

An Introduction to Minimalist Grammars:  
*Complexity of the Shortest Move Constraint*

(July 22, 2009)

Gregory Kobele

Humboldt Universität zu Berlin

`kobele@rz.hu-berlin.de`

Jens Michaelis

Universität Bielefeld

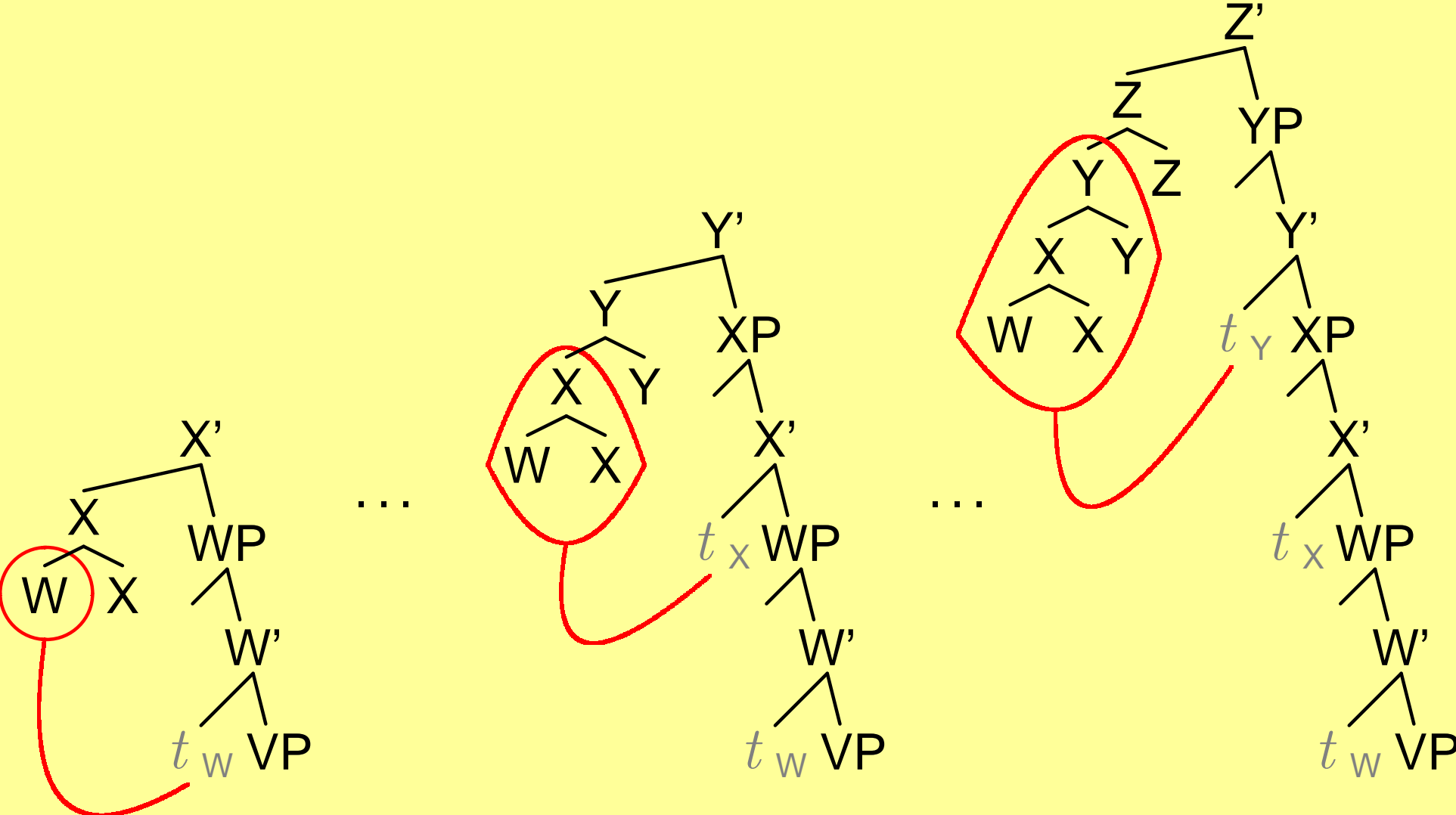
`jens.michaelis@uni-bielefeld.de`

- 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,

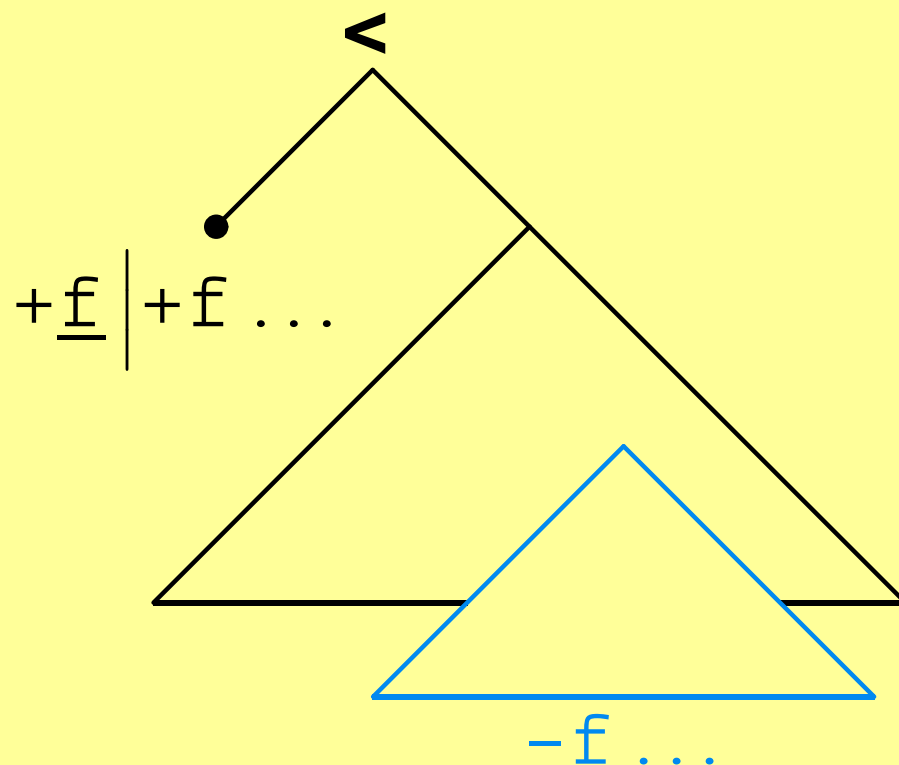
whenever we are concerned with a case of successive head movement, i.e. recursive adjunction of a (complex) head to a higher head, it obeys strict cyclicity.

# Successive cyclic left head adjunction

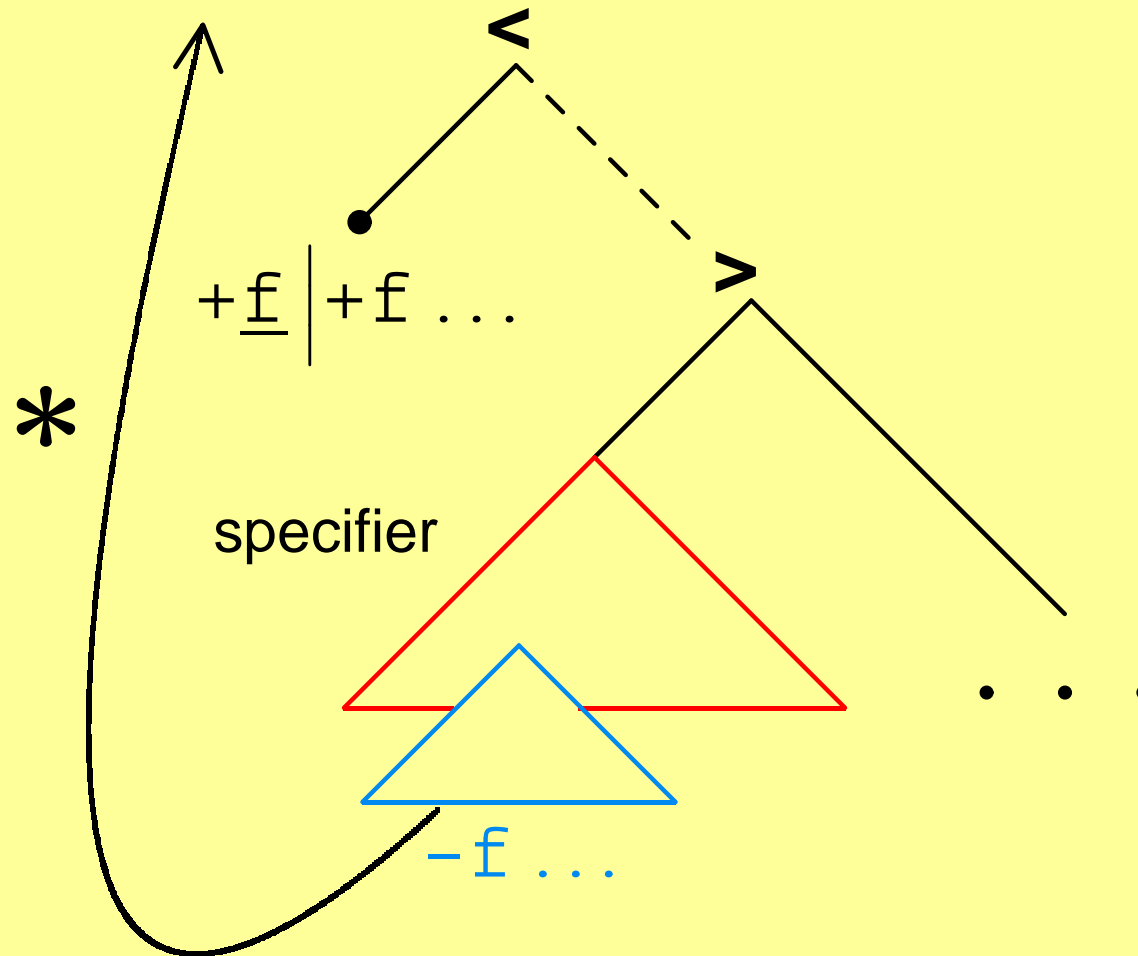


- The number of competing licensee features triggering a movement is (finitely) bounded by  $n$ .

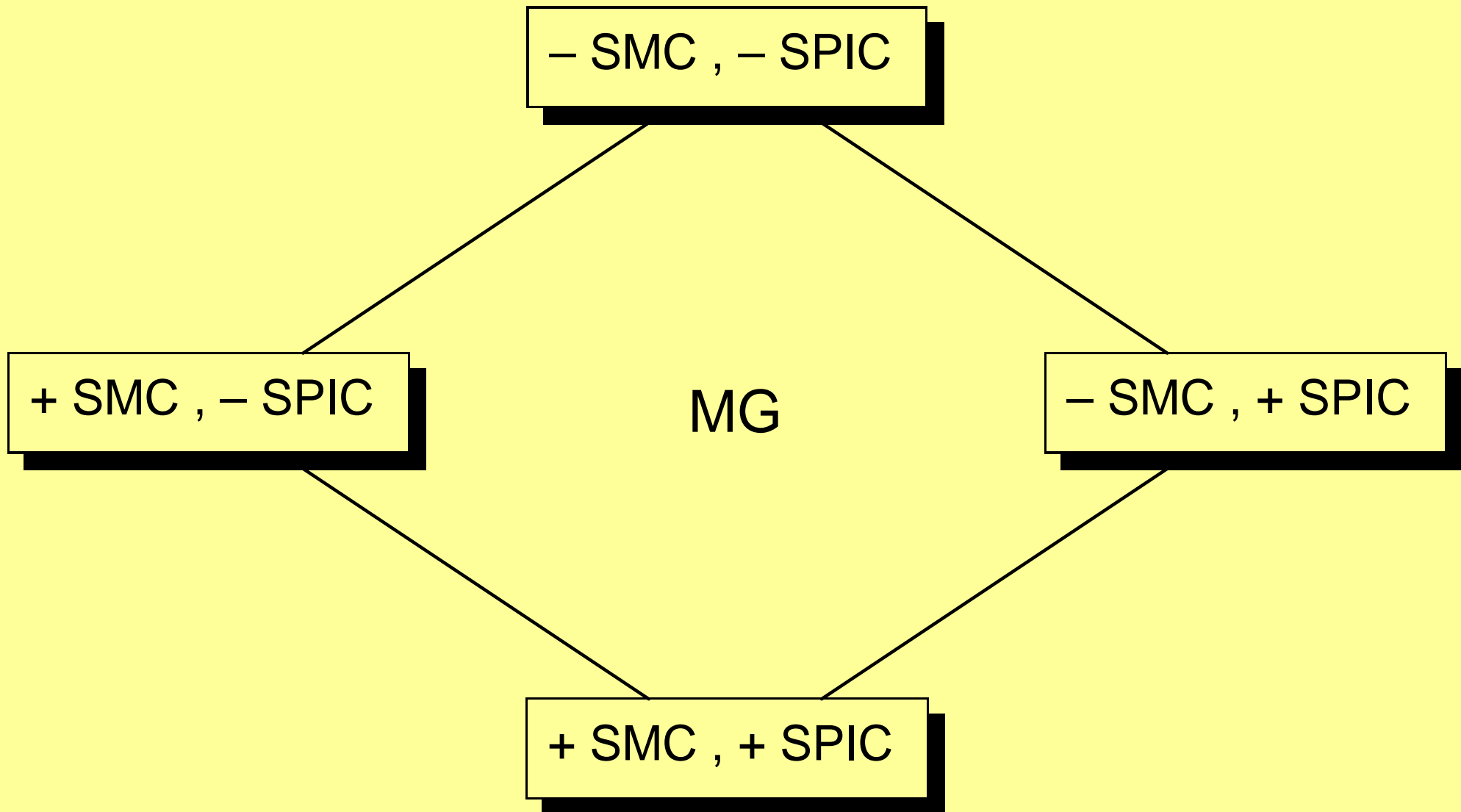
In the strictest version  $n = 1$ , i.e., there is at most one maximal projection displaying a matching licensee feature:



