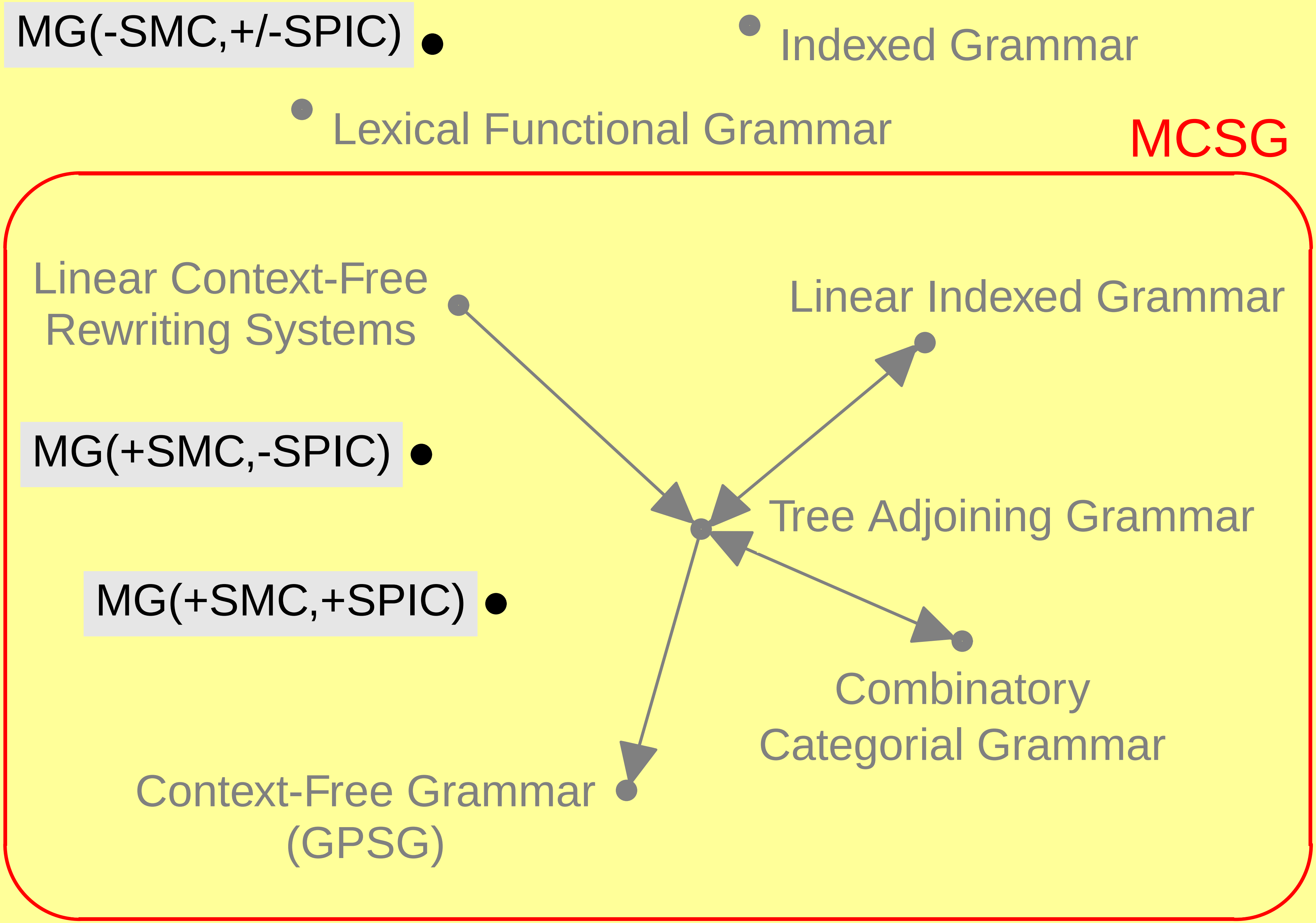


SMC and SPIC — restricting the move-operator domain

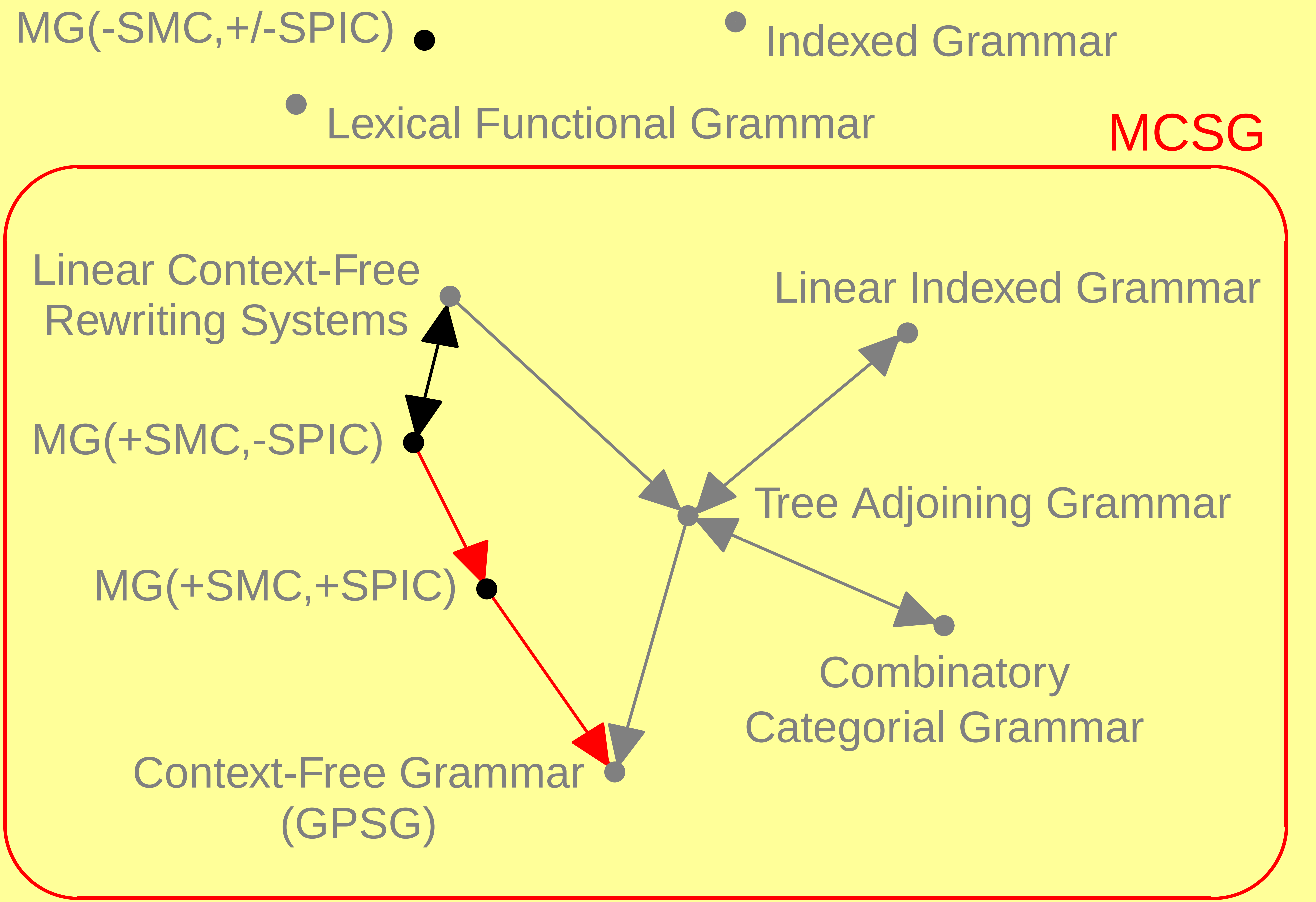


MCSG-landscape




MG(-SMC,+/-SPIC)
Indexed Grammar
Lexical Functional Grammar
MCSG
Linear Context-Free Rewriting Systems
Linear Indexed Grammar
MG(+SMC,-SPIC)
Tree Adjoining Grammar
MG(+SMC,+SPIC)
Combinatory Categorial Grammar
Context-Free Grammar (GPSG)

MCSG-landscape (enhanced)

