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

Head movement constraint (HMC) (Stabler 1997)

- The implementation of

  head movement in MGs is in accordance with the HMC

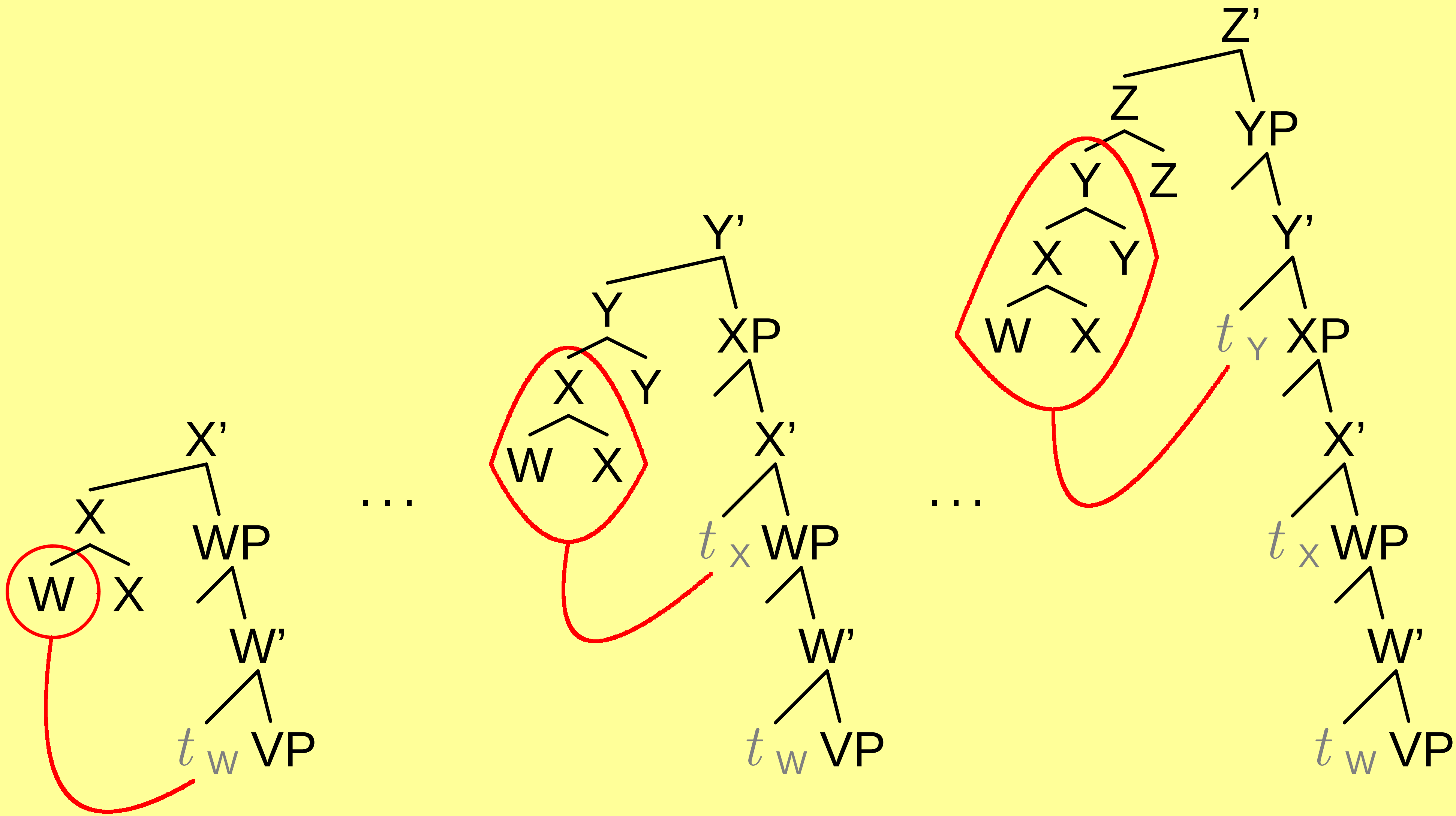
  - demanding

    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.

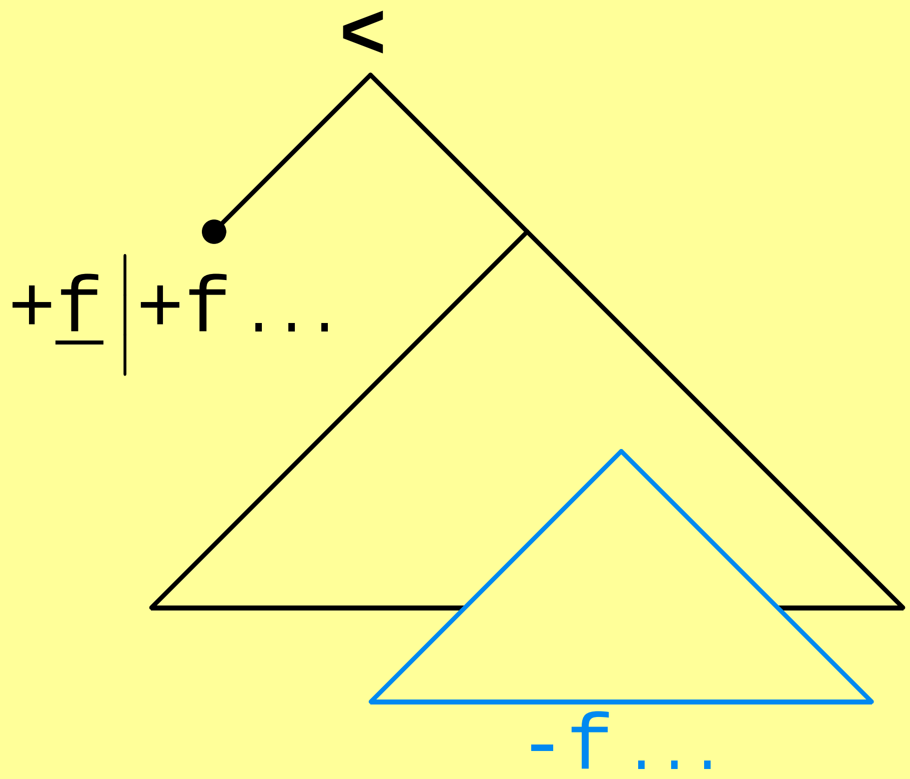
Successive cyclic left head adjunction



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

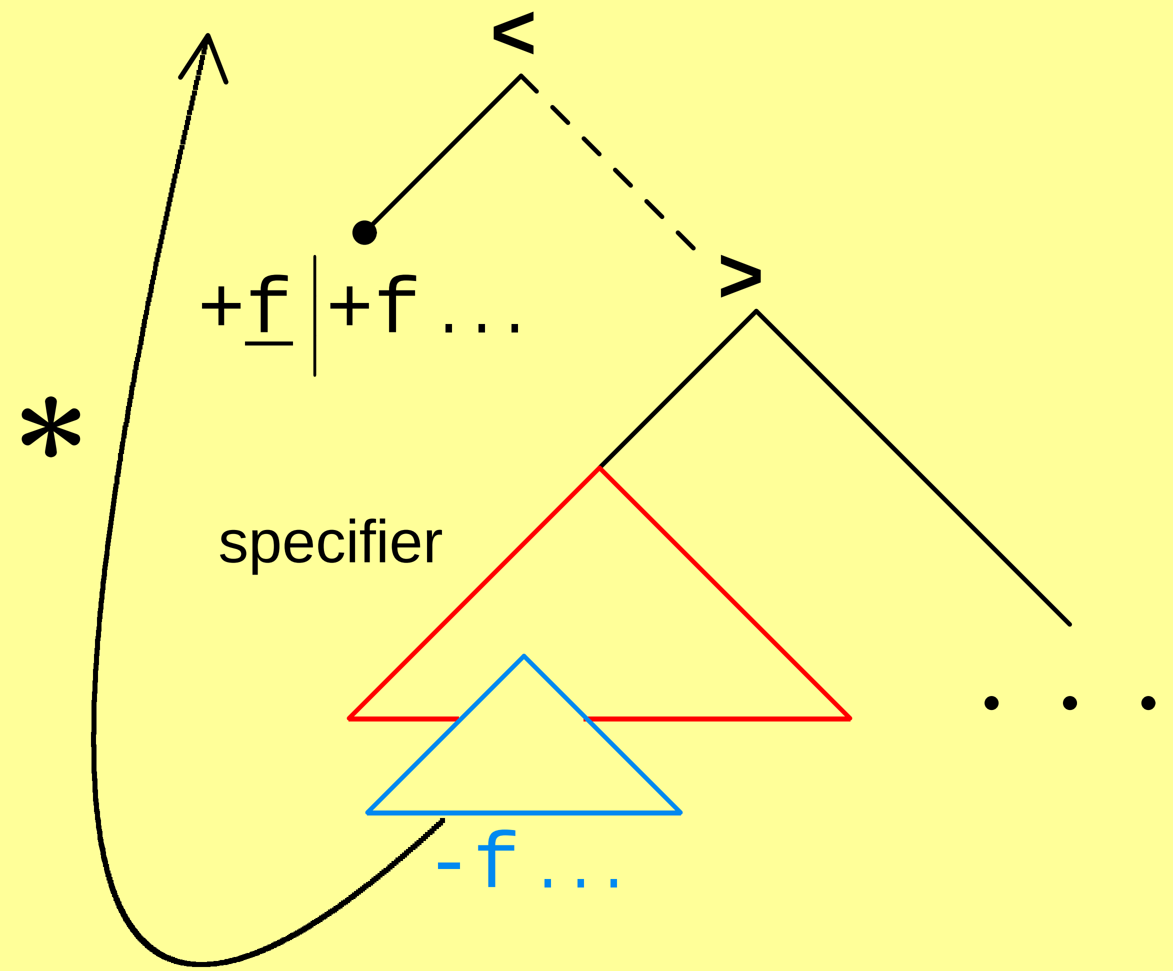
- The number of competing licensee features triggering a movement is (finitely) bounded by n.