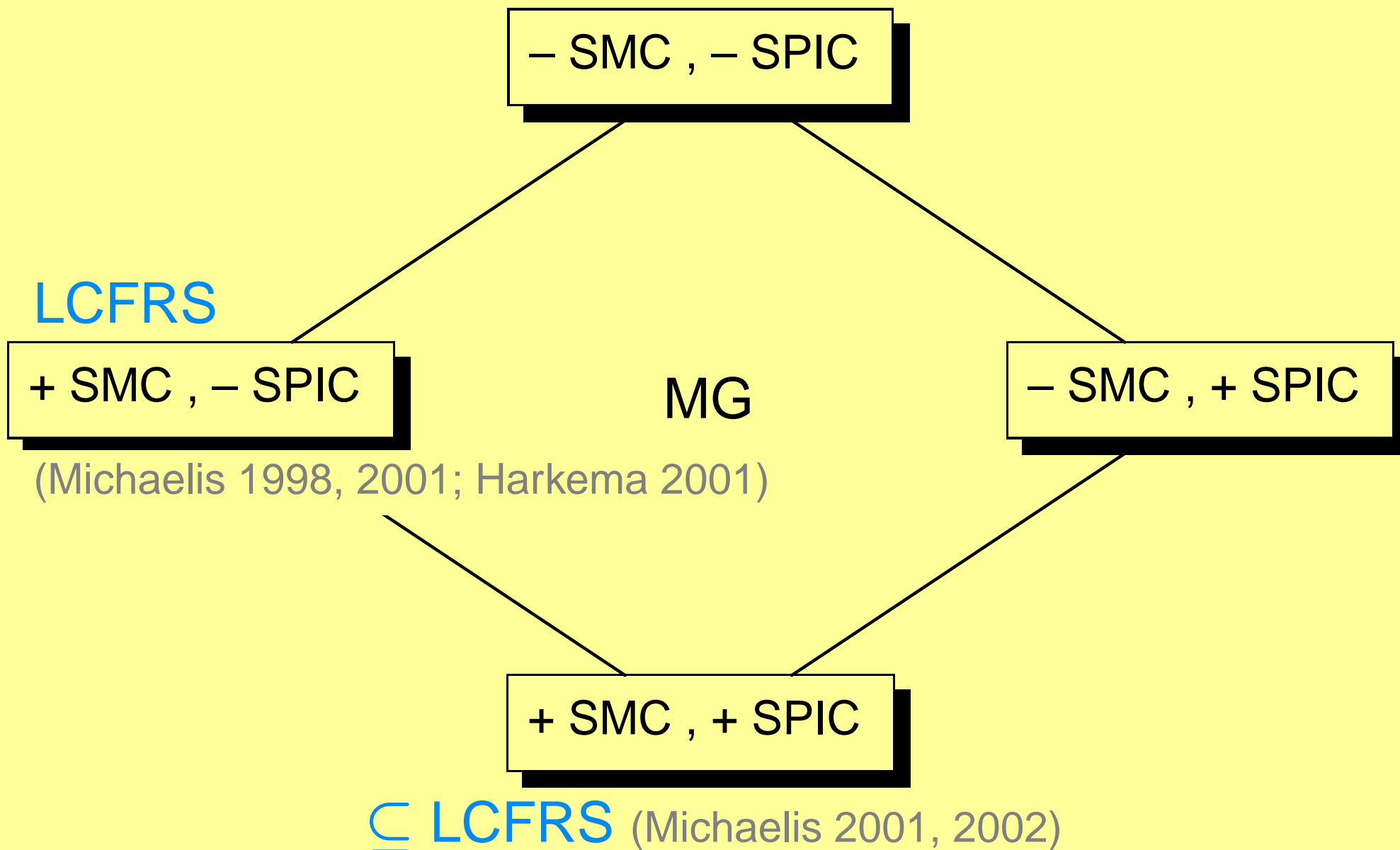
- Proper “extraction” from specifiers is blocked.



# SMC and SPIC — restricting the move-operator domain



# SMC and SPIC — restricting the move-operator domain



## + SMC , – SPIC — generative capacity

- The crucial methods, in particular,
  - ◆ developed to prove that MGs provide a weakly equivalent subclass of LCFRSs (cf. Michaelis 1998), and
  - ◆ leading to the succinct, chain-based MG-reformulation presented in Stabler & Keenan 2000 [2003] — reducing “classical” MGs to their “bare essentials:”
- Defining a finite partition on the “relevant” MG-tree set,
  - giving rise to a finite set of nonterminals in LCFRS-terms,
  - deriving all possible “terminal yields.”

# Reducing an MG(+SMC,-/+SPIC)

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

A minimal expression  $\tau \in \text{Closure}(G)$  is **relevant** : $\iff$

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

# Reducing an MG(+SMC,-/+SPIC)

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

A minimal expression  $\tau \in \text{Closure}(G)$  is **relevant** : $\Leftrightarrow$

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

- In fact, this kind of structure is **characteristic of each expression  $\tau \in \text{Closure}(G)$  involved in creating a complete expression in  $G$  due to the SMC.**

# A finite partition of set of relevant expressions

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

- Reduce  $\tau$  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 set of relevant expressions

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

- Reduce  $\tau$  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.

↗ 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 set of relevant expressions

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

- Reduce  $\tau$  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.

⚡→ only **finitely many equivalence classes**

Relevance:

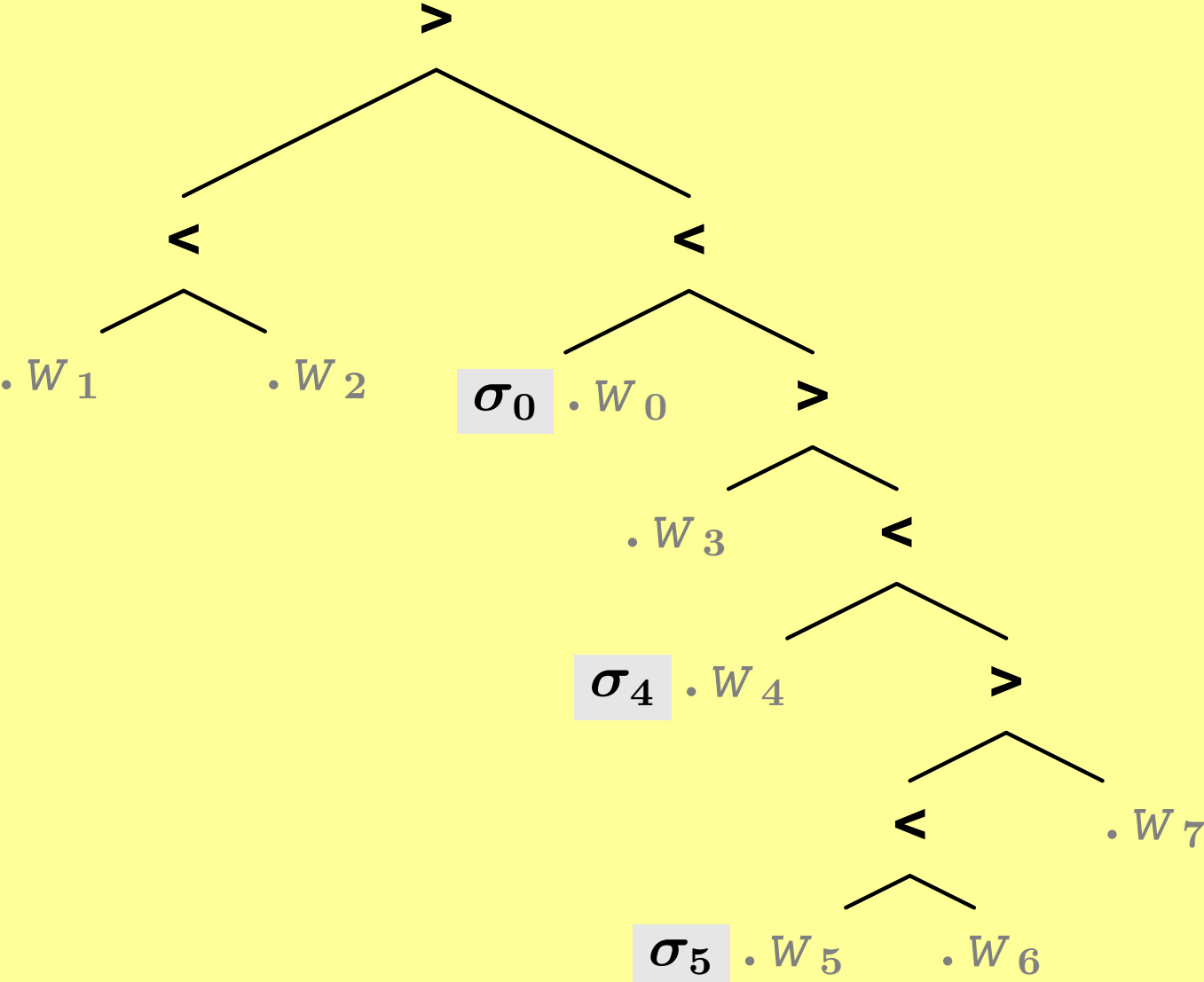
The resulting tuple has at most  $m+1$  components,  $m = |\text{Licensees}|$ .

Structure building by cancellation of features:

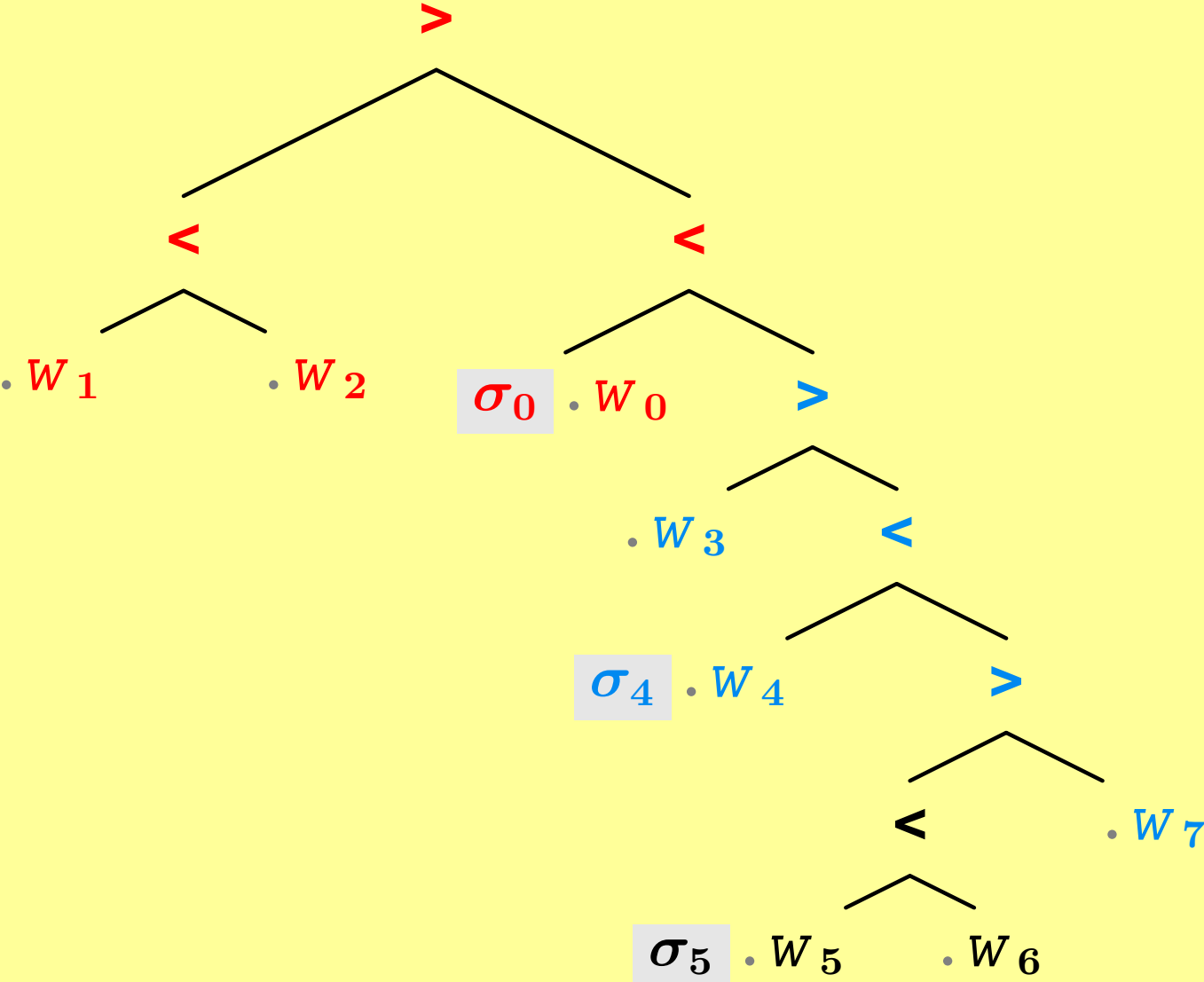
Each tuple component is the suffix of the syntactic prefix of the label of a lexical item.