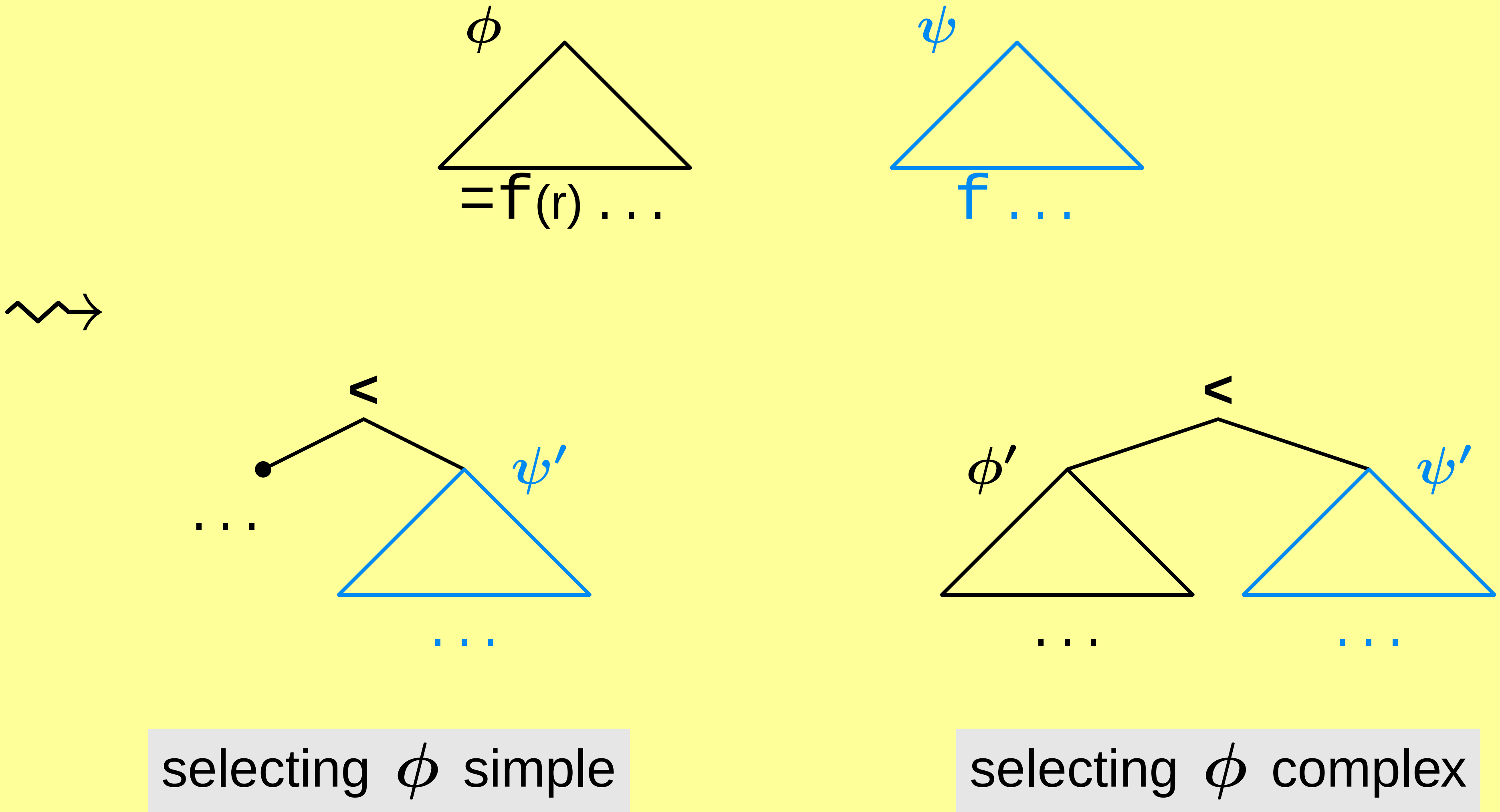


- There are different types of syntactic features.