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

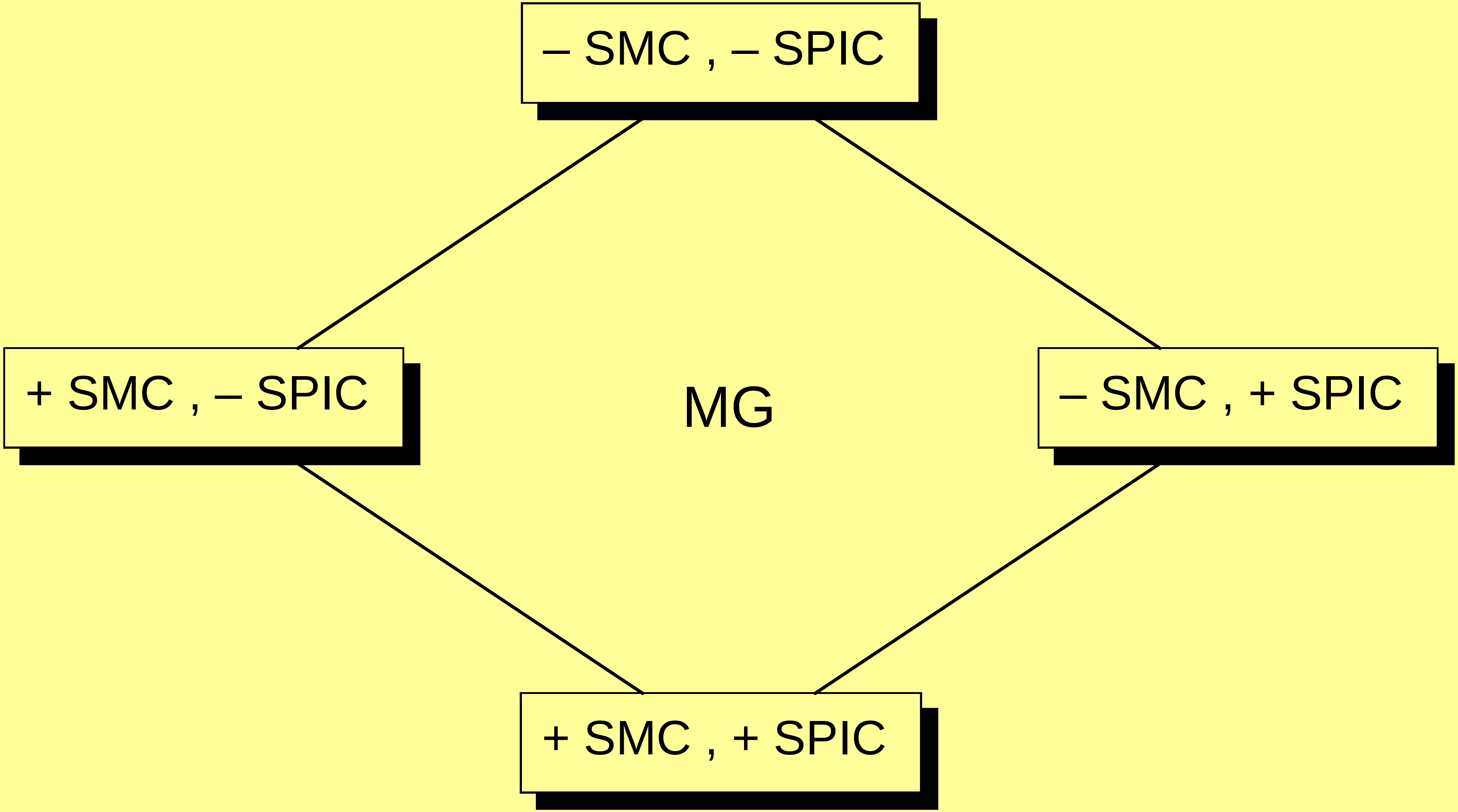


Specifier island condition (SPIC) (Stabler 1999)

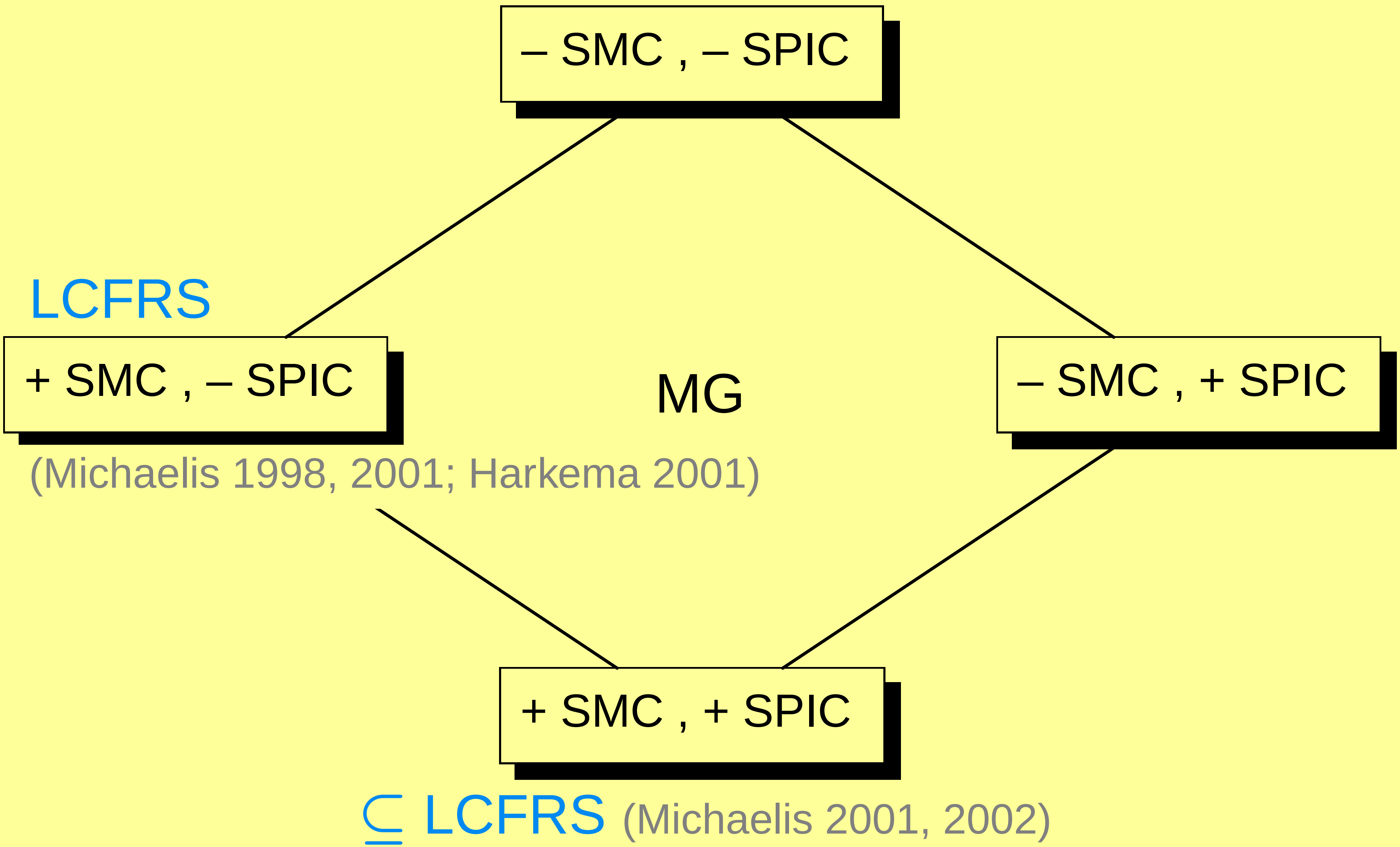
- Proper “extraction” from specifiers is blocked.



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 $= \langle$ Features , Lexicon , Ω , c $\rangle$ be an MG

A minimal expression $\tau \in$ Closure(G) is relevant $:\Longleftrightarrow$

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

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

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

A minimal expression $\tau \in \mathrm{Closure}(\mathrm{G})$ 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 expression $\tau \in \mathrm{Closure}(\mathrm{G})$ involved in creating a complete expression in G due to the SMC.

A finite partition of set of relevant expressions

Basic idea: consider relevant $\tau \in$ Closure(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 set of relevant expressions

Basic idea : consider relevant $\tau \in$ Closure(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 = |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 $\tau \in$ Closure(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.

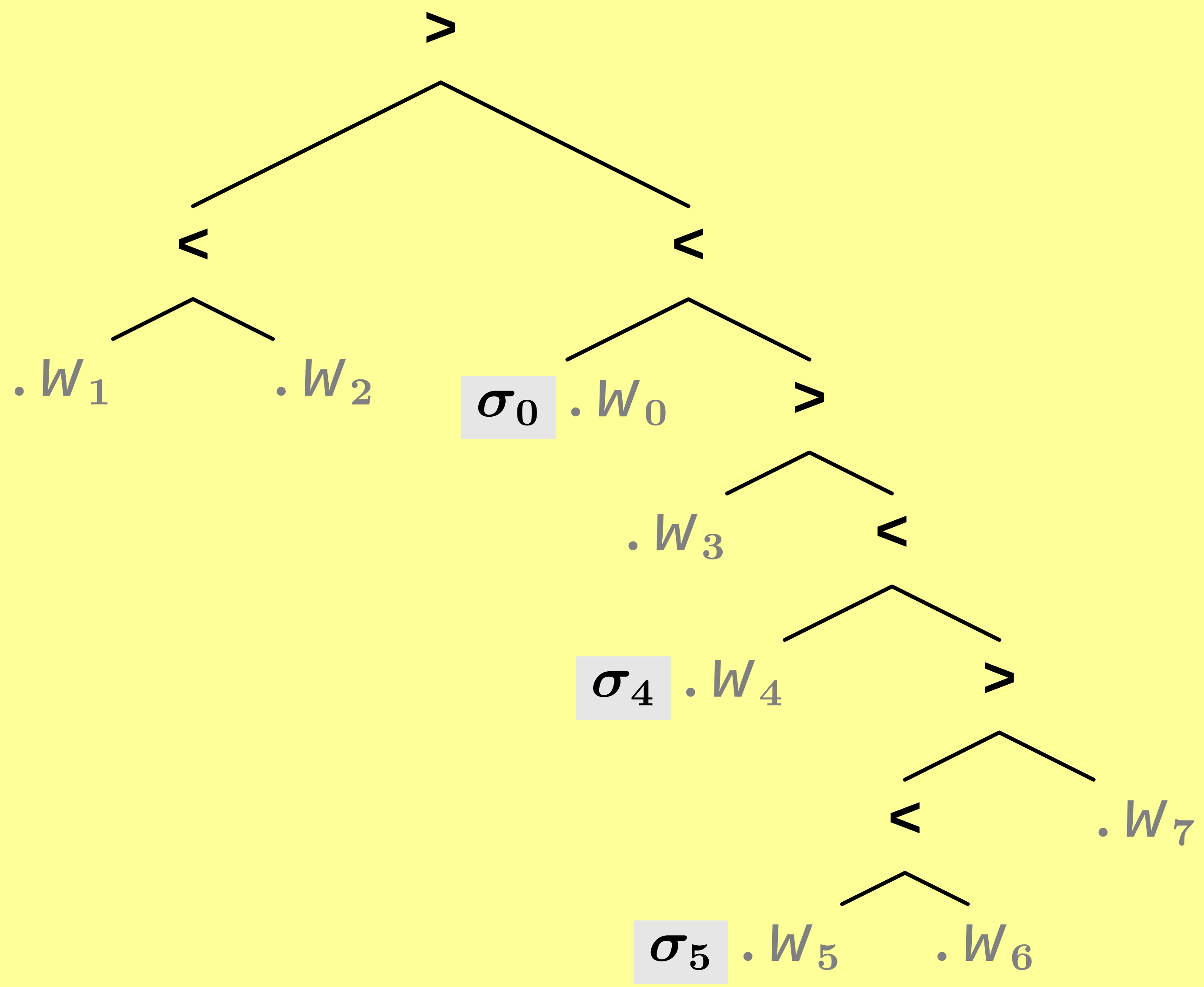
$\rightsquigarrow$ only finitely many equivalence classes