⚡→ regarding the partition, applications of 'merge' and 'move' do not depend on the chosen representatives

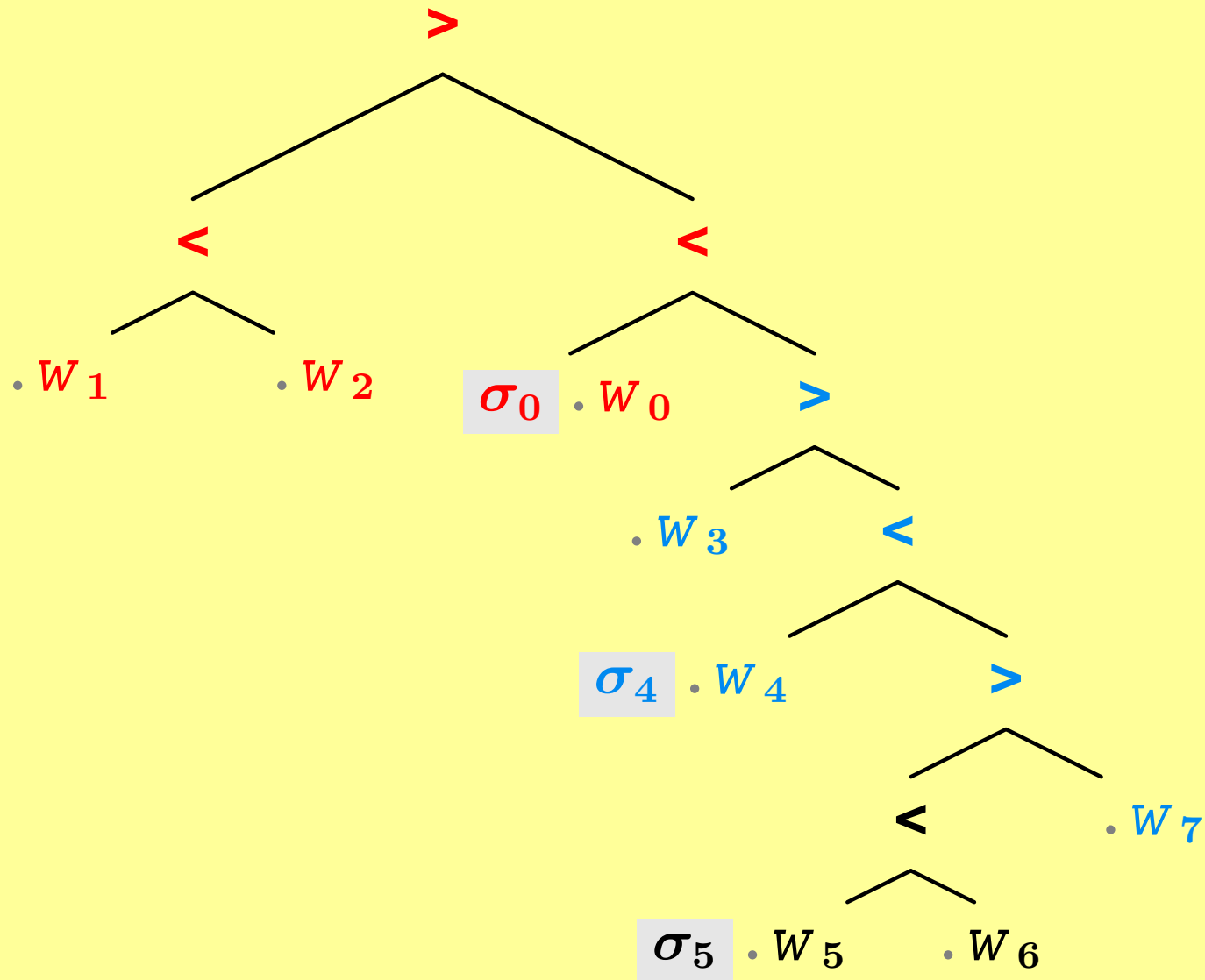
# Reducing an MG(+SMC,-/+SPIC)



# Reducing an MG(+SMC,-/+SPIC)

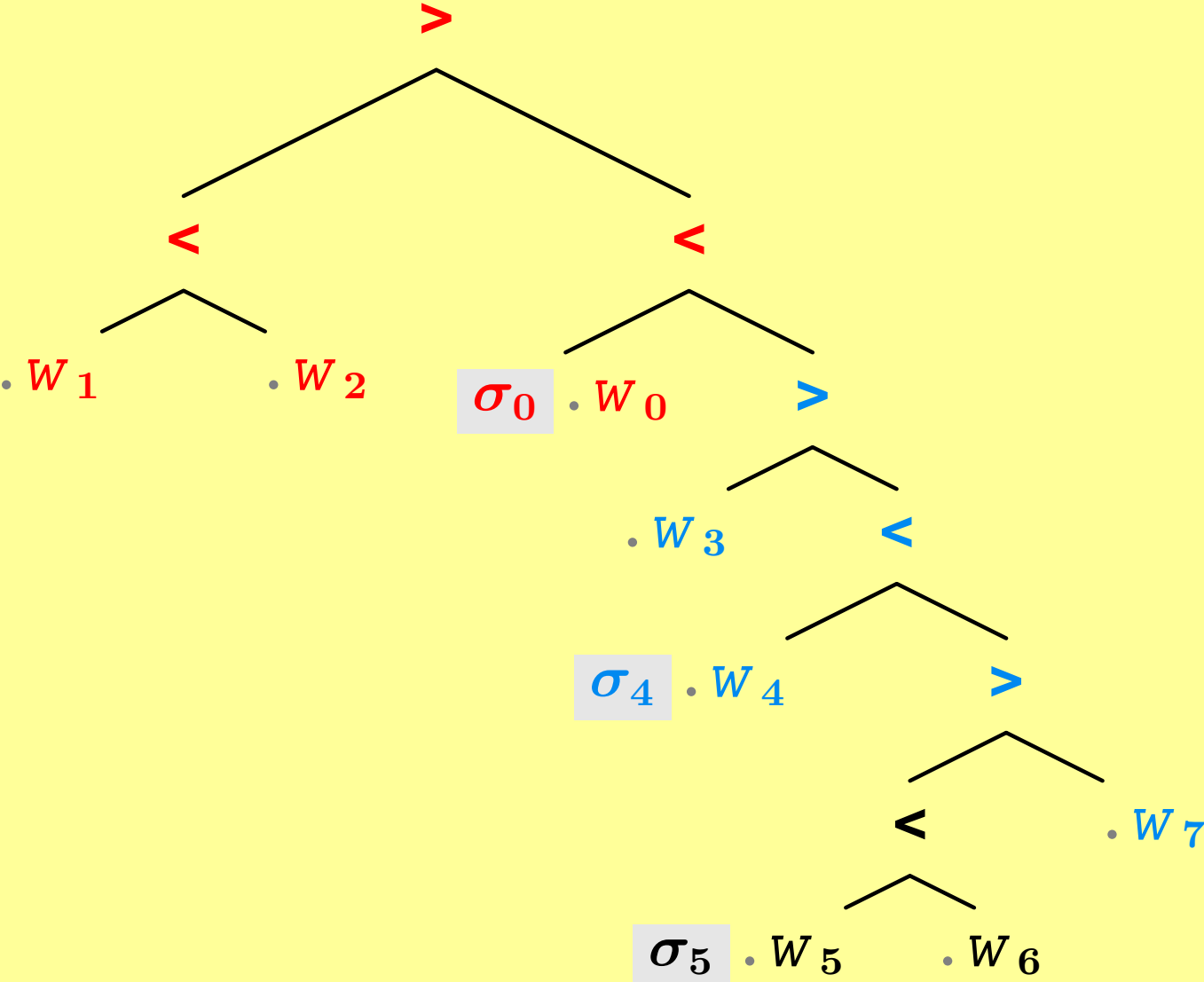


# Reducing an MG(+SMC,-/+SPIC)



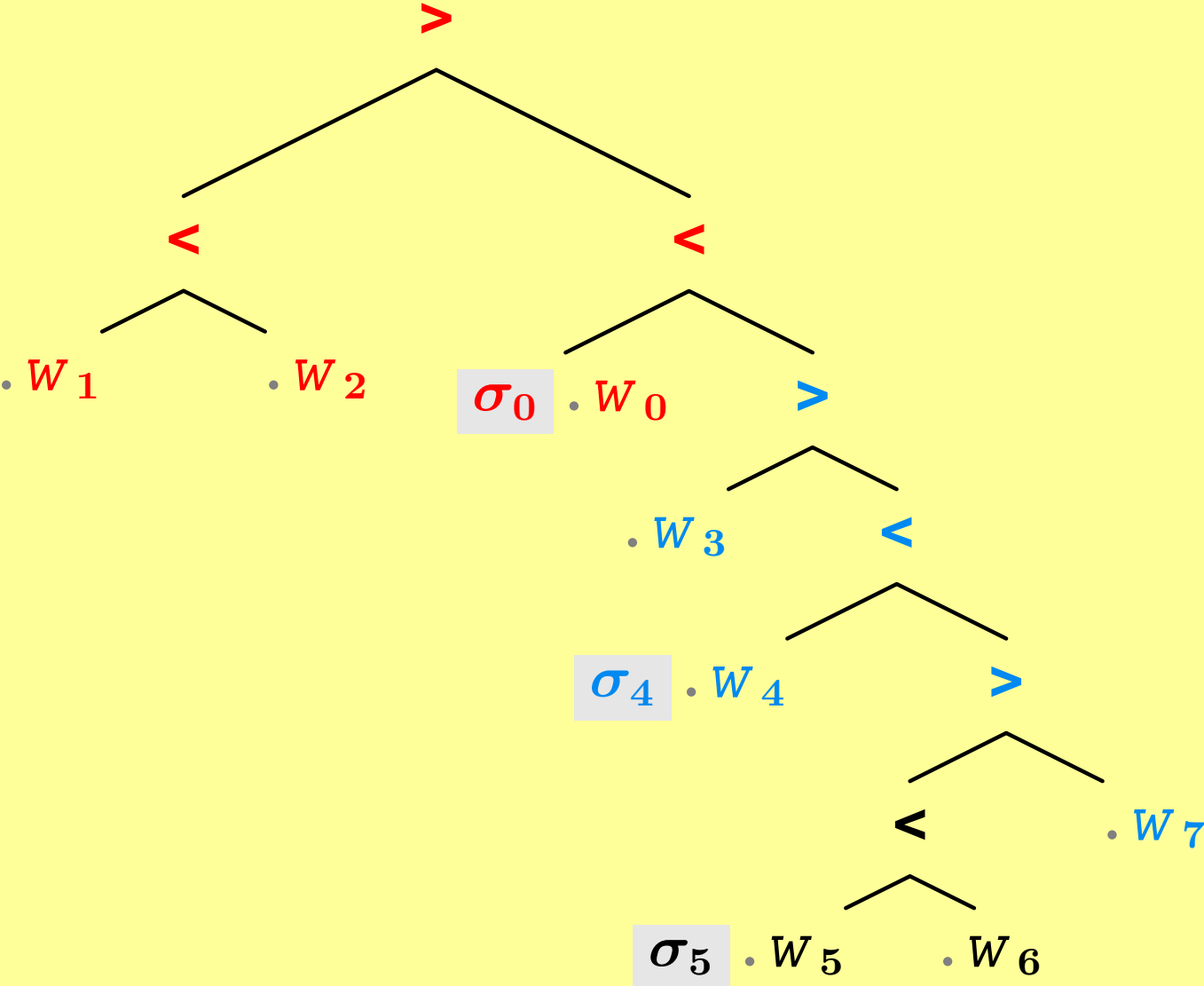
$\langle \sigma_0 .W_1W_2W_0 , \sigma_4 .W_3W_4W_7 , \sigma_5 .W_5W_6 \rangle$