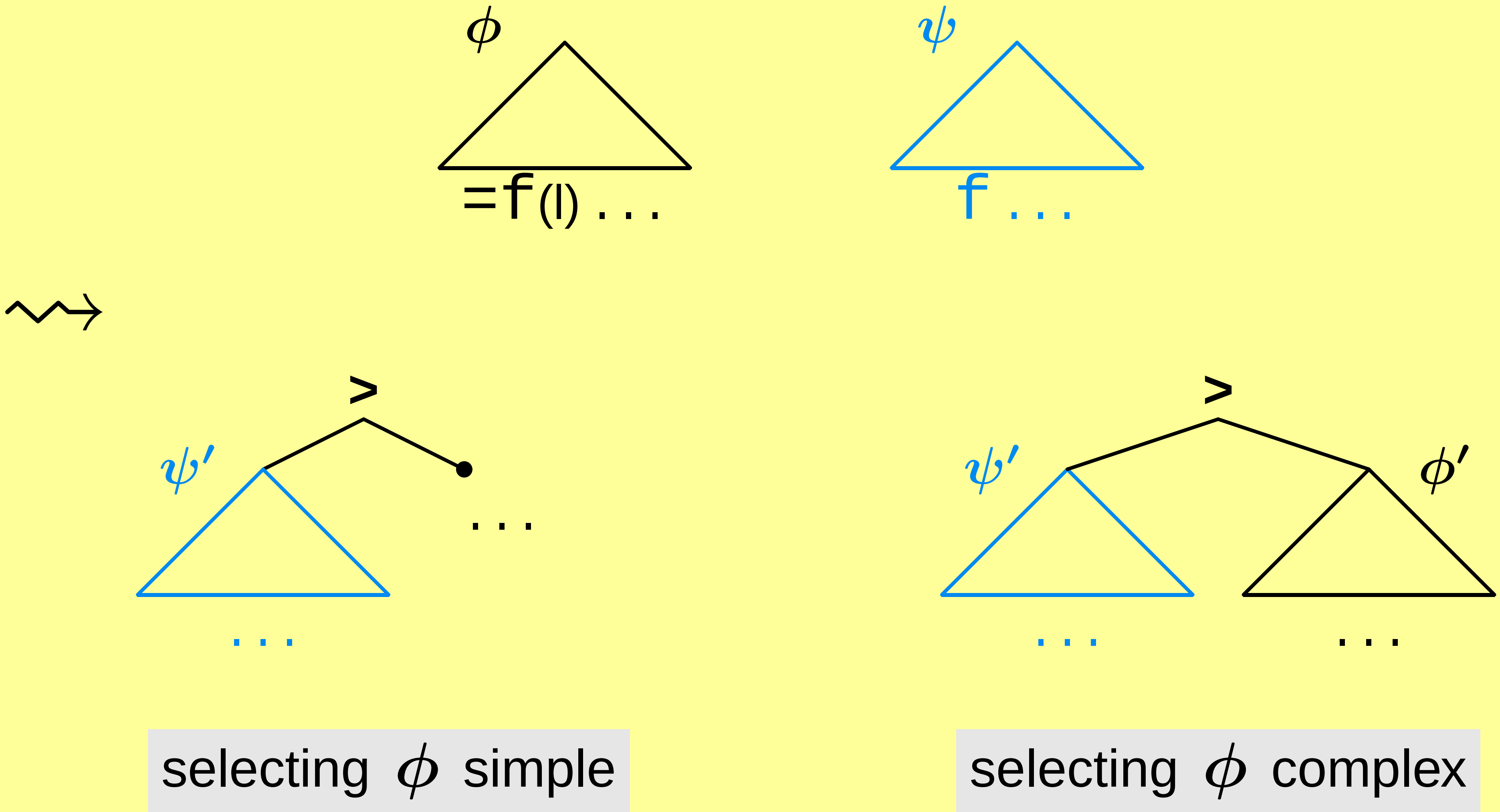
(basic) categories:	x , y , z , . . .	[Base]
(merge-) selectors:	=x(r) , =y(r) , =z(r) , . . .	[Selectors , right]
	. . .	
	=x(l) , =y(l) , =z(l) , . . .	[Selectors , left]
	. . .	
(move-) licensees:	-x , -y , -z , . . .	[Licensees]
(move-) licensors:	+x , +y , +z , . . .	[Licensors]

. . .