Relevance:
The resulting tuple has at most m+1 components, m = |Licensees|.

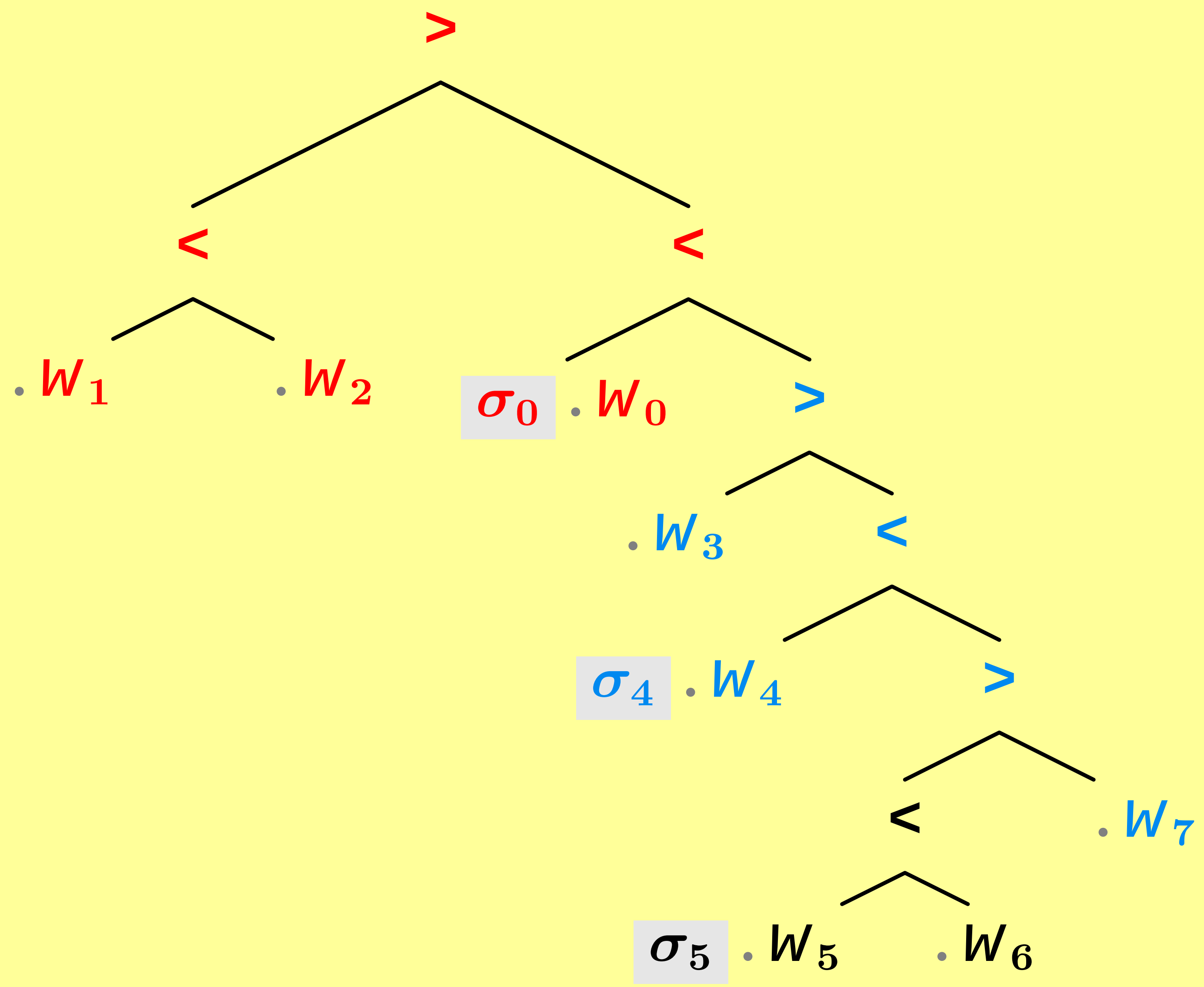
Structure building by cancellation of features:
Each tuple component is the suffix of the syntactic prefix of the label of a lexical item.

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

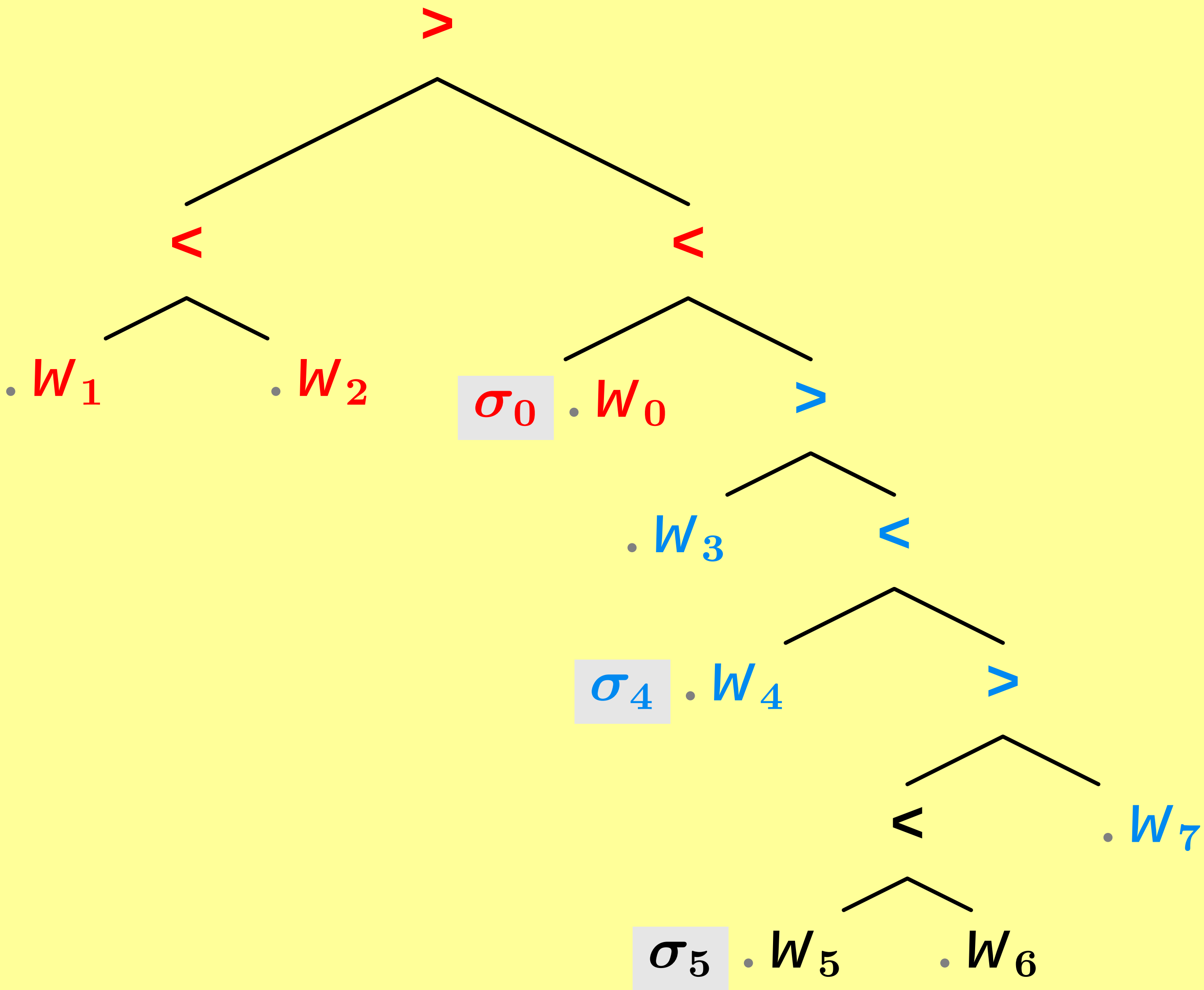
Reducing an MG(+SMC,-/+SPIC)