# Reducing an MG(+SMC,-/+SPIC)



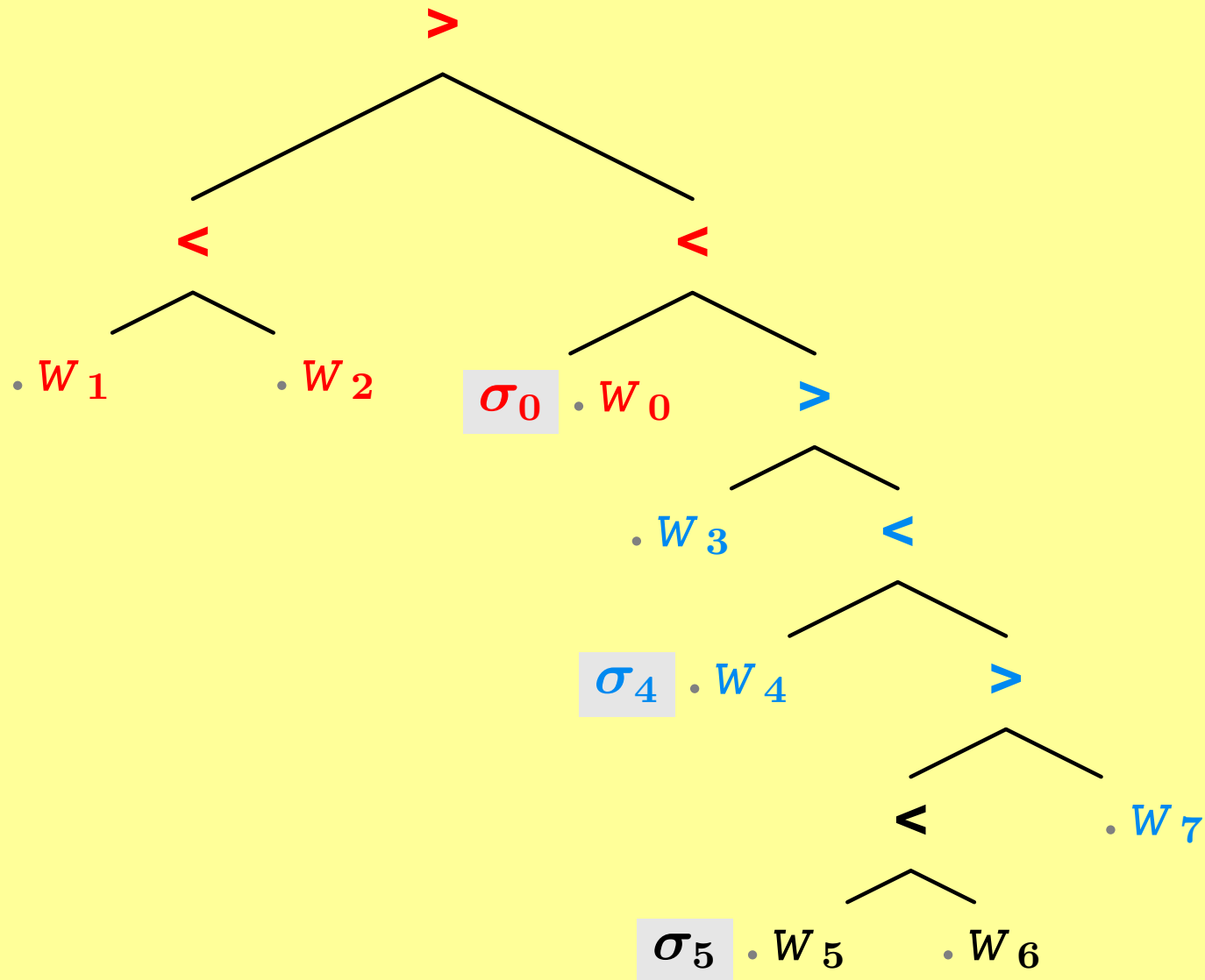
$\langle \sigma_0, \sigma_4, \sigma_5 \rangle$

# Reducing an MG(+SMC,-/+SPIC)



$\langle \sigma_0, \sigma_4, \sigma_5 \rangle$

# Reducing an MG(+SMC,-/+SPIC)



$$\langle \sigma_0, \sigma_4, \sigma_5 \rangle \Rightarrow \langle W_1W_2W_0, W_3W_4W_7, W_5W_6 \rangle$$

## MG-example 2

$(\alpha_0)$  = t . c . *that*

$(\alpha_5)$  v . *laugh*

$(\alpha_1)$  = t . + wh . c .  $\emptyset$

$(\alpha_6)$  = n . d . - k . *the*

$(\alpha_2)$  =  $\tilde{v}$  . + k . t .  $\emptyset$

$(\alpha_7)$  = n . d . - k . - wh . *which*

$(\alpha_3)$  = v . = d .  $\tilde{v}$  .  $\emptyset$

$(\alpha_8)$  n . *king*

$(\alpha_4)$  = d . + k . v . *eat*

$(\alpha_9)$  n . *pie*

## MG-example 2

=n .d .-k .-wh .*which*

n .*pie*

# MG-example 2

$:: \hat{=} \text{simple}$  ,  $: \hat{=} \text{complex}$

*=n.d.-k.-wh.which*

$\langle \text{=n.d.-k.-wh.which}, :: \rangle$

*n.pie*

$\langle \text{n.pie}, :: \rangle$

# MG-example 2

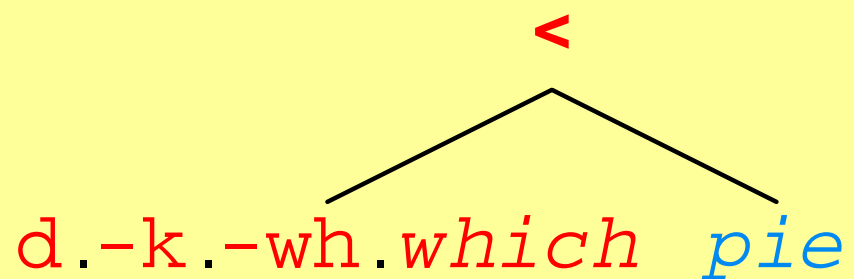
$:: \hat{=} \text{simple}$  ,  $: \hat{=} \text{complex}$

*=n.d.-k.-wh.which*

$\langle \text{=n.d.-k.-wh.which}, :: \rangle$

*n.pie*

$\langle \text{n.pie}, :: \rangle$



# MG-example 2

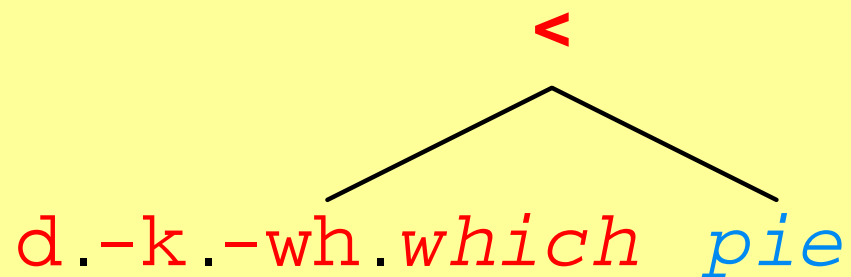
$:: \hat{=} \text{simple}$  ,  $: \hat{=} \text{complex}$

*=n.d.-k.-wh.which*

$\langle \text{=n.d.-k.-wh.which}, :: \rangle$

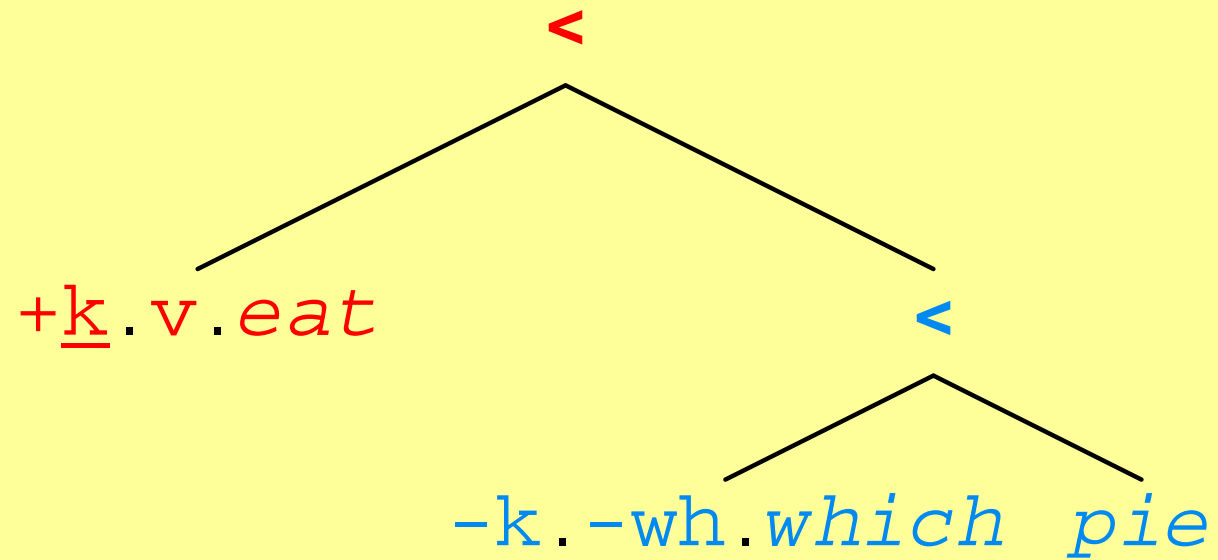
*n.pie*

$\langle \text{n.pie}, :: \rangle$



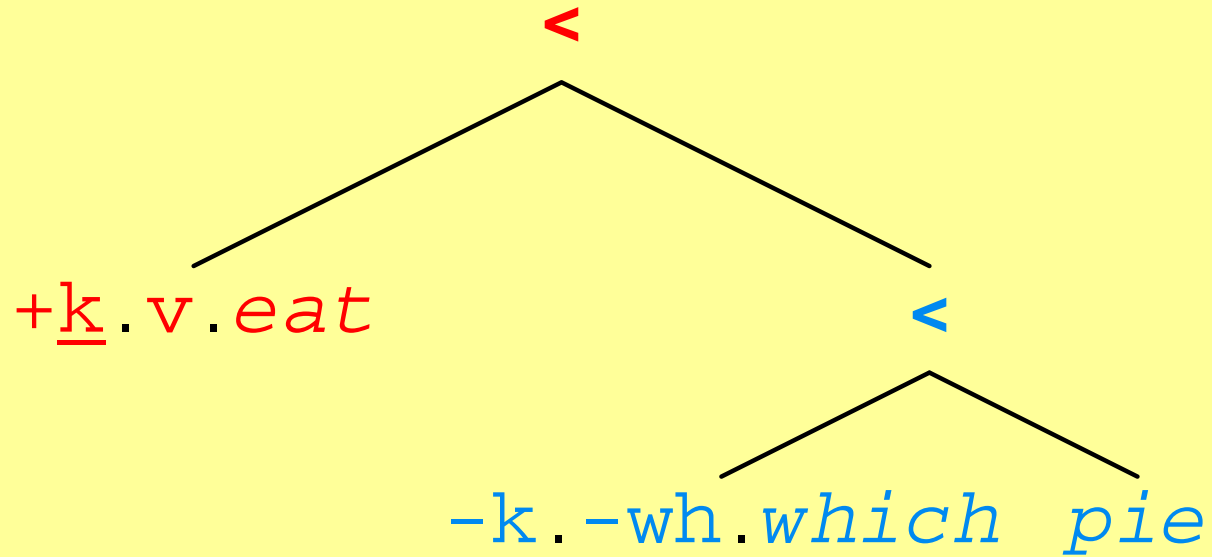
$\langle \text{d.-k.-wh.which pie}, : \rangle$