merge : Trees × Trees $\xrightarrow{\text{part}}$ Trees



merge : Trees × Trees $\xrightarrow{\text{part}}$ Trees

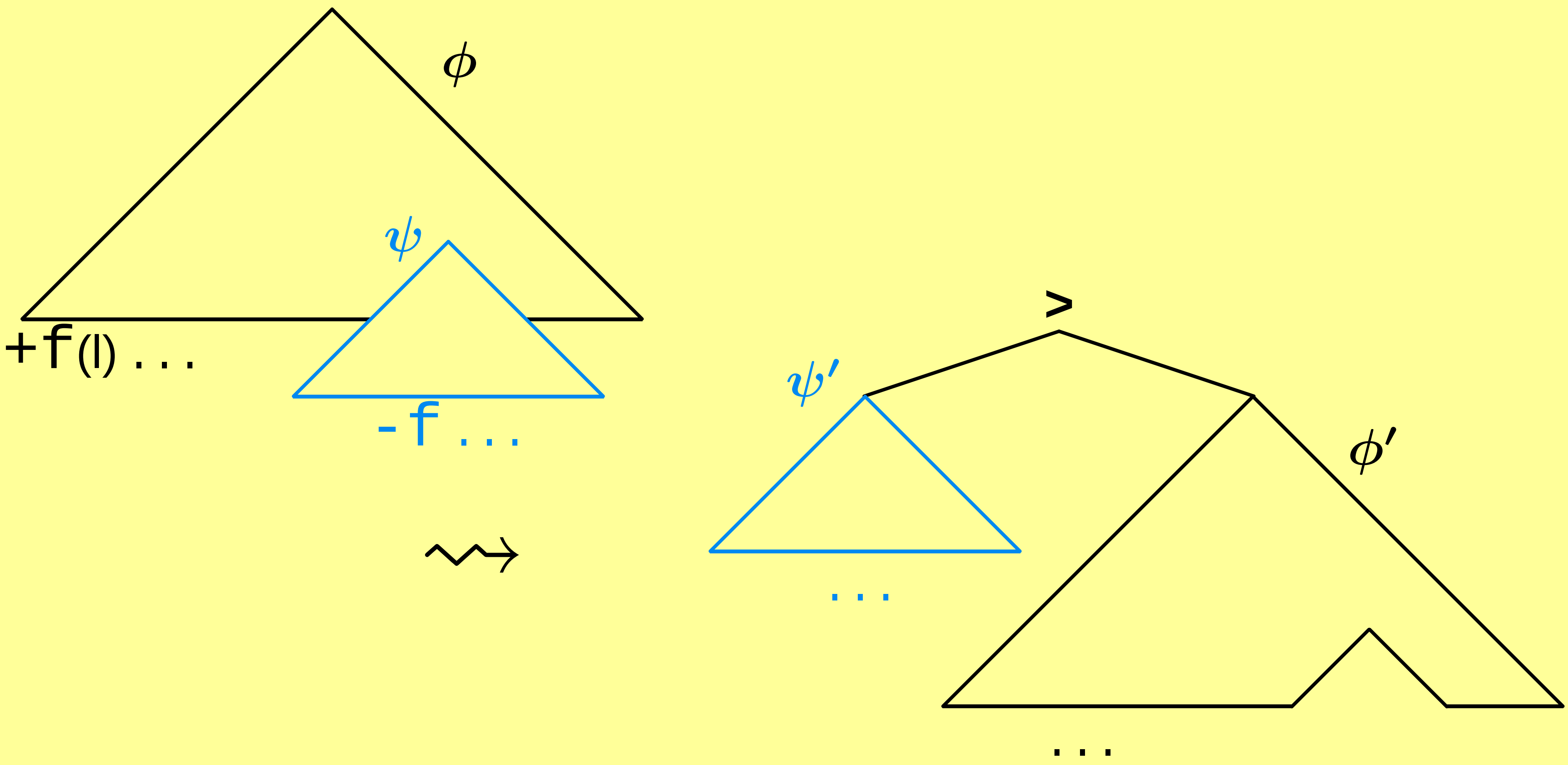


- There are different types of syntactic features.