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

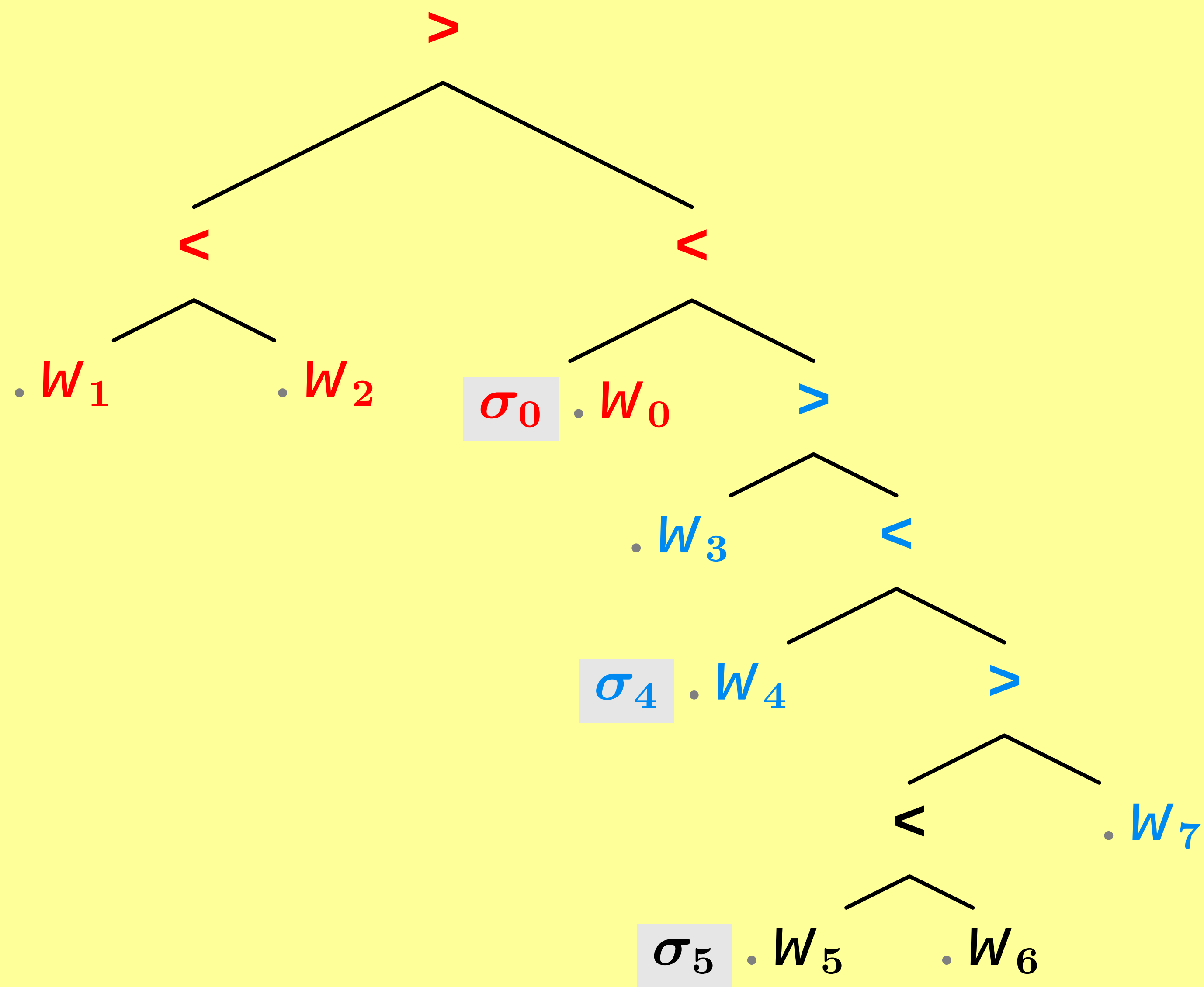


Reducing an MG(+SMC,-/+SPIC)



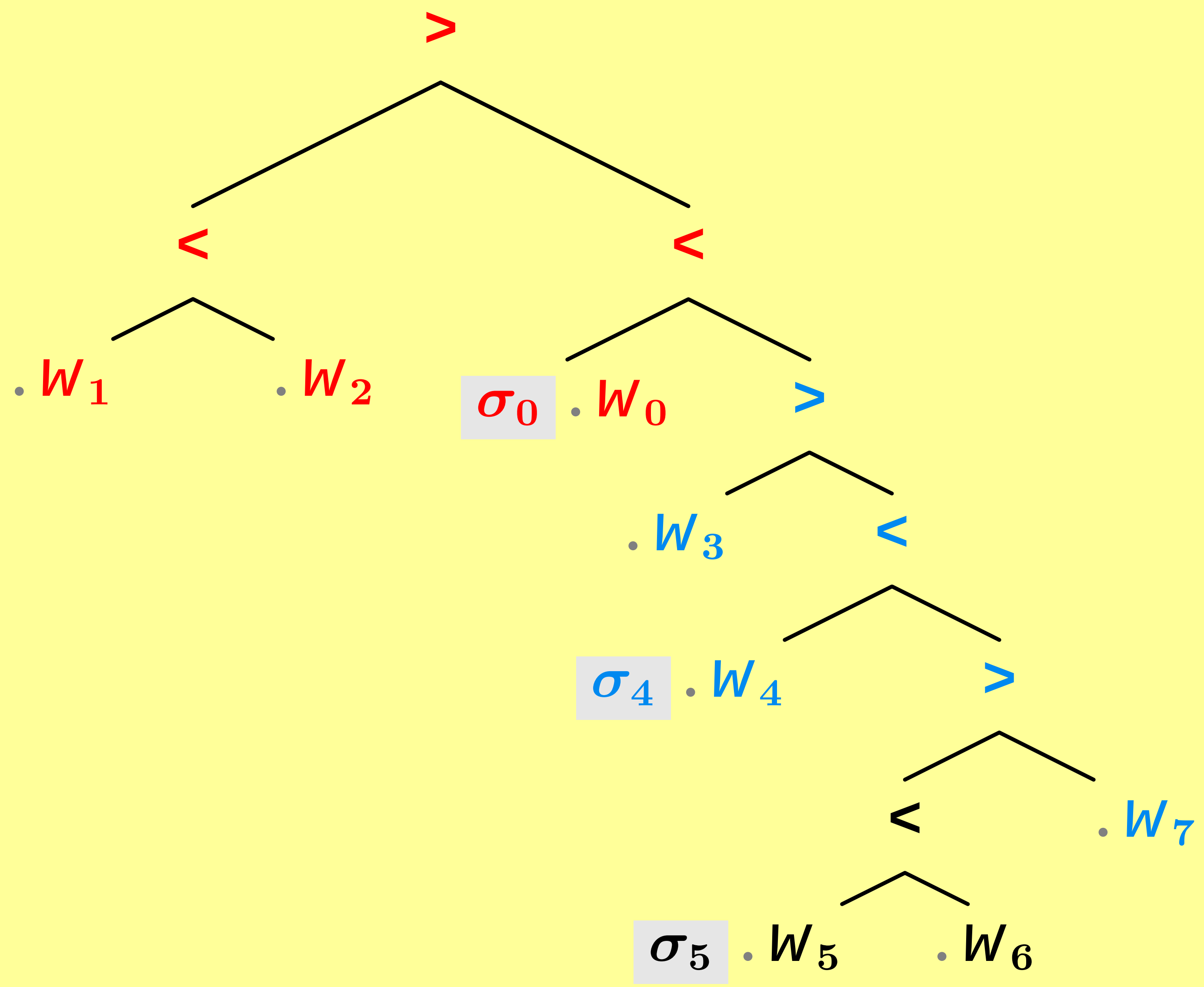
$$\left\langle\ \sigma_0 \cdot w_1 w_2 w_0\ ,\ \sigma_4 \cdot w_3 w_4 w_7\ ,\ \sigma_5 \cdot w_5 w_6\ \right\rangle$$

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



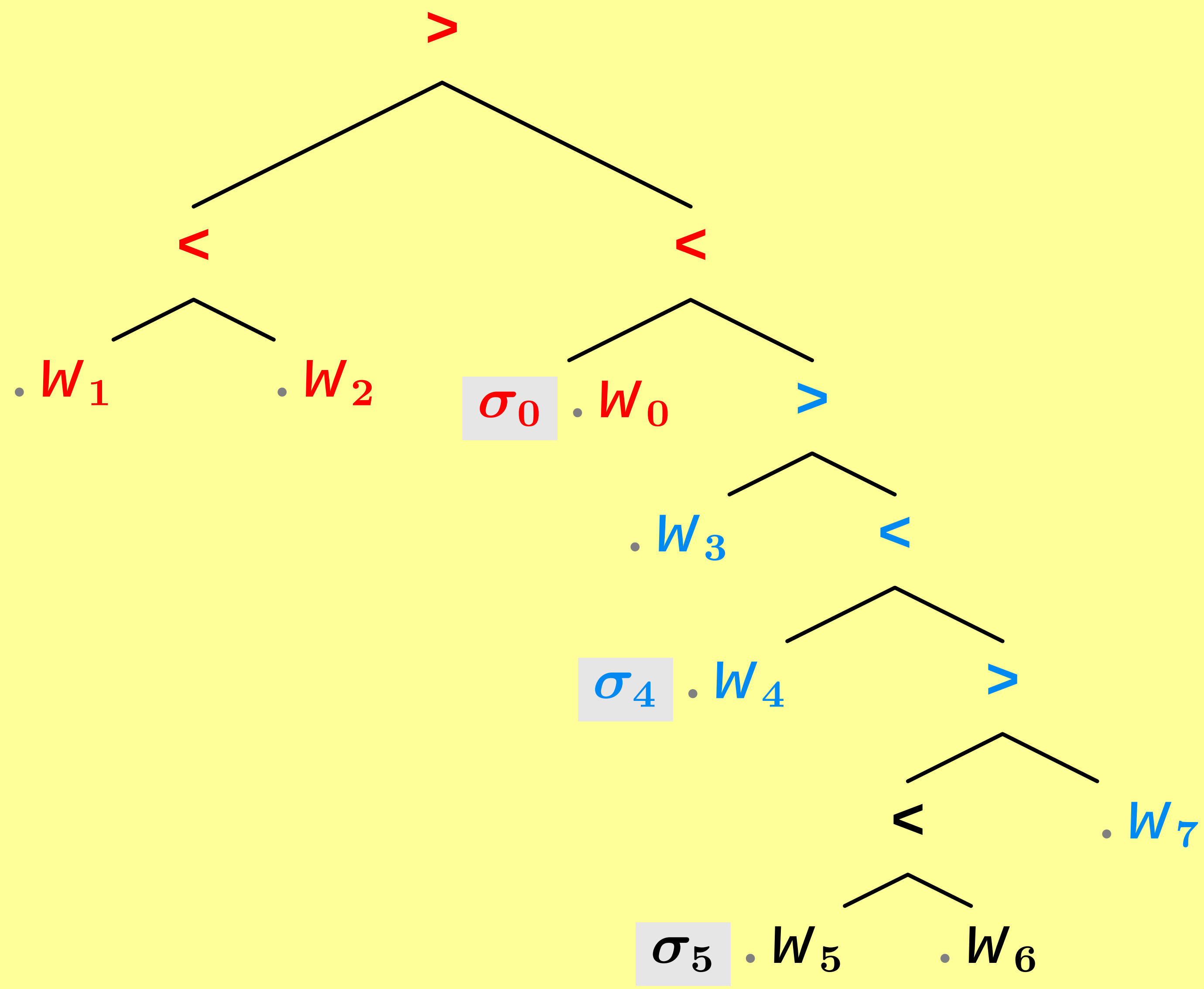
$$\langle \sigma_0 \quad , \quad \sigma_4 \quad , \quad \sigma_5 \rangle$$