# MG-example 2



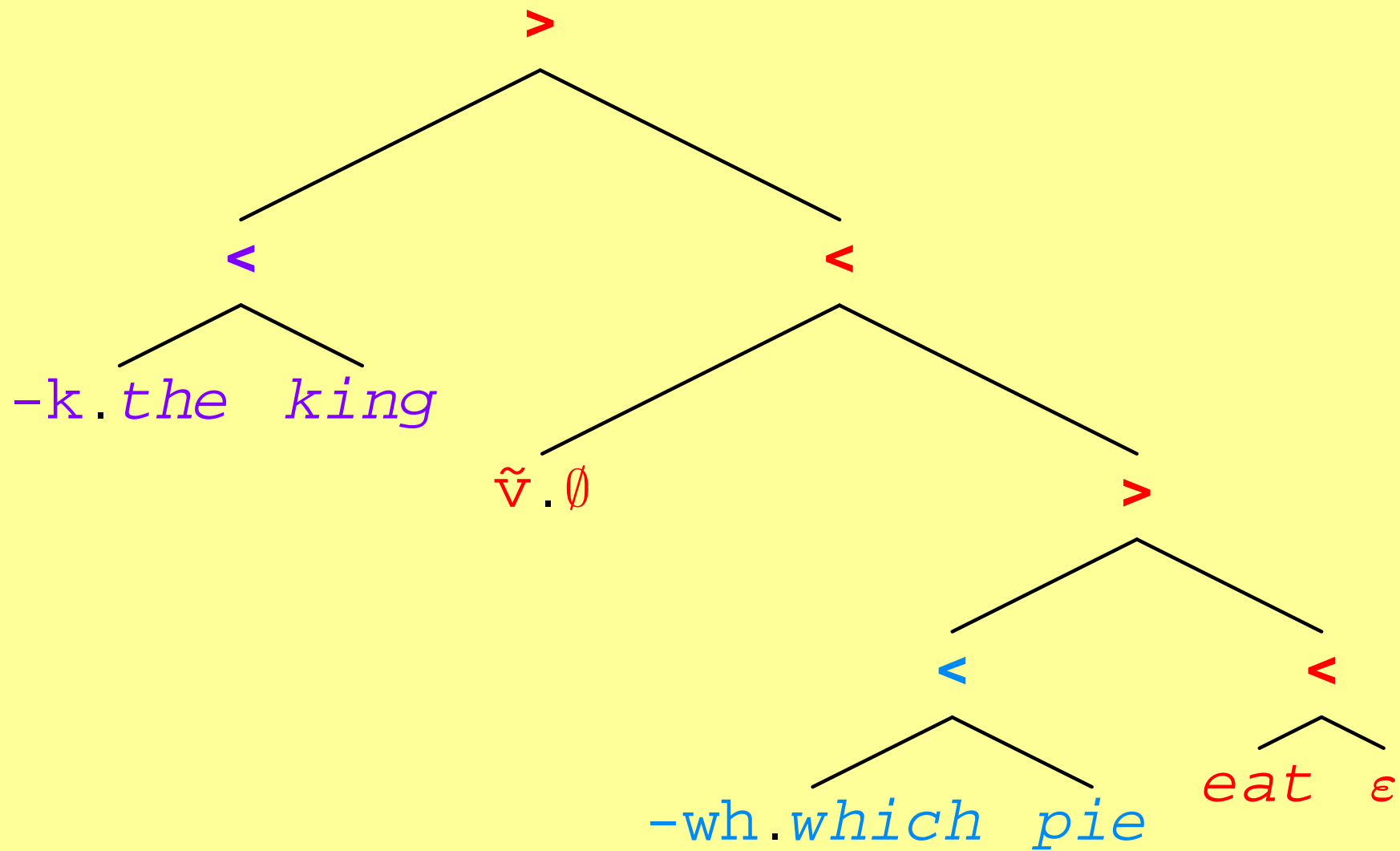
# MG-example 2

::  $\hat{=}$  simple , :  $\hat{=}$  complex



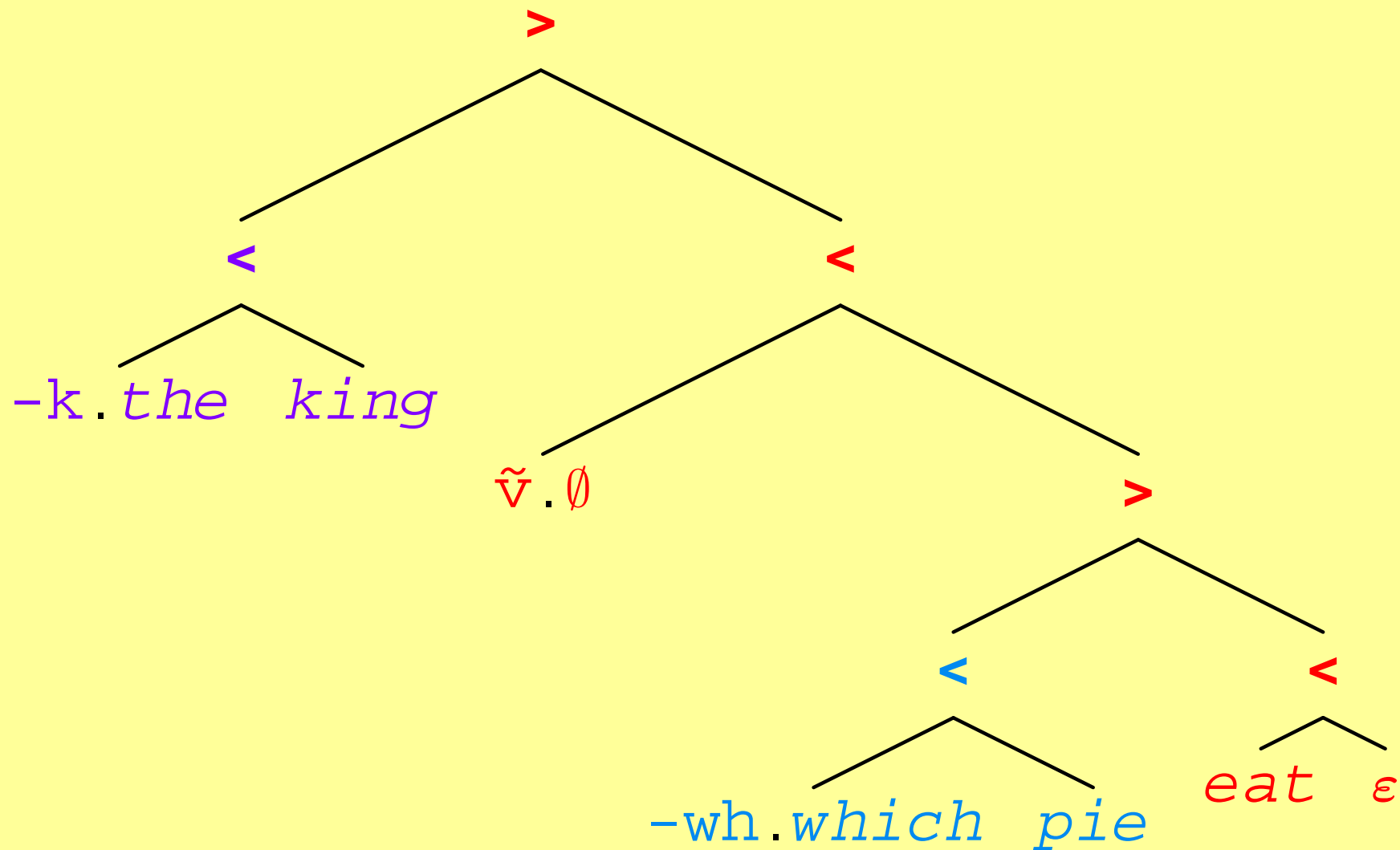
$\langle +\underline{k}.v.eat, -k.-wh.which\ pie, : \rangle$

# MG-example 2



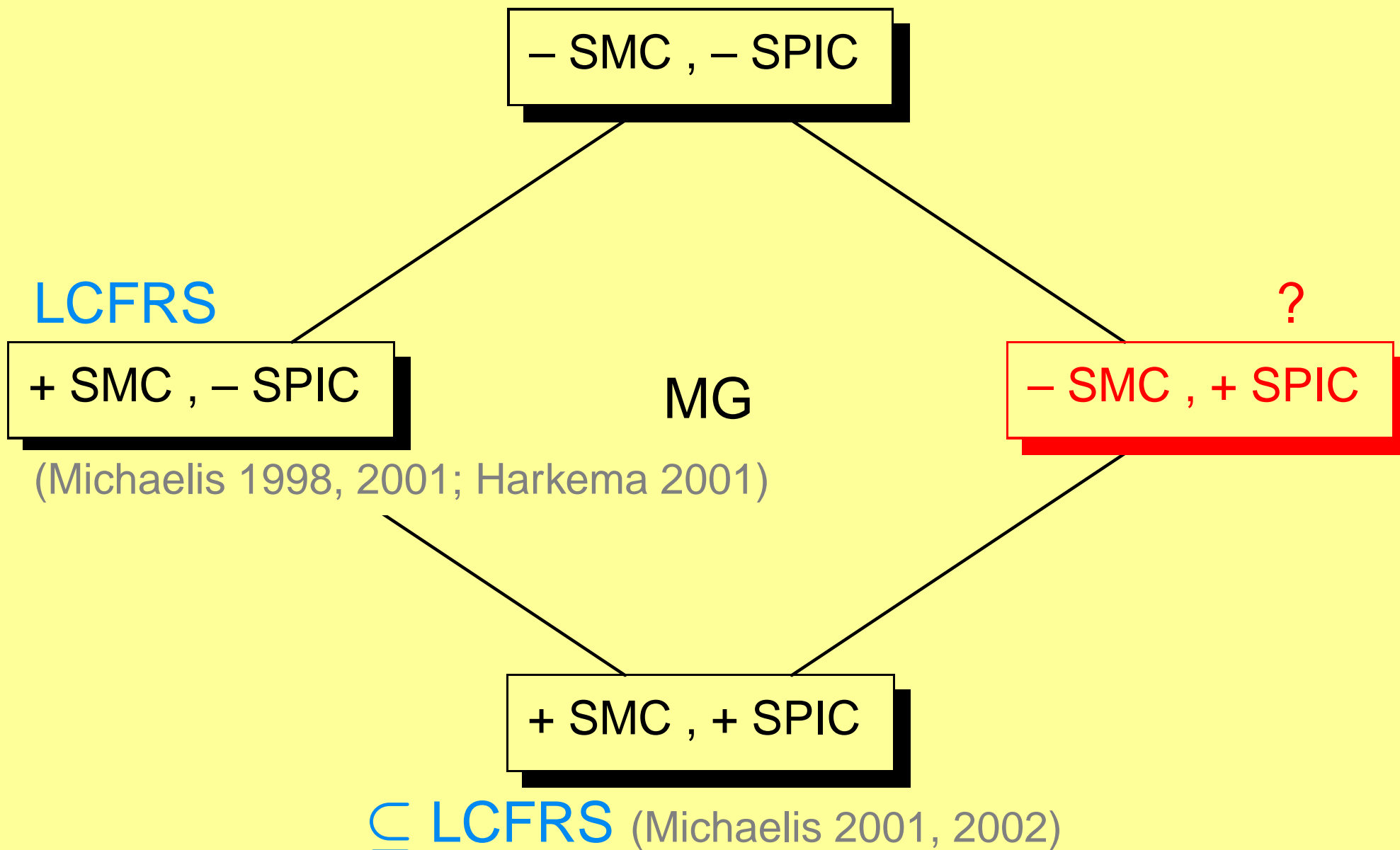
# MG-example 2

$:: \hat{=}$  simple ,  $: \hat{=}$  complex



$\langle \tilde{v}.eat, -wh.which\ pie, -k.the\ king, : \rangle$

# SMC and SPIC — restricting the move-operator domain

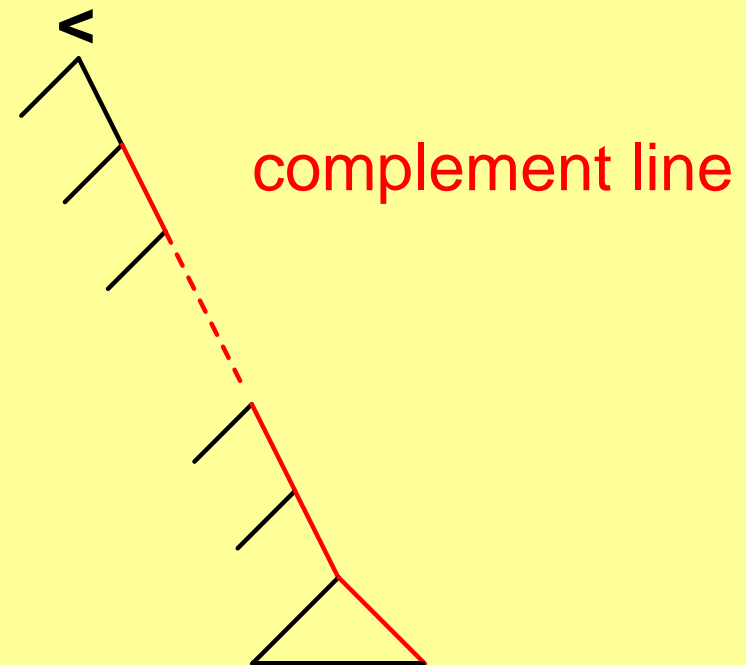
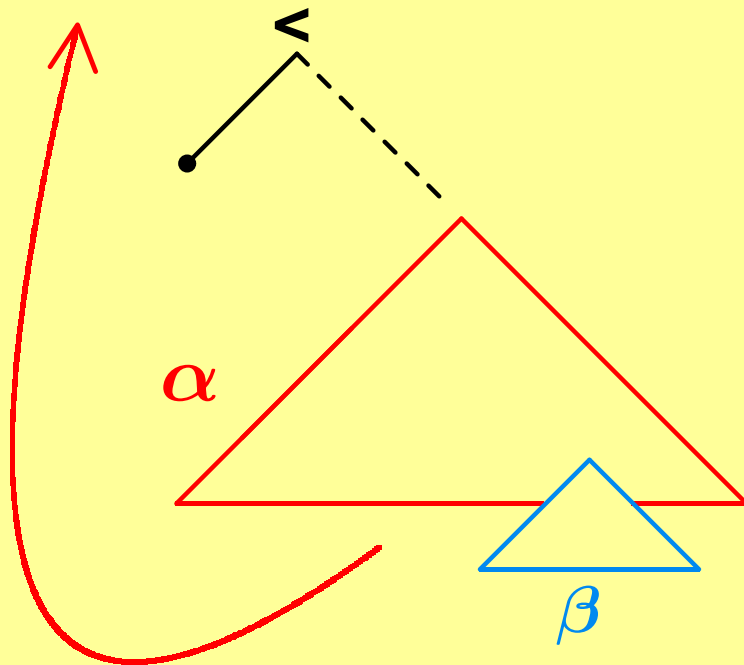


– SMC , + SPIC — generative capacity

- Gärtner & Michaelis 2005 shows that  $MG(-SMC,+SPIC)$ s allow derivation of non-mildly context-sensitive languages.
- Kobele & Michaelis 2005 shows that, in fact, **every recursively enumerable language can be derived by an  $MG(-SMC,+SPIC)$** .  
This is true for essentially two reasons:

– SMC , + SPIC — generative capacity

- Because of the SPIC, movement of a constituent  $\alpha$  into a specifier position freezes every proper subconstituent  $\beta$  within  $\alpha$ .
- Without the SMC, therefore, the complement line of a tree can technically be used as two independent counters, or, as a queue.



## MG-example — complexity results concerning LCs

- An example of a non-mildly context-sensitive MG(−SMC,+SPIC) deriving a language without constant growth property, namely,

$$\{ a^{2^n} \mid n \geq 0 \} = \{ a, aa, aaaa, aaaaaaaaa, \dots \}$$

1    2        4                    8                    ...