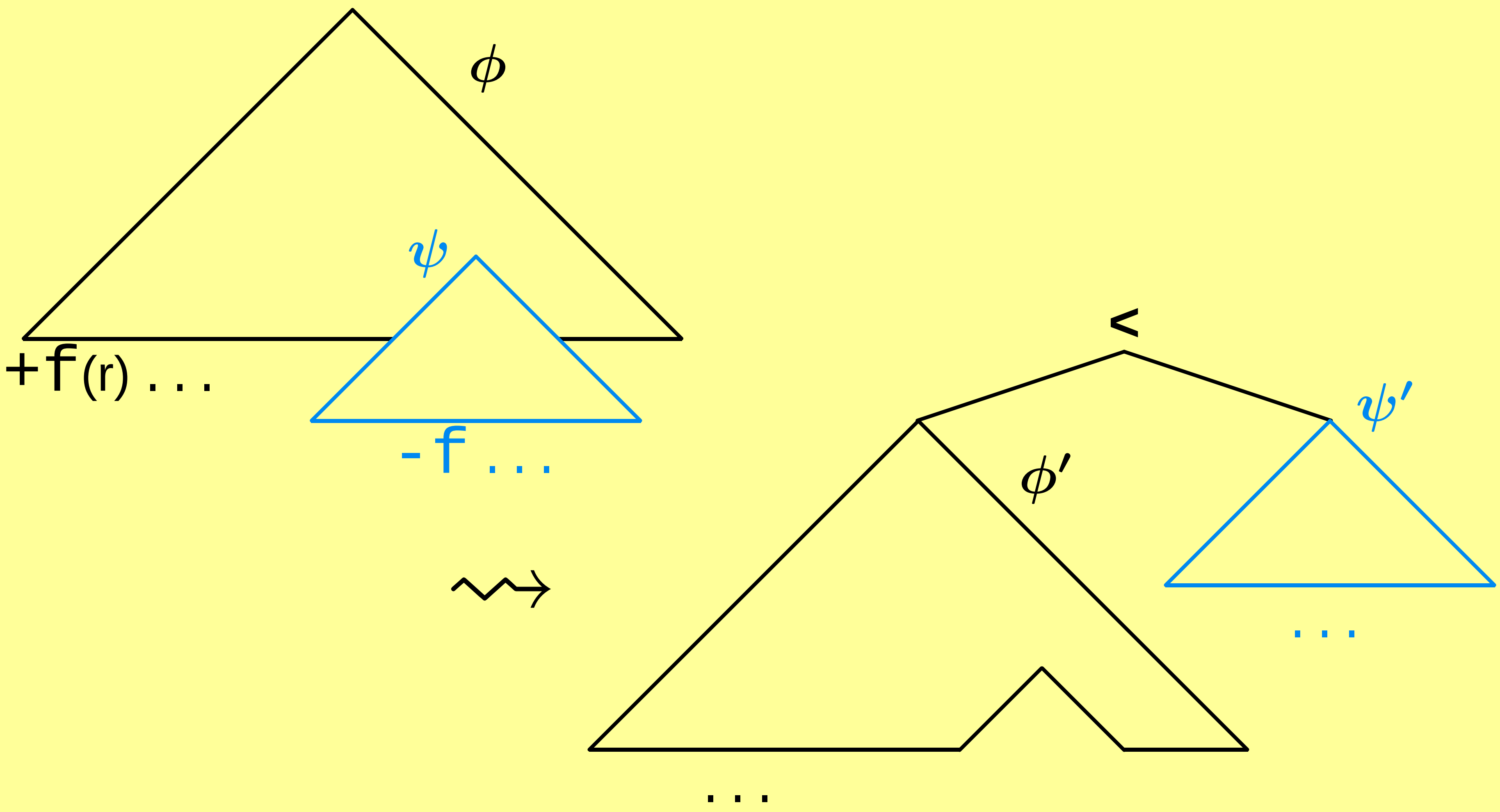
(basic) categories:	x , y , z , . . .	[Base]
(merge-) selectors:	. . .	[Selectors]
(move-) licensees:	-x , -y , -z , . . .	[Licensees]
(move-) licensors:	+x(l) , +y(l) , +z(l) , . . .	[Licensors , left]
	+x(r) , +y(r) , +z(r) , . . .	[Licensors , right]

. . .