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



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

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



$$\langle \sigma_0 \; , \; \sigma_4 \; , \; \sigma_5 \rangle \Longrightarrow \langle w_1 w_2 w_0 \; , \; w_3 w_4 w_7 \; , \; w_5 w_6 \rangle$$

MG-example 2

(α_0) =t.c.*that*

(α_1) =t.+wh.c.∅

(α_2) =ṽ.+k.t.∅

(α_3) =v.=d.ṽ.∅

(α_4) =d.+k.v.*eat*

(α_5) v.*laugh*

(α_6) =n.d.-k.*the*

(α_7) =n.d.-k.-wh.*which*

(α_8) n.*king*

(α_9) n.*pie*

MG-example 2

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

n.*pie*

=n.d.-k.-wh.*which* ⟨=n.d.-k.-wh.*which*, ::⟩

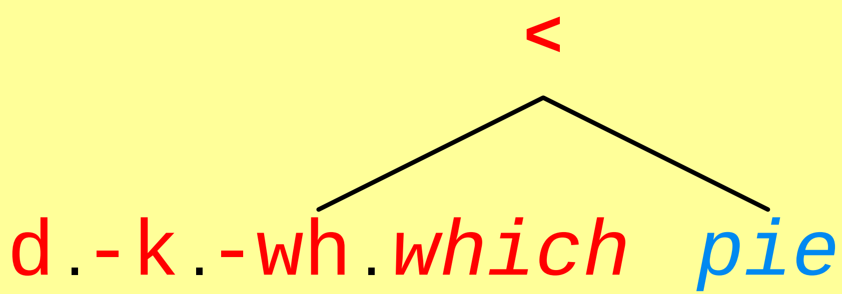
n.*pie* ⟨n.*pie*, ::⟩

MG-example 2

:: ≙ simple , : ≙ complex

=n.d.-k.-wh.*which* ⟨ =n.d.-k.-wh.*which*, :: ⟩

n.*pie* ⟨ n.*pie*, :: ⟩

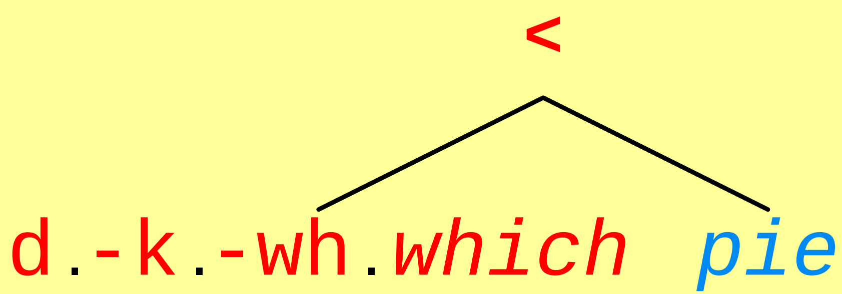


MG-example 2

:: ≙ simple , : ≙ complex

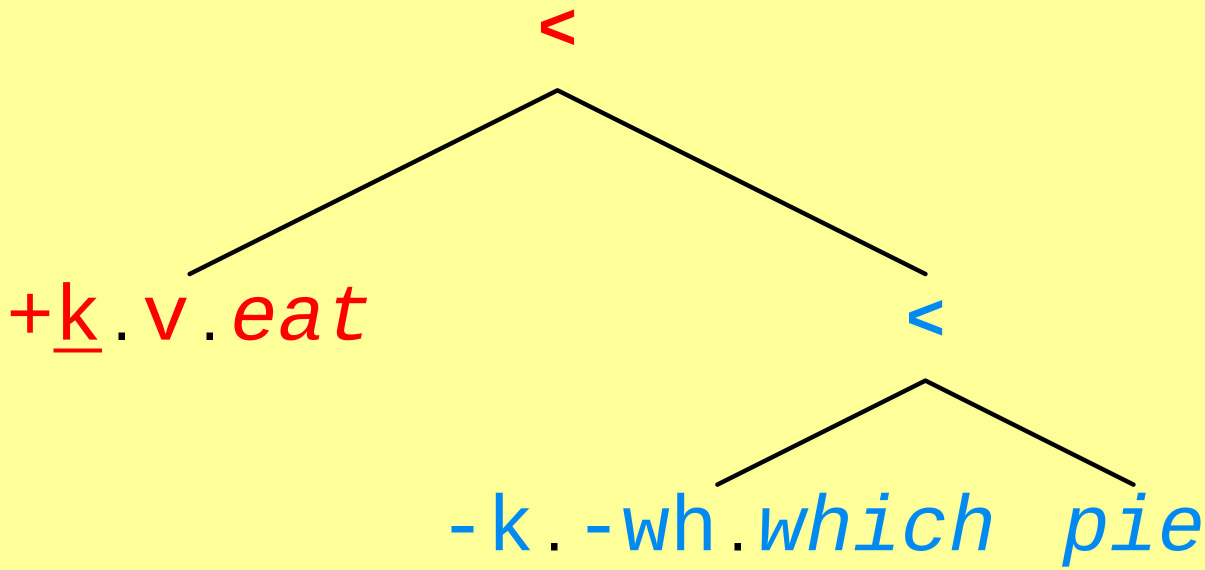
=n.d.-k.-wh.*which* ⟨ =n.d.-k.-wh.*which* , :: ⟩