# MG-example — complexity results concerning LCs

w.-m

=w.x.-l

=x.+m.y.-m

=y.+l.z.-l

=z.y.-l

=z.x.-l

=x.+m.c

=c.+l.c.a

# MG-example — complexity results concerning LCs

licensee  $-m$  “marks”  
end/start of “outer” cycle

$w.-m$

“initialize”

$=w.x.-1$

---

end “outer” cycle “appropriately:”  
check licensee  $-m$

$=x.+m.y.-m$

“outer” cycle

start new “outer” cycle:  
introduce new licensee  $-m$

$=y.+1.z.-1$

“reintroduce” and “double”  
the just checked licensee  $-1$

$=z.y.-1$

“inner” cycle

$=z.x.-1$

---

leave final cycle “appropriately:”  
check licensee  $-m$

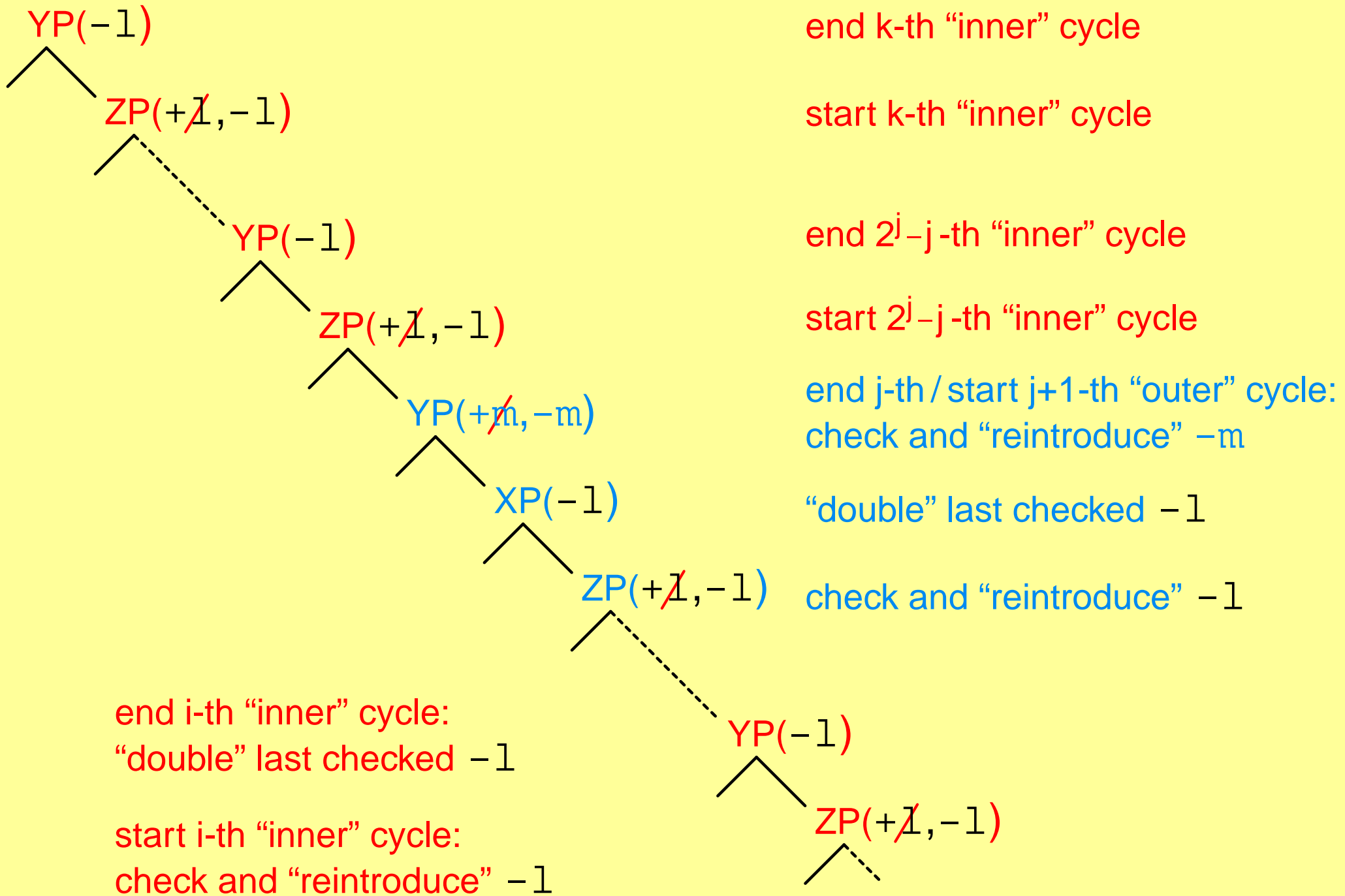
$=x.+m.c$

“finalize”

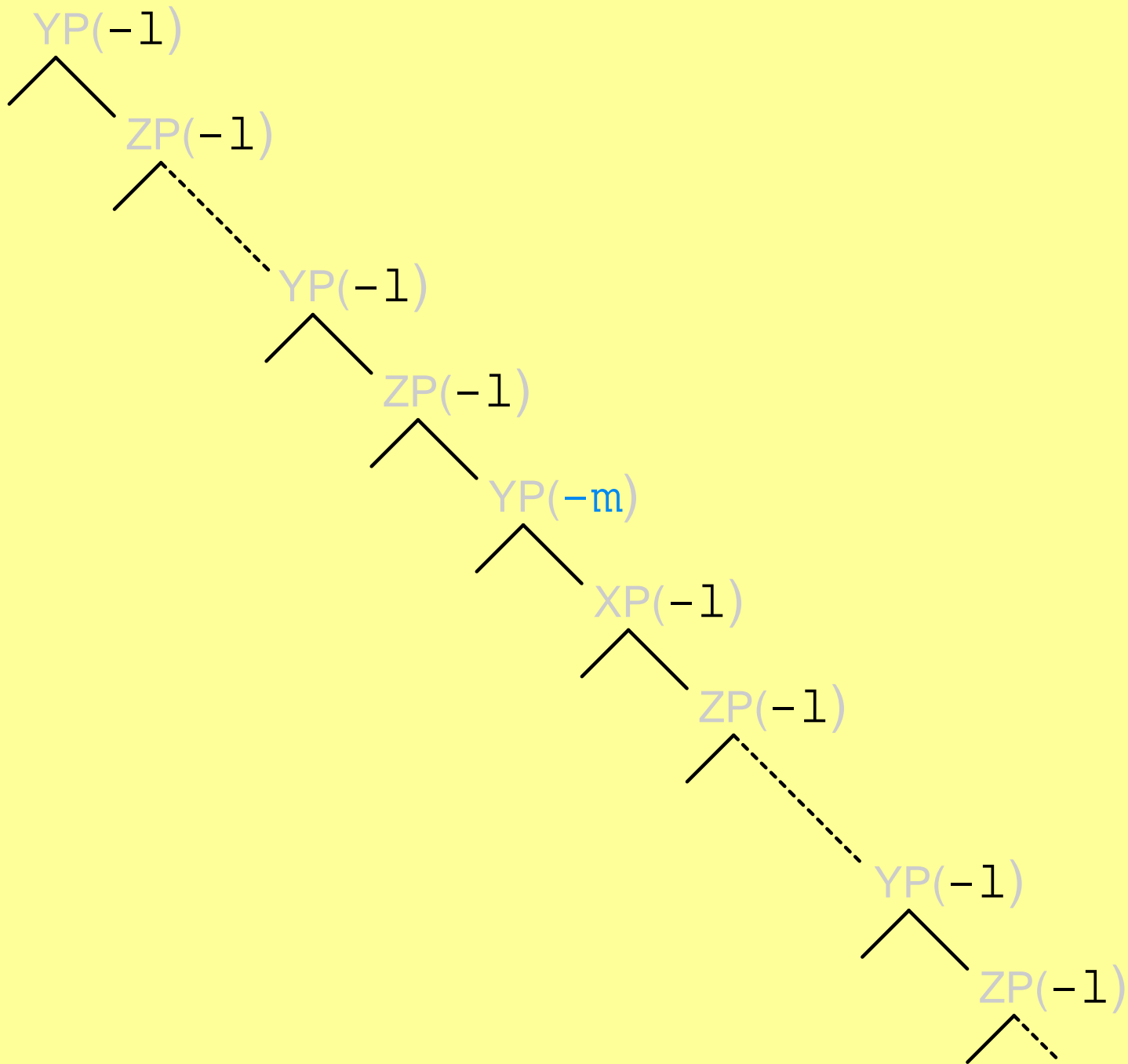
check successively licensee  $-1$ ,  
each time introducing an  $a$

$=c.+1.c.a$

# MG-example — complexity results concerning LCs



# MG-example — complexity results concerning LCs



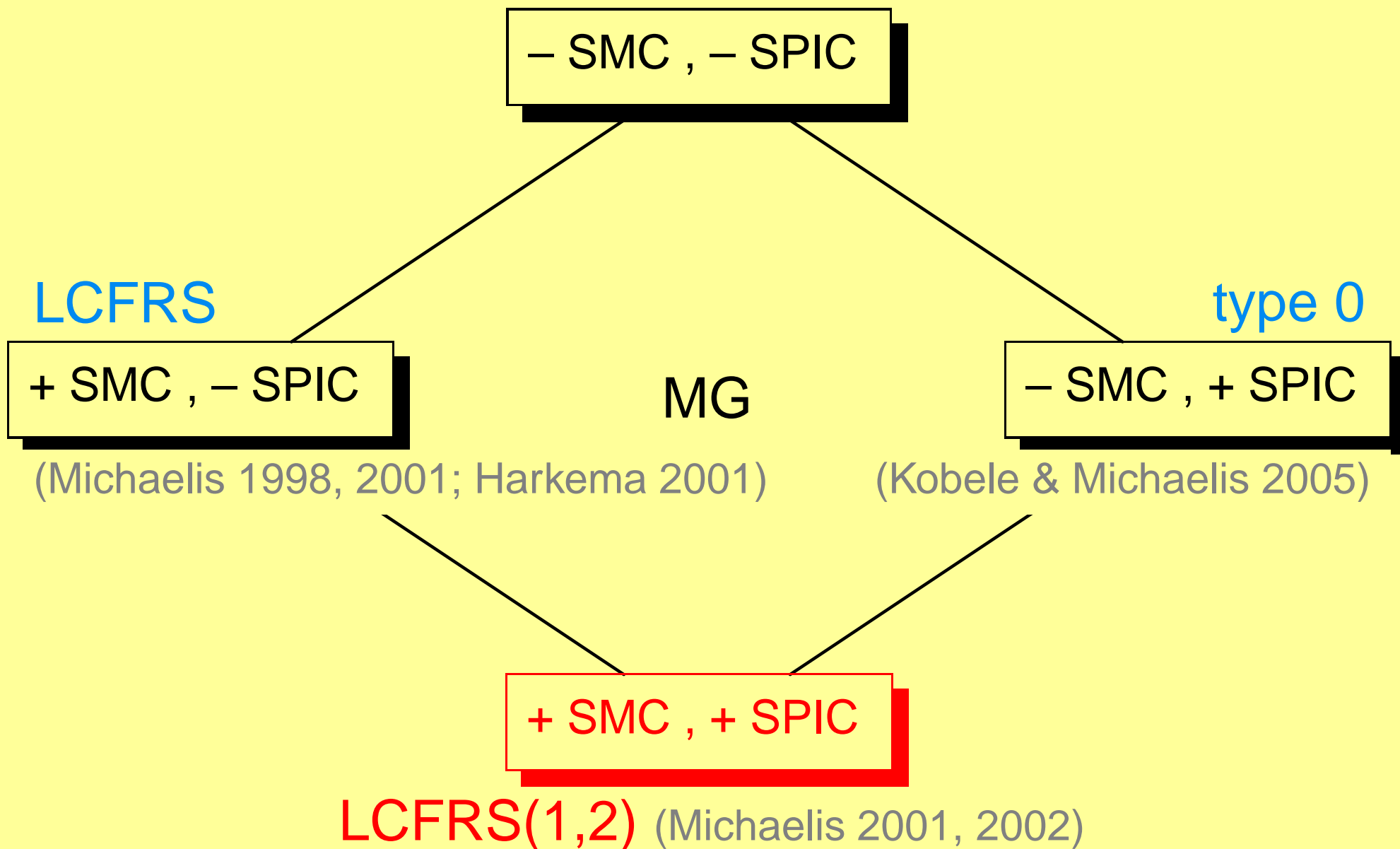
## MG-example — complexity results concerning LCs

- Starting the “outer” cycle, the currently derived tree shows  $2^n$  successively embedded complements on the complement line each with an unchecked instance of  $-1$ , and a lowest one with an unchecked instance of  $-m$ .
- Going through the cycle provides a successive “roll-up” of those complements in order to check the displayed features. Thereby,  $2^{n+1}$  successively embedded complements on the complement line are created, again, all displaying feature  $-1$  and a lowest one displaying feature  $-m$ .
- Leaving the cycle procedure after a cycle has been completed, leads to a final checking of the displayed licensees, where for each instance of  $-1$  an instance of  $a$  is introduced in the structure.

+ SMC , + SPIC — generative capacity

- In contrast to the – SMC , + SPIC - case,  
adding the SPIC to the SMC has a restrictive effect (Michaelis 2005)

+ SMC , + SPIC — generative capacity



## LCFRS(1,2) — a restricted LCFRS-normal form

An LCFRS  $G = \langle N, T, F, R, S \rangle$  is an **LCFRS(1,2)** iff

- ◆ each nonterminating rule is of the form  $A \rightarrow f(B)$  or  $A \rightarrow f(B, C)$ ,
- ◆ if  $A \rightarrow f(B, C)$ , nonterminal  $B$  derives only simple terminal strings.

## LCFRS(1,2) — a restricted LCFRS-normal form

An LCFRS  $G = \langle N, T, F, R, S \rangle$  is an **LCFRS(1,2)** iff

- ◆ each nonterminating rule is of the form  $A \rightarrow f(B)$  or  $A \rightarrow f(B, C)$ ,
- ◆ if  $A \rightarrow f(B, C)$ , nonterminal  $B$  derives only simple terminal strings.