$$\text{move} : \text{Trees} \xrightarrow{\text{part}} 2^{\text{Trees}}$$



move : Trees $\xrightarrow{\text{part}}$ 2^{Trees}

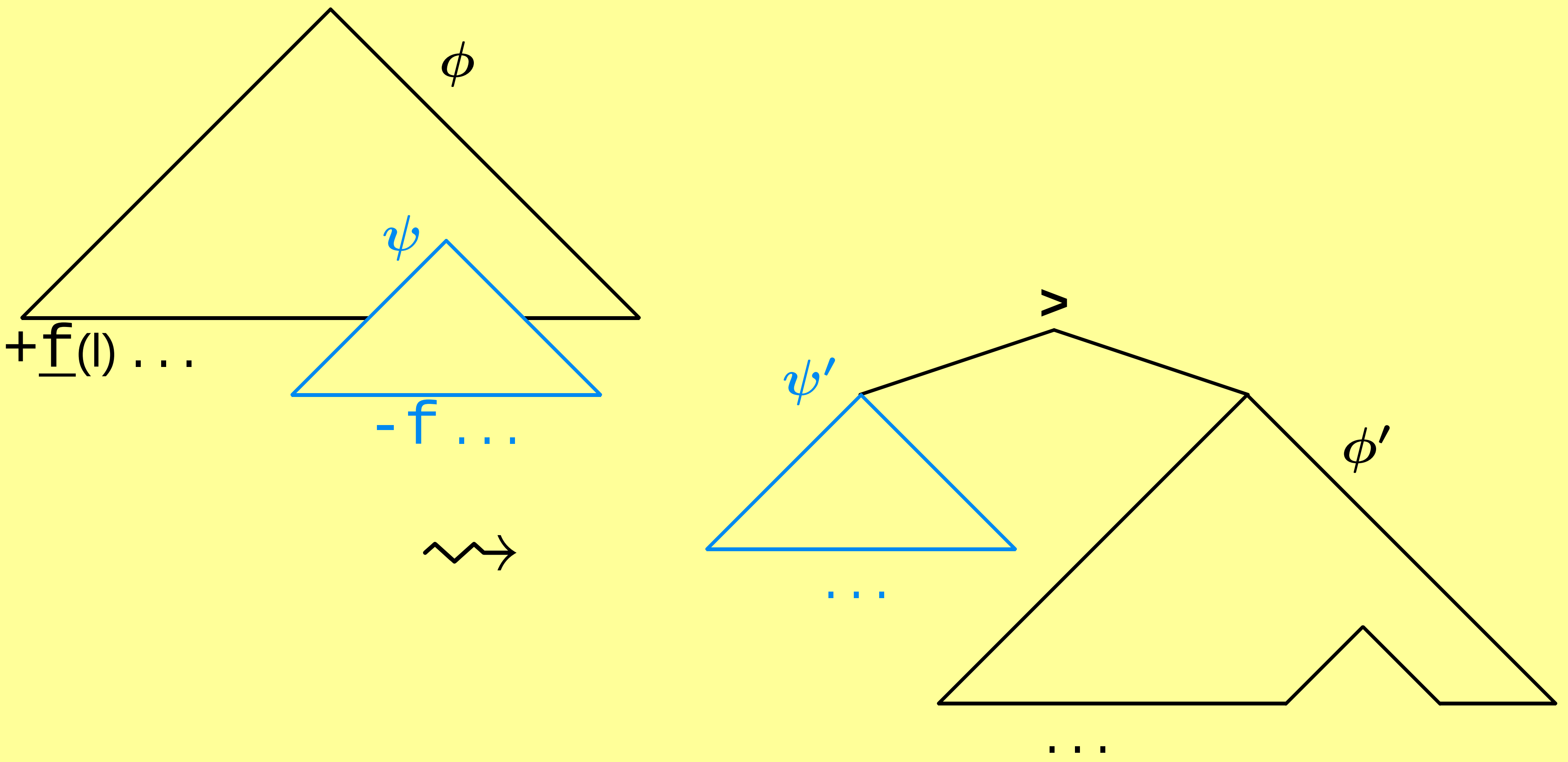


- There are different types of syntactic features.