n.*pie* ⟨ n.*pie* , :: ⟩



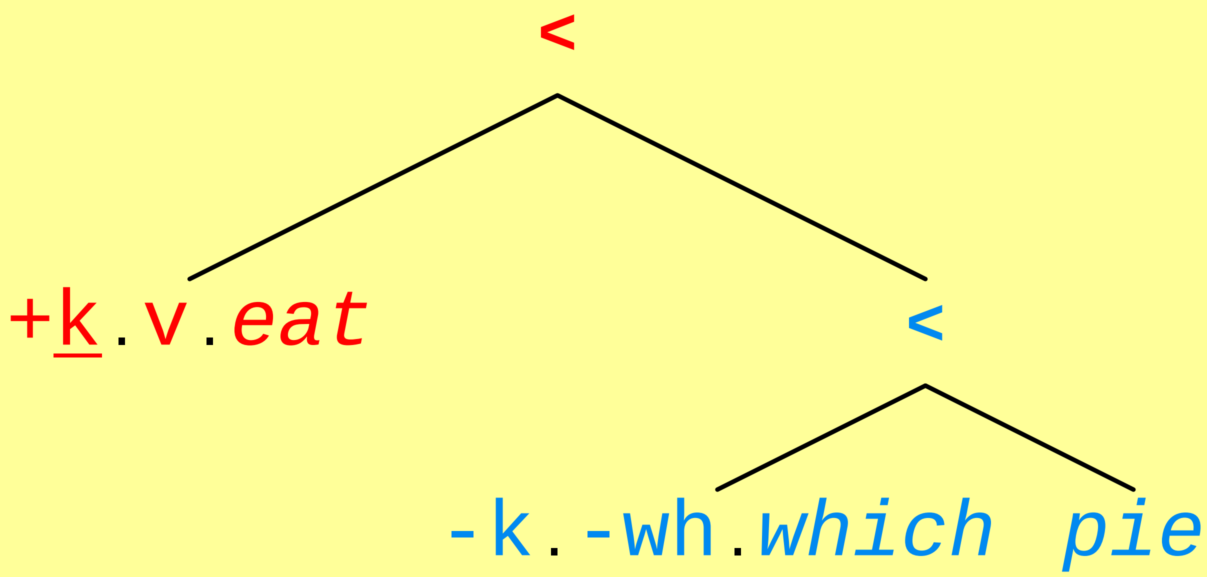
⟨ d.-k.-wh.*which* *pie* , : ⟩

MG-example 2



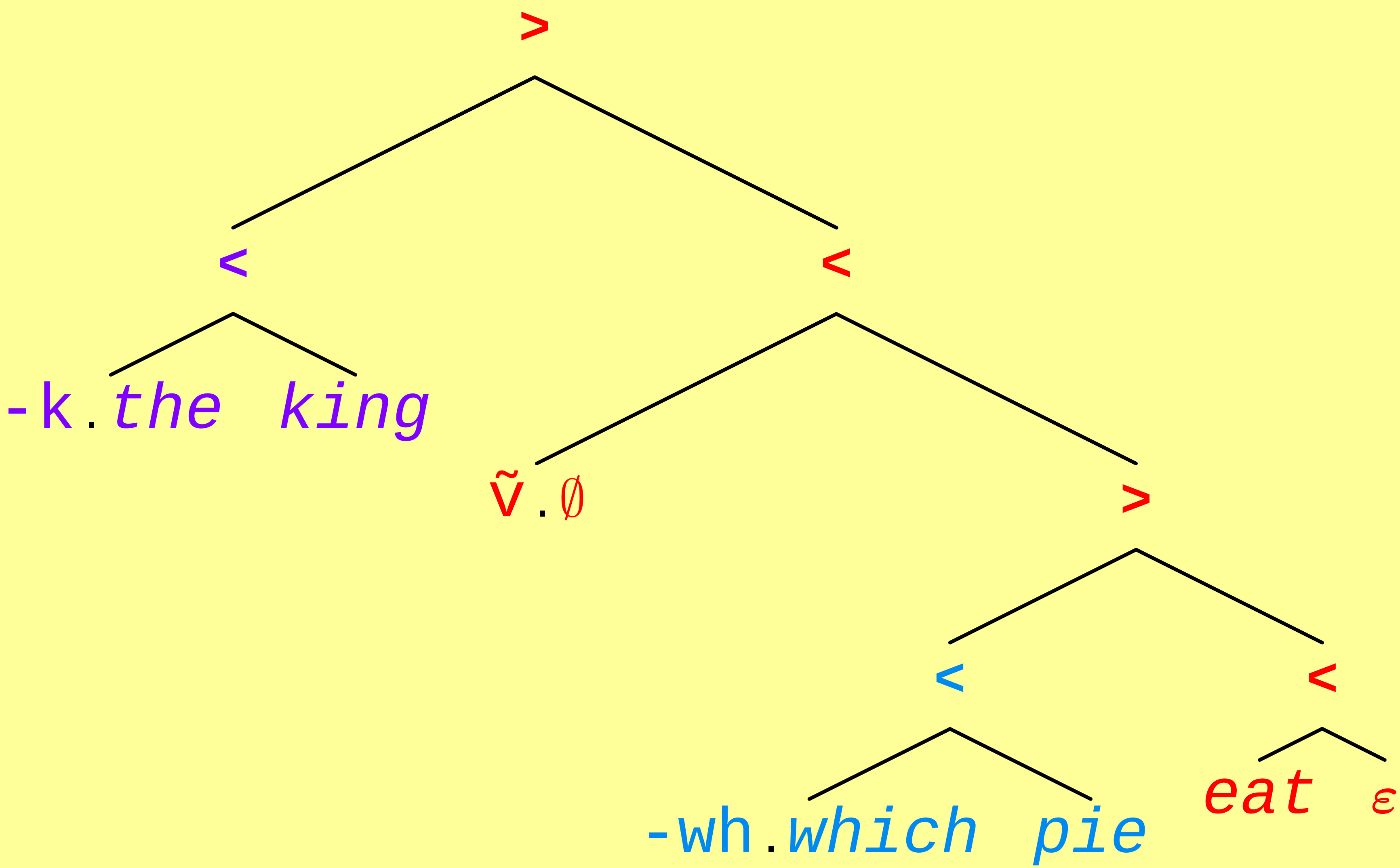
MG-example 2

:: ≙ simple , : ≙ complex



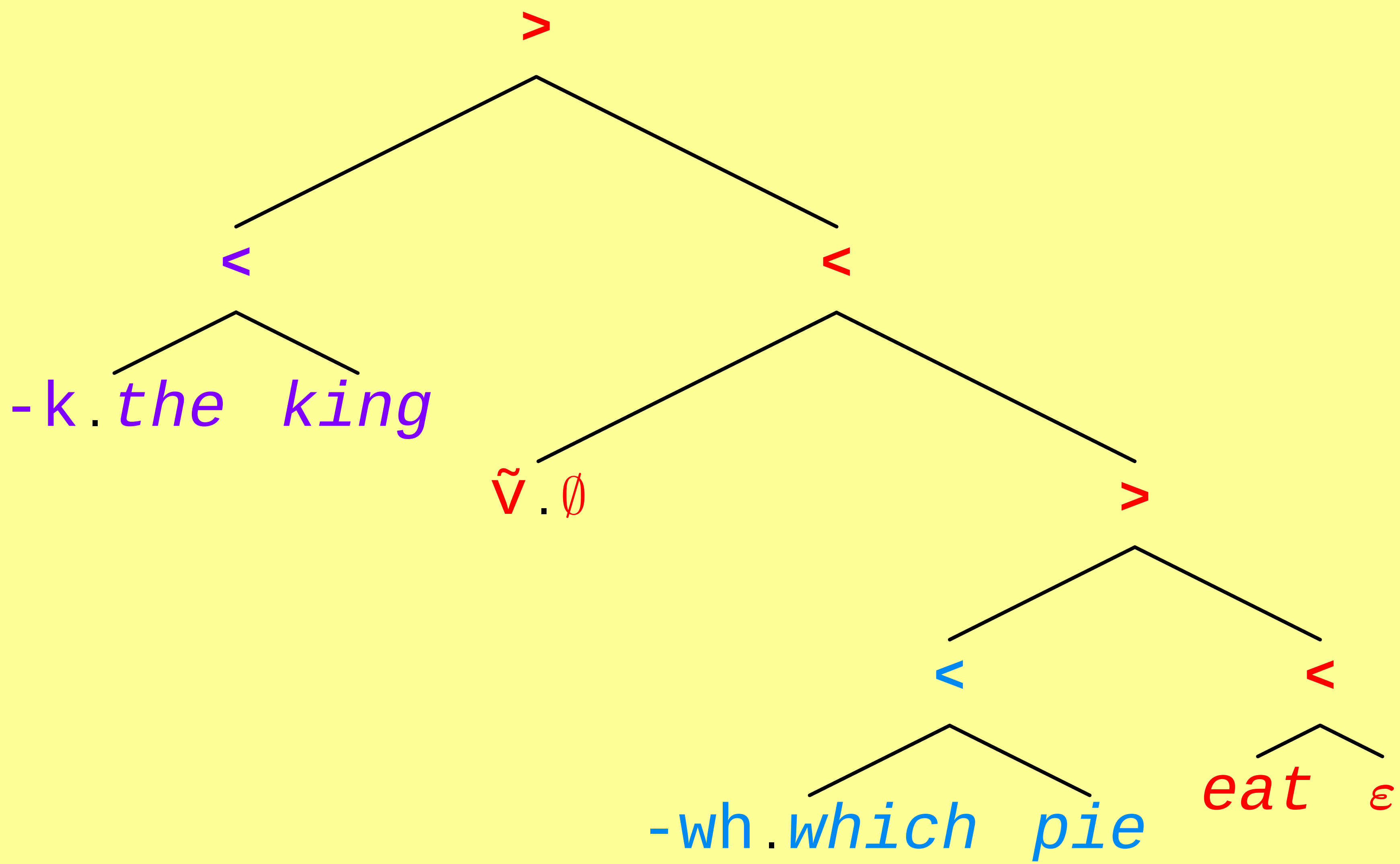
⟨ +<u>k</u>.v.*eat* , -k.-wh.*which pie* , : ⟩