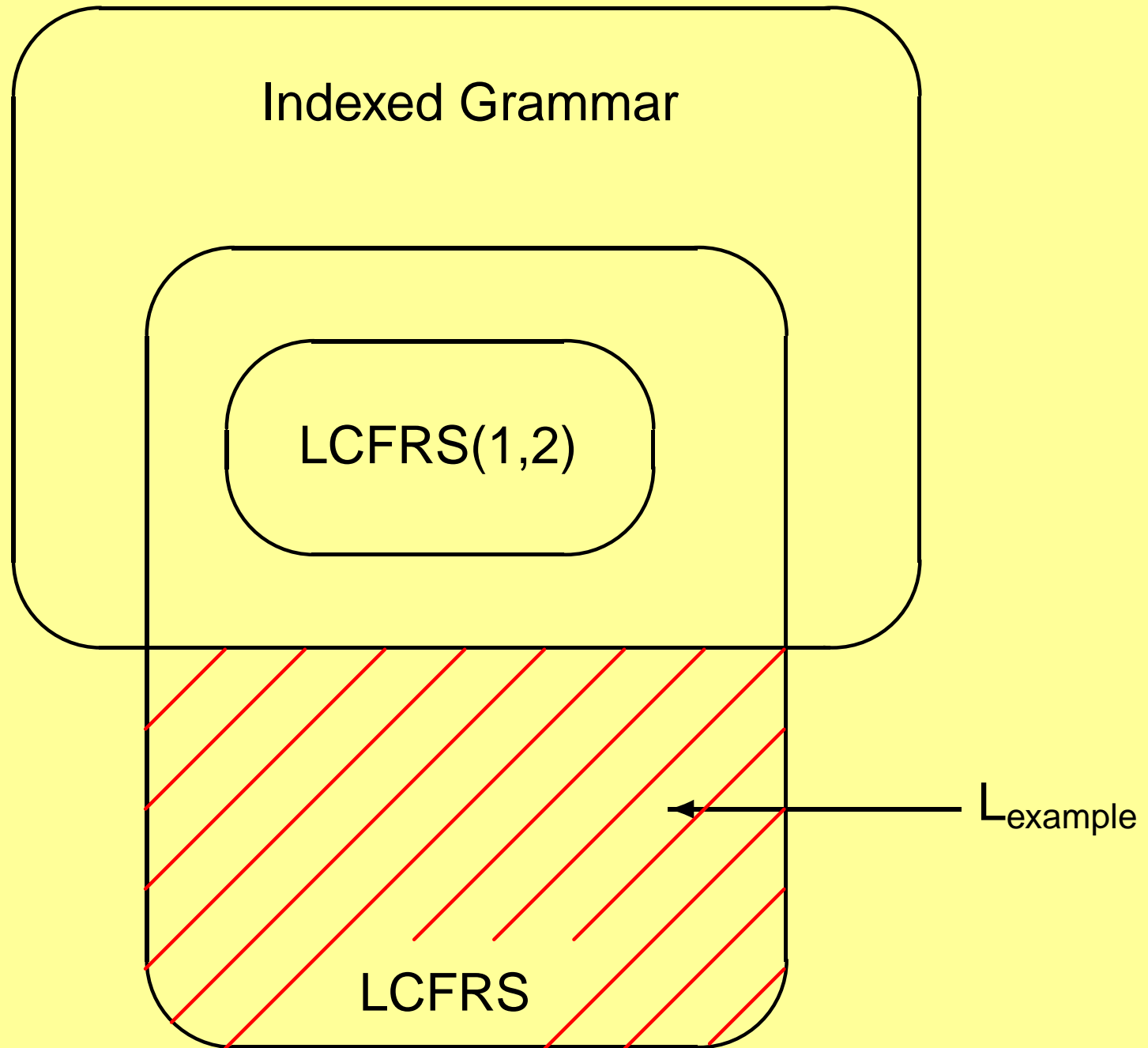
---

■ Excludes a **non-indexed, but LCFRS-string language** such as:

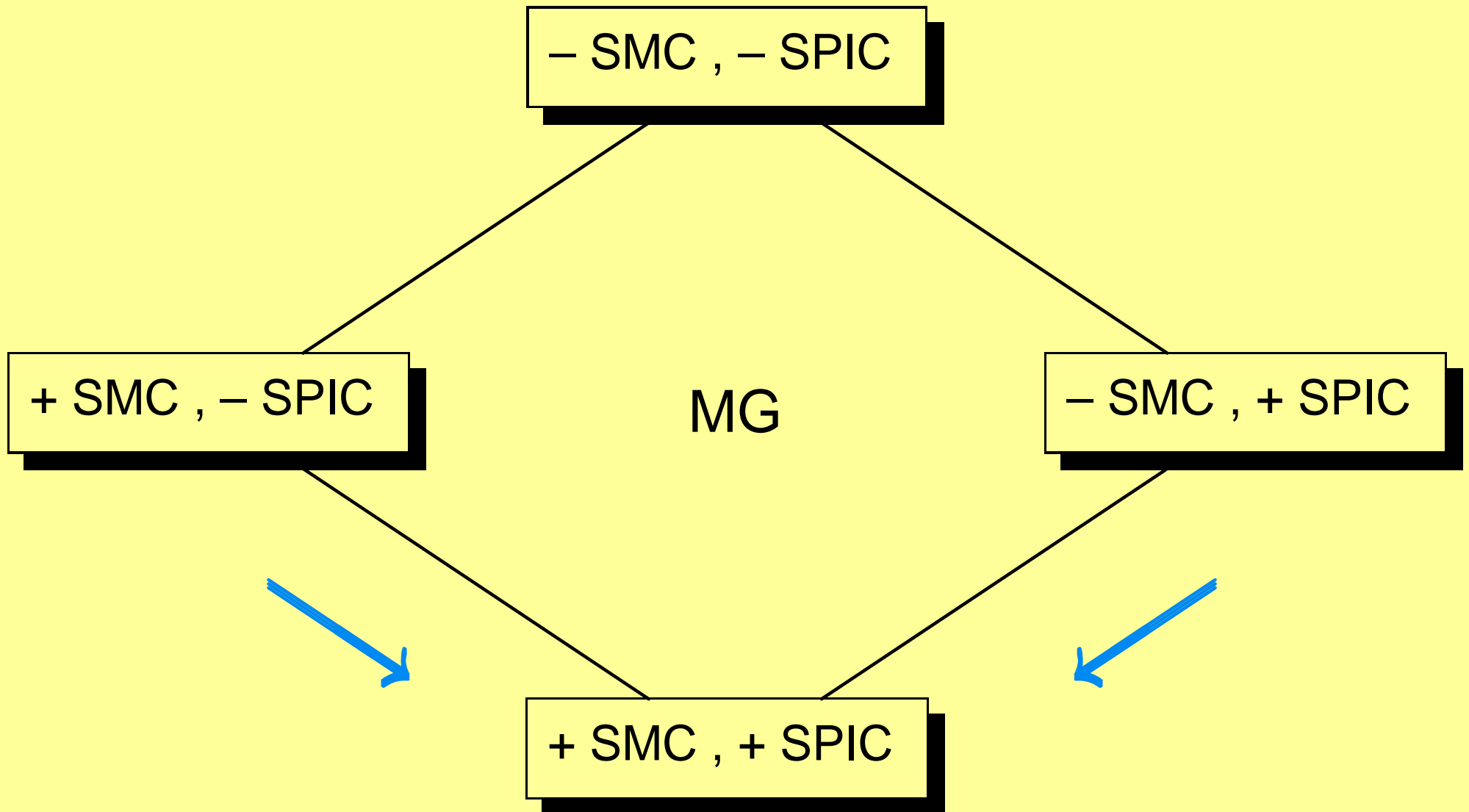
$$\left\{ w_1 \cdots w_n z_n w_n \cdots z_1 w_1 z_0 w_n^R \cdots w_1^R \mid w_i \in \{a, b\}^+, z_n \cdots z_0 \text{ Dyck word} \right\}$$

---

# LCFRS(1,2) — a restricted LCFRS-normal form

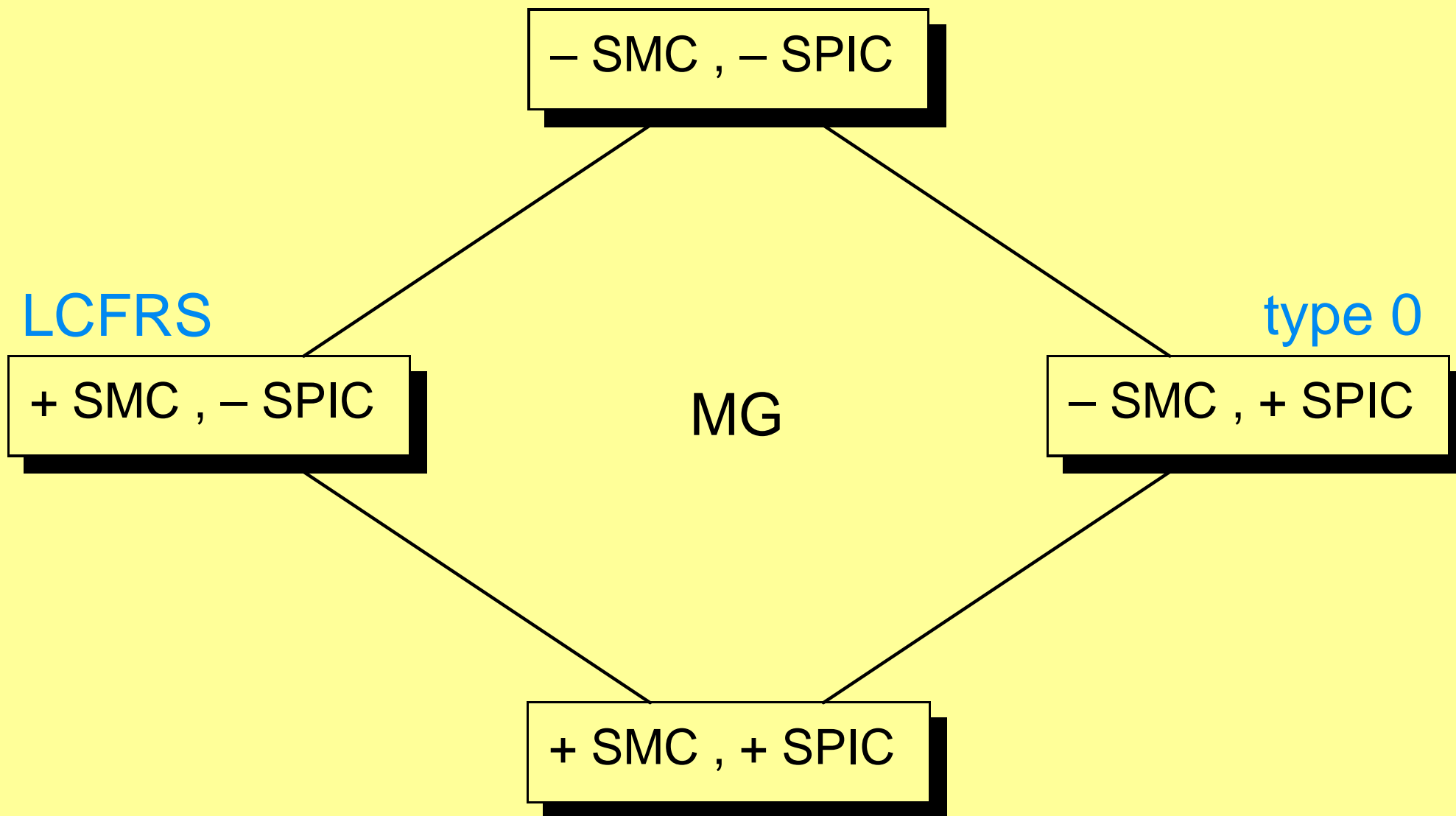


# SMC and SPIC — restricting the move-operator domain

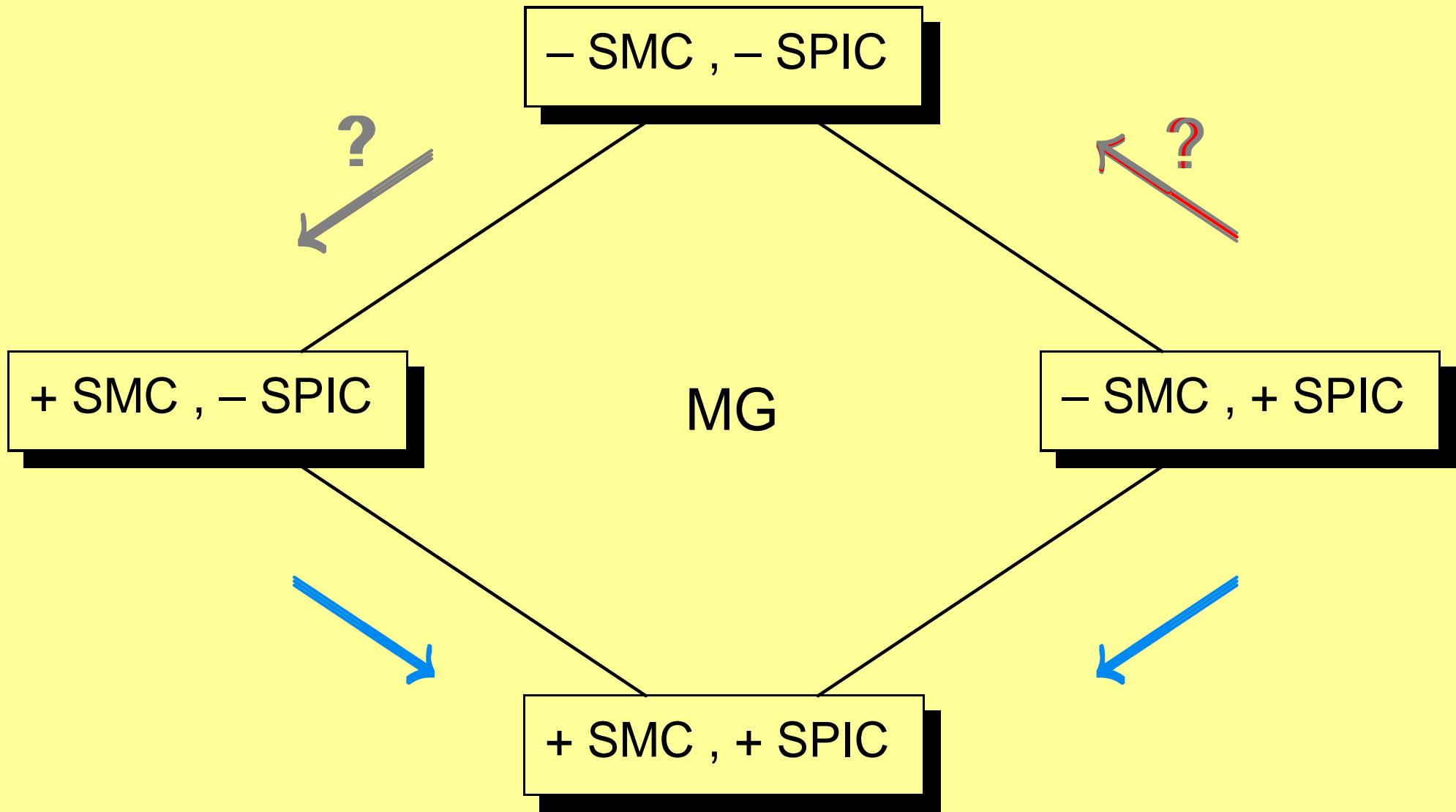


# SMC and SPIC — restricting the move-operator domain

MELL-proof-search (Salvati 2008)