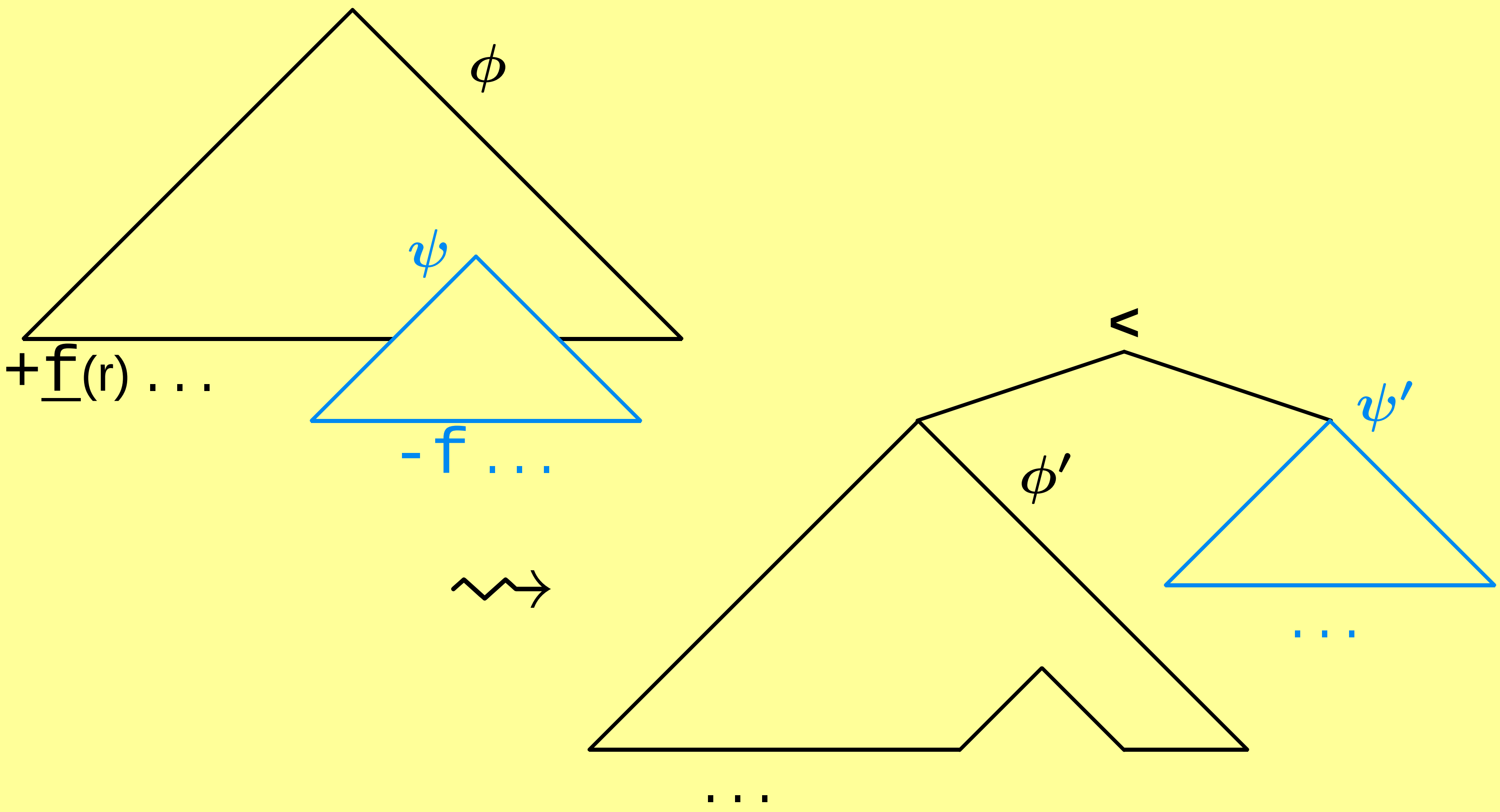
(basic) categories:	x , y , z , . . .	[Base]
(merge-) selectors:	. . .	[Selectors]
(move-) licensees:	-x , -y , -z , . . .	[Licensees]
(move-) licensors:	+x , +y , +z , . . .	[Licensors , weak]
	+<u>x</u>(l) , +<u>y</u>(l) , +<u>z</u>(l) , . . .	[Licensors , strong]
	+<u>x</u>(r) , +<u>y</u>(r) , +<u>z</u>(r) , . . .	

. . .

move : Trees $\xrightarrow{\text{part}}$ 2^{Trees}



$\mathsf{move} : \mathsf{Trees} \xrightarrow{\text{part}} 2^{\mathsf{Trees}}$



Further outlook

- MGs can be extended with the operations adjoin and scramble involving two new types of syntactic features and a unilateral checking of their instantiations (Frey & Gärtner 2002, Gärtner & Michaelis 2003).

- If, in particular, categorial features are not deleted after checking, but marked as checked — and thus are still accessible — acyclic ("late") adjunction can be defined as a subtype of adjoin.

- As to the interaction of the SMC and a corresponding adjunct island constraint (AIC), the addition of the AIC has no effect, independently of the presence of the SMC.