MG-example 2



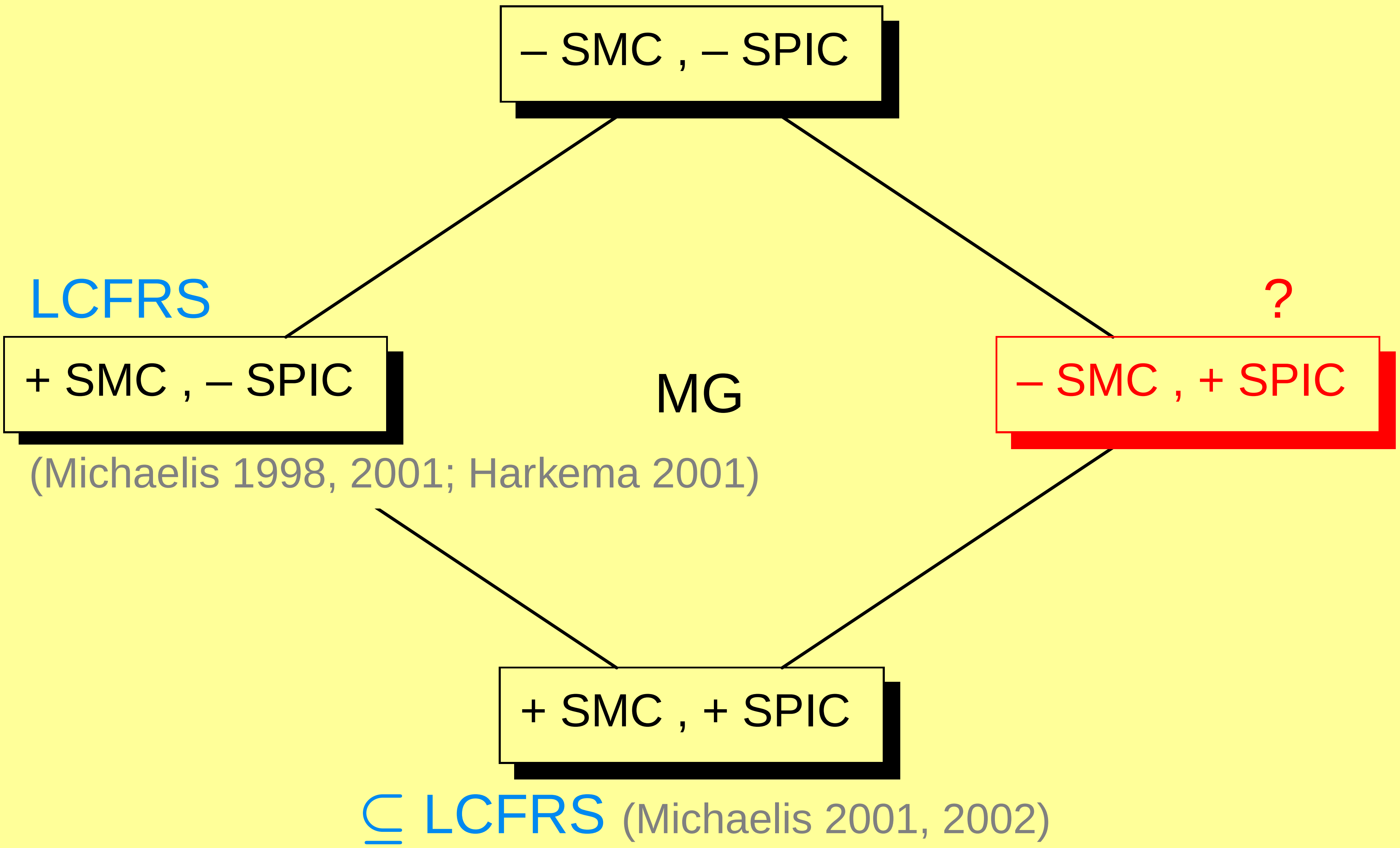
MG-example 2

:: ≙ simple , : ≙ complex



⟨ ṽ.*eat*, -wh.*which pie*, -k.*the king*, : ⟩

SMC and SPIC — restricting the move-operator domain

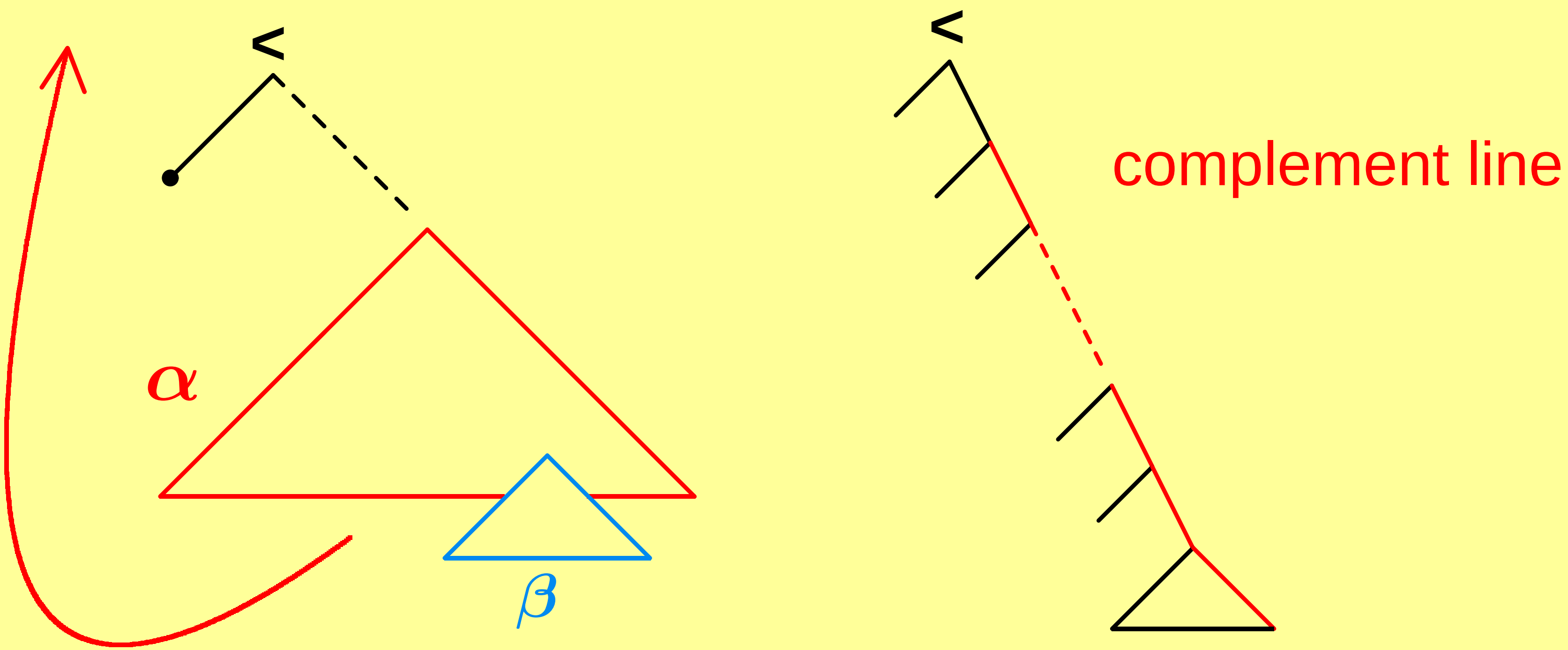


- Gärtner & Michaelis 2005 shows that MG(–SMC,+SPIC)s allow derivation of non-mildly context-sensitive languages.

- 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:

- Because of the SPIC, movement of a constituent α into a specifier position freezes every proper subconstituent β within α.

- Without the SMC, therefore, the complement line of a tree can technically be used as two independent counters, or, as a queue.



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

$$\left\{ a^{2^n} \mid n \geq 0 \right\} = \left\{ a\,,\,aa\,,\,aaaa\,,\,aaaaaaaa\,,\,\ldots \right\}$$