# SMC and SPIC — restricting the move-operator domain



## A further extension — multiple wh-movement and the SMC

- A potential **objection** against MG(+SMC)'s: you **cannot deal with multiple wh-movement**. /\* example from Bulgarian \*/

<b>ko</b> <sub>j</sub>	<b>kogo</b> <sub>j</sub>	<b>kakvo</b> <sub>k</sub>	<b>t</b> <sub>j</sub>	e	pital	<b>t</b> <sub>j</sub>	<b>t</b> <sub>k</sub>
who	whom	what		AUX	ask		

- ◆ Recall the SMC-implementation in MGs: the number of competing licensee features triggering a movement is (finitely) bounded.
- **Answer**: we can, if we **implement** the **wh-cluster hypothesis** going back to Rudin (1988) such that we **introduce two new syntactic feature types and a corresponding operator**.

## A further extension — multiple wh-movement and the SMC

- *c(luster)-licensees*:  $\triangle x, \triangle y, \triangle z, \dots$   
*c(luster)-licensors*:  $\nabla x, \nabla y, \nabla z, \dots$

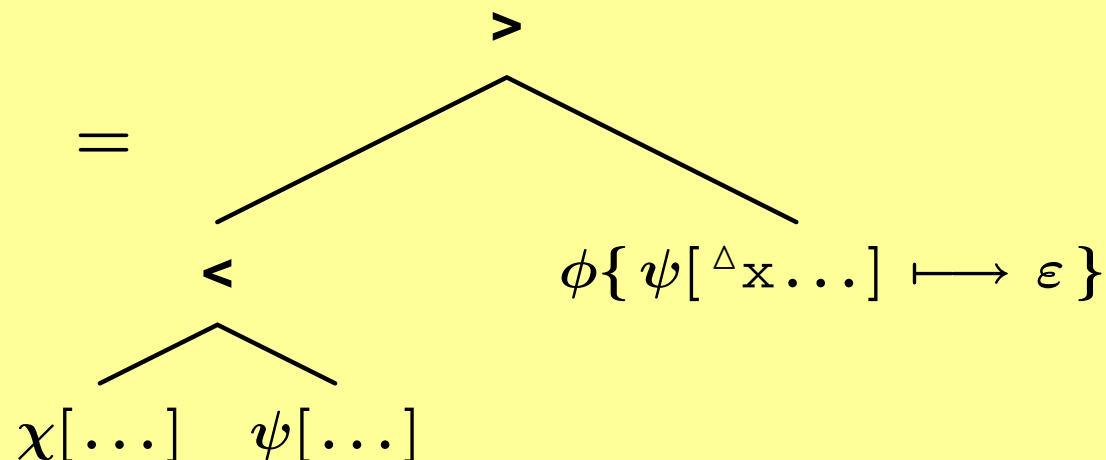
# Structure building functions

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

■  $\phi \in \text{Domain}(\text{cluster}) : \iff$

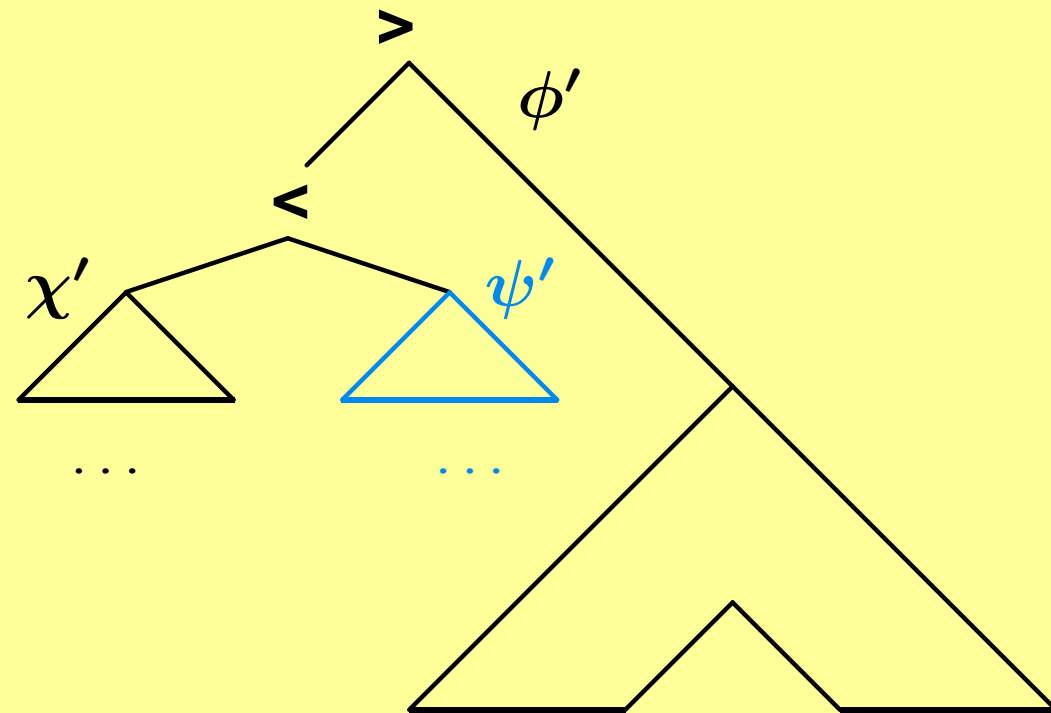
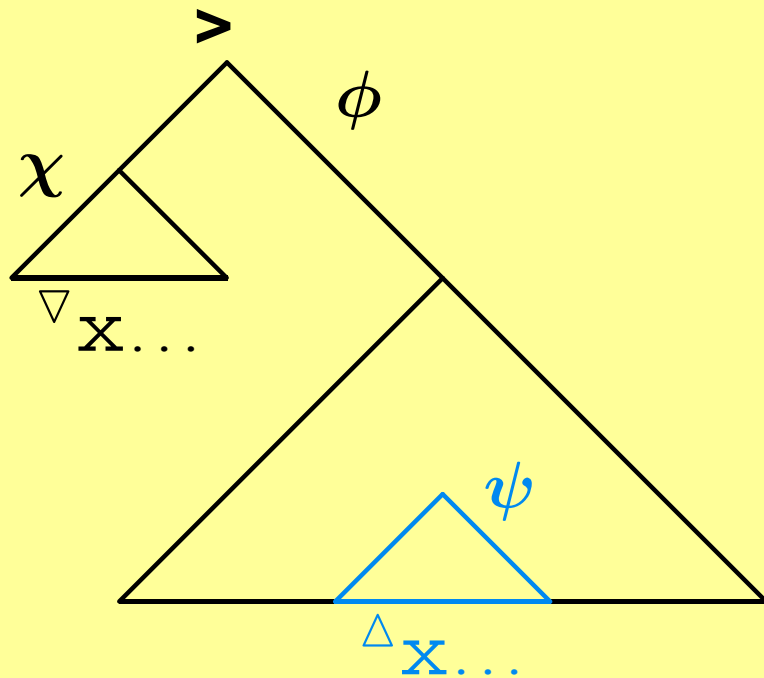
- The highest specifier  $\chi$  of  $\phi$  displays c-licensor  $\nabla_{\mathbf{x}}$
- there is a (unique [SMC]) maximal projection  $\psi$  within  $\phi$  that displays the corresponding c-licensee  $\triangle_{\mathbf{x}}$

■  $\text{cluster}(\phi) =$



# Structure building functions

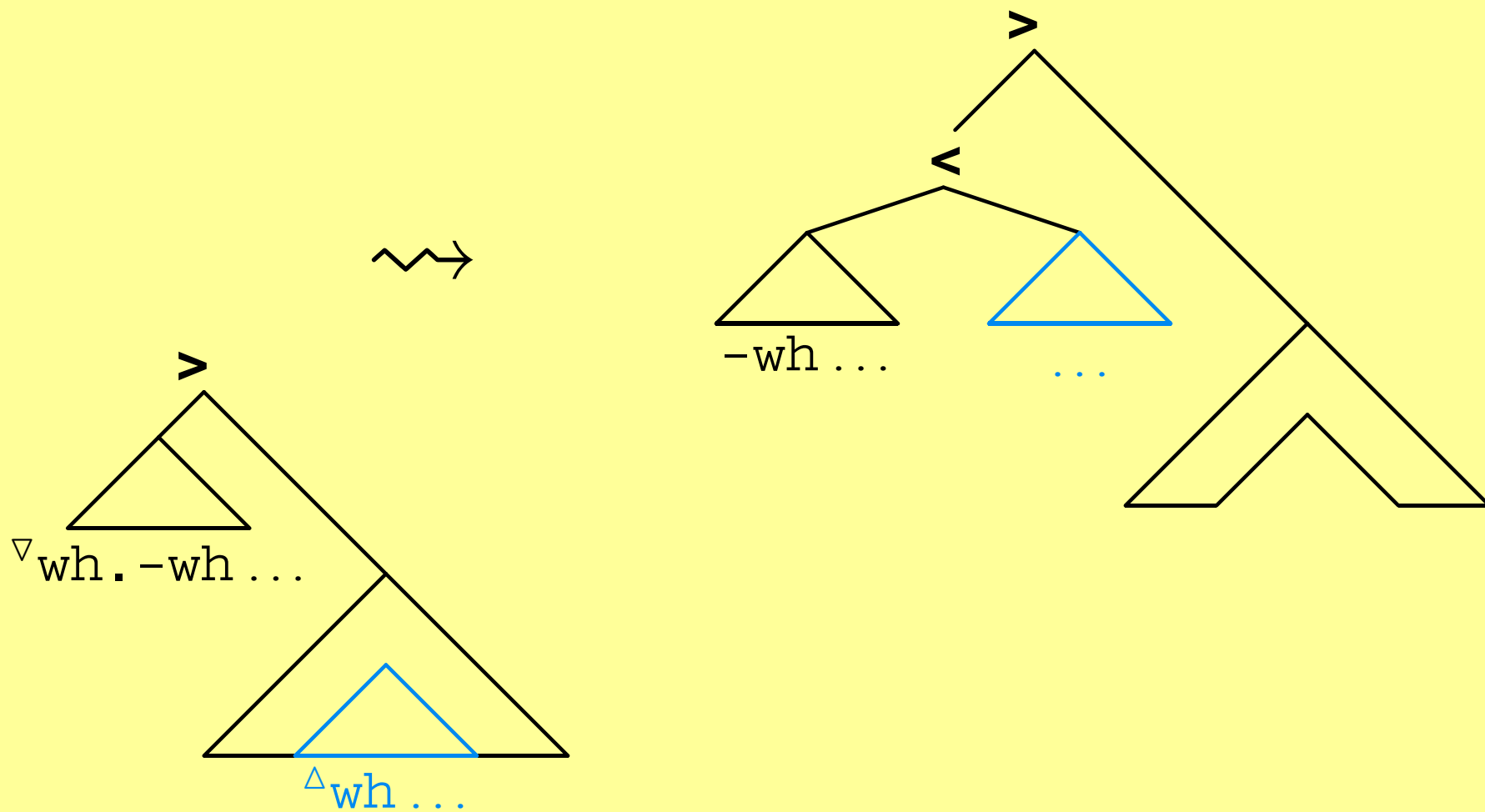
cluster : Trees  $\xrightarrow{\text{part}}$  2Trees



## A further extension — multiple wh-movement and the SMC

- In order to outline the general case, we next sketch derivations for wh-clustering with two wh-phrases: crucially **exactly one  $-wh$  licensee is necessary** for deriving a well-formed cluster, and **no more than one  $\triangle wh$  is displayed at any derivation step**.

# Wh-clustering, $n = 2$ , crucial step 1



# Wh-clustering, $n = 2$ , crucial step 2