1 2 4 8 ...

```
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`

`=w.x.-l`

"initialize"

---

end "outer" cycle "appropriately:"
check licensee `-m`

start new "outer" cycle:
introduce new licensee `-m`

"reintroduce" and "double"
the just checked licensee `-l`

`=x.+m.y.-m`

`=y.+l.z.-l`

`=z.y.-l`

`=z.x.-l`

"outer" cycle

"inner" cycle

---

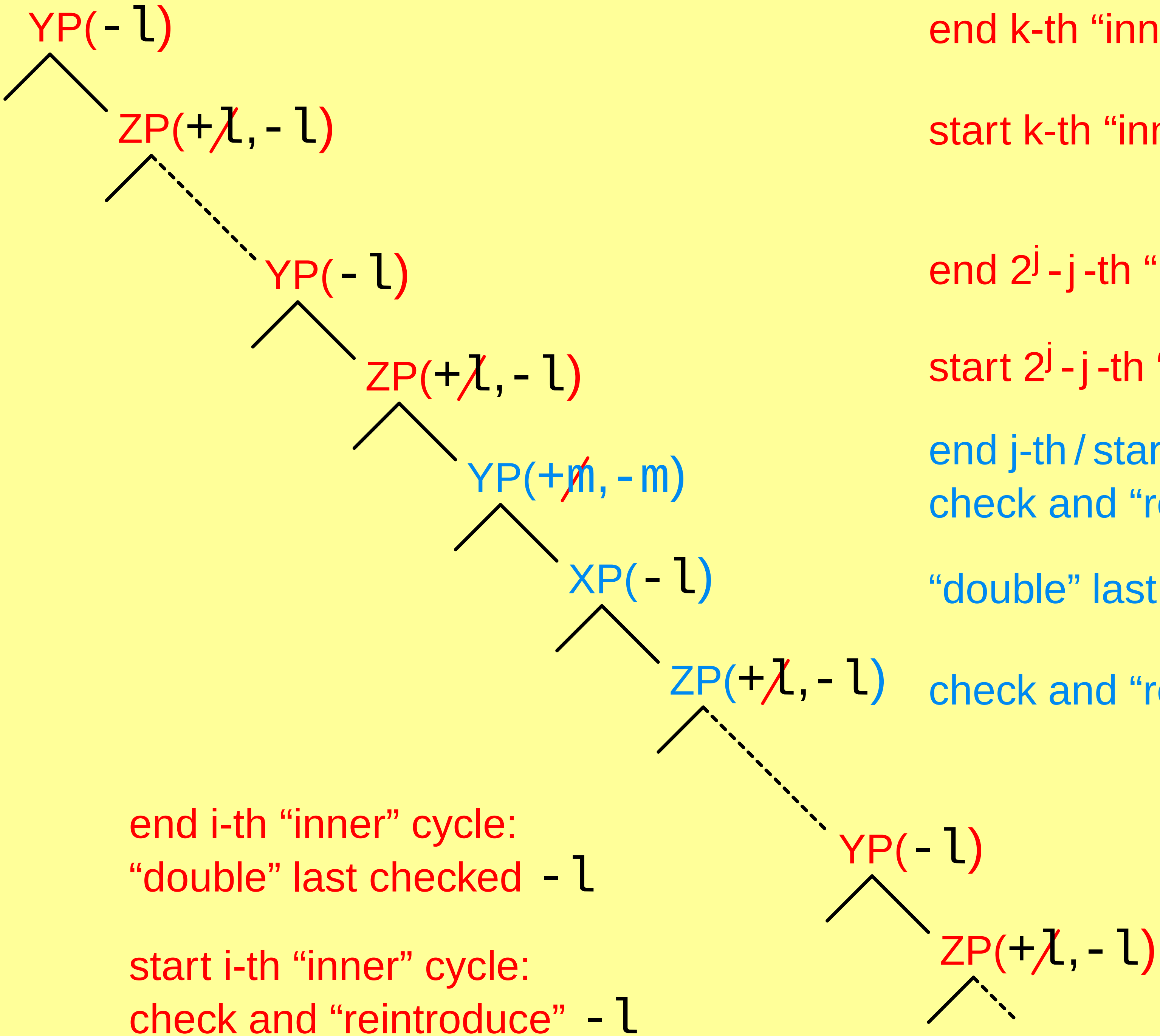
leave final cycle "appropriately:"
check licensee `-m`

check successively licensee `-l`,
each time introducing an *a*

`=x.+m.c`

`=c.+l.c.`*a*

"finalize"



end k-th “inner” cycle

start k-th “inner” cycle

end 2^j - j -th “inner” cycle

start 2^j - j -th “inner” cycle

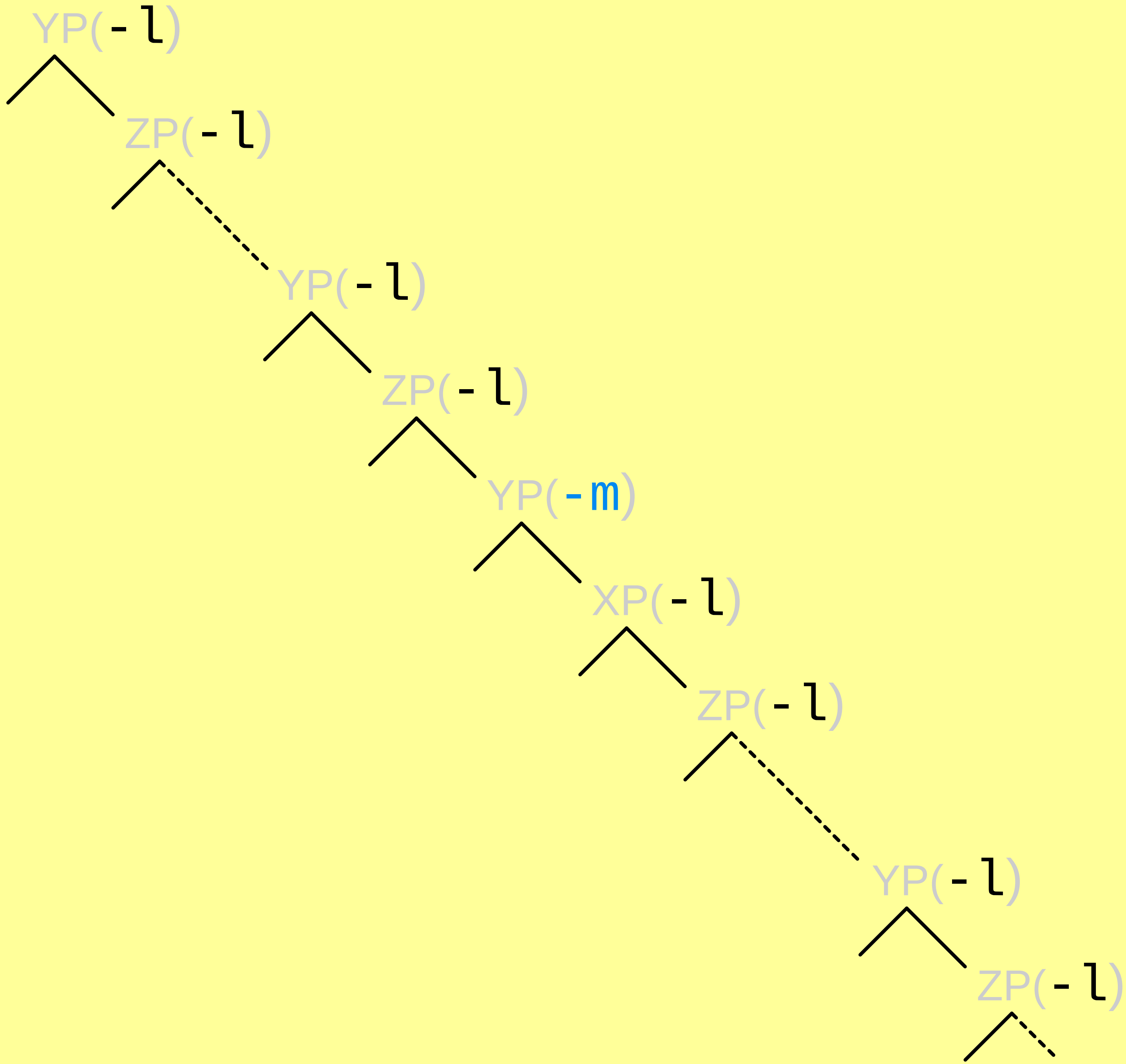
end j-th / start j+1-th “outer” cycle:
check and “reintroduce” -m

“double” last checked -l

check and “reintroduce” -l

end i-th “inner” cycle:
“double” last checked -l

start i-th “inner” cycle:
check and “reintroduce” -l

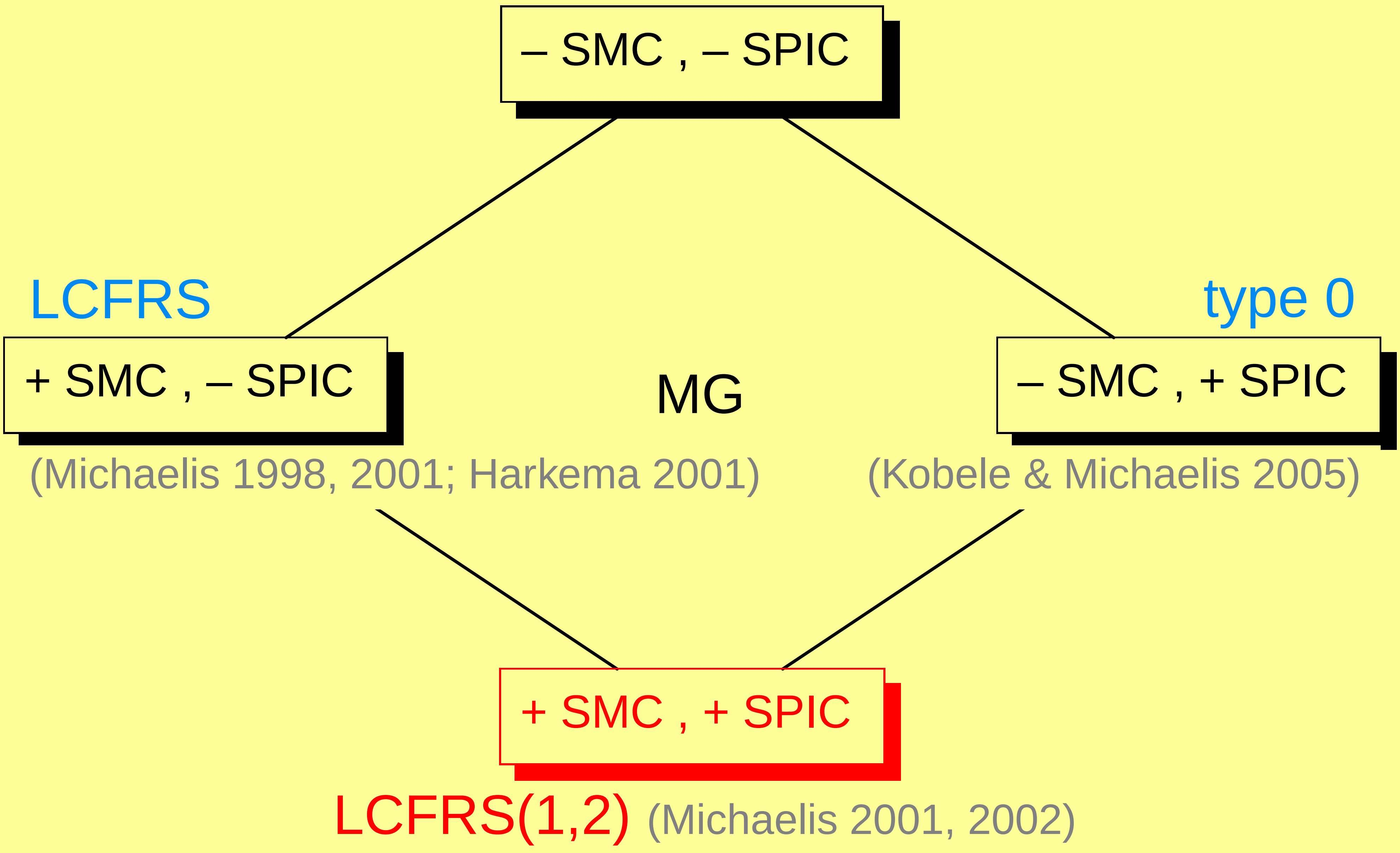

YP(-l)
ZP(-l)
YP(-l)
ZP(-l)
YP(-m)
XP(-l)
ZP(-l)
YP(-l)
ZP(-l)

- Starting the "outer" cycle, the currently derived tree shows 2^n successively embedded complements on the complement line each with an unchecked instance of `-l`, 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 `-l` 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 `-l` 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 $A \longrightarrow f(B)$ or $A \longrightarrow f(B, C)$,
- if $A \longrightarrow f(B, C)$, nonterminal B derives only simple terminal strings.

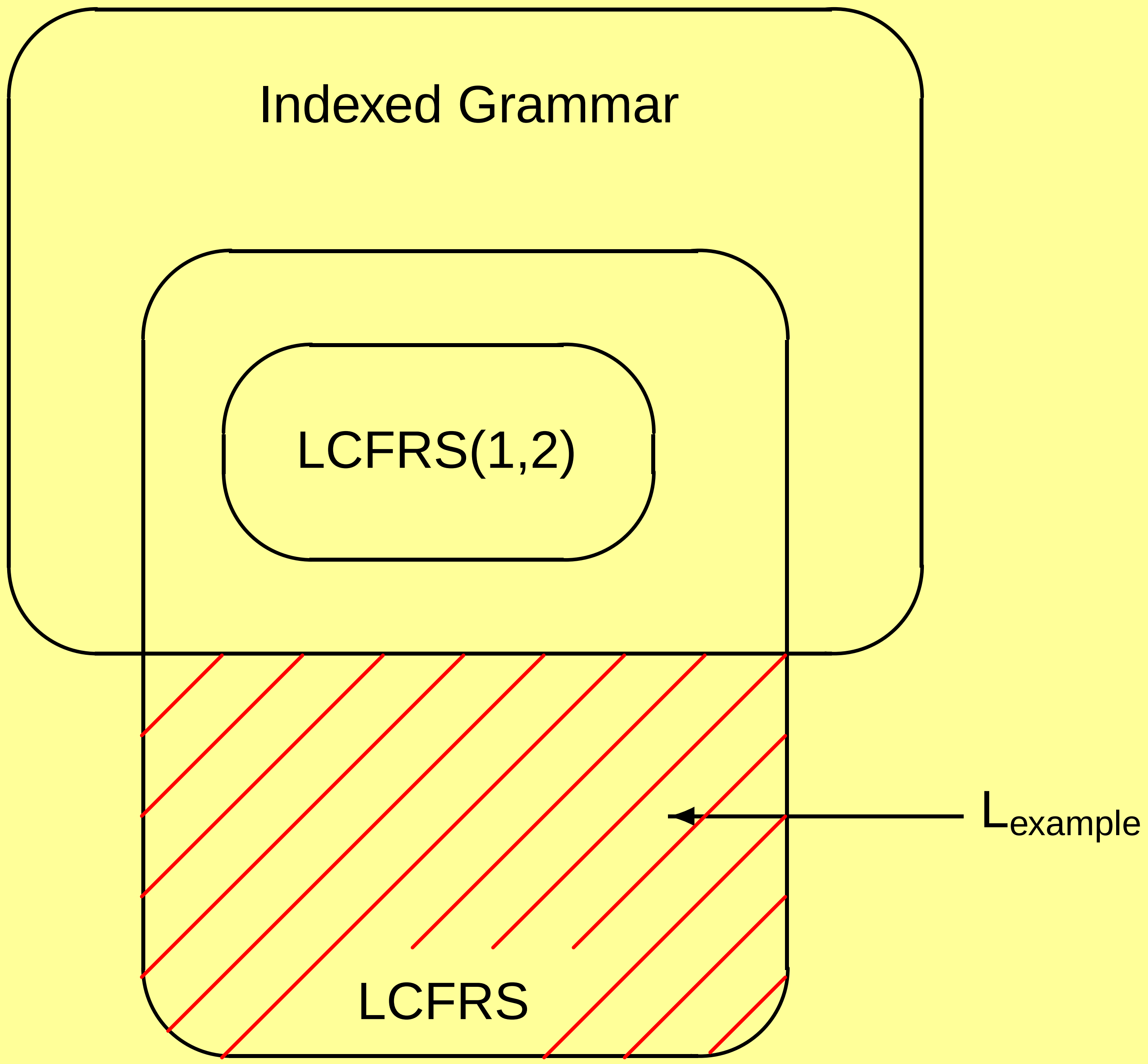
LCFRS(1,2) — a restricted LCFRS-normal form

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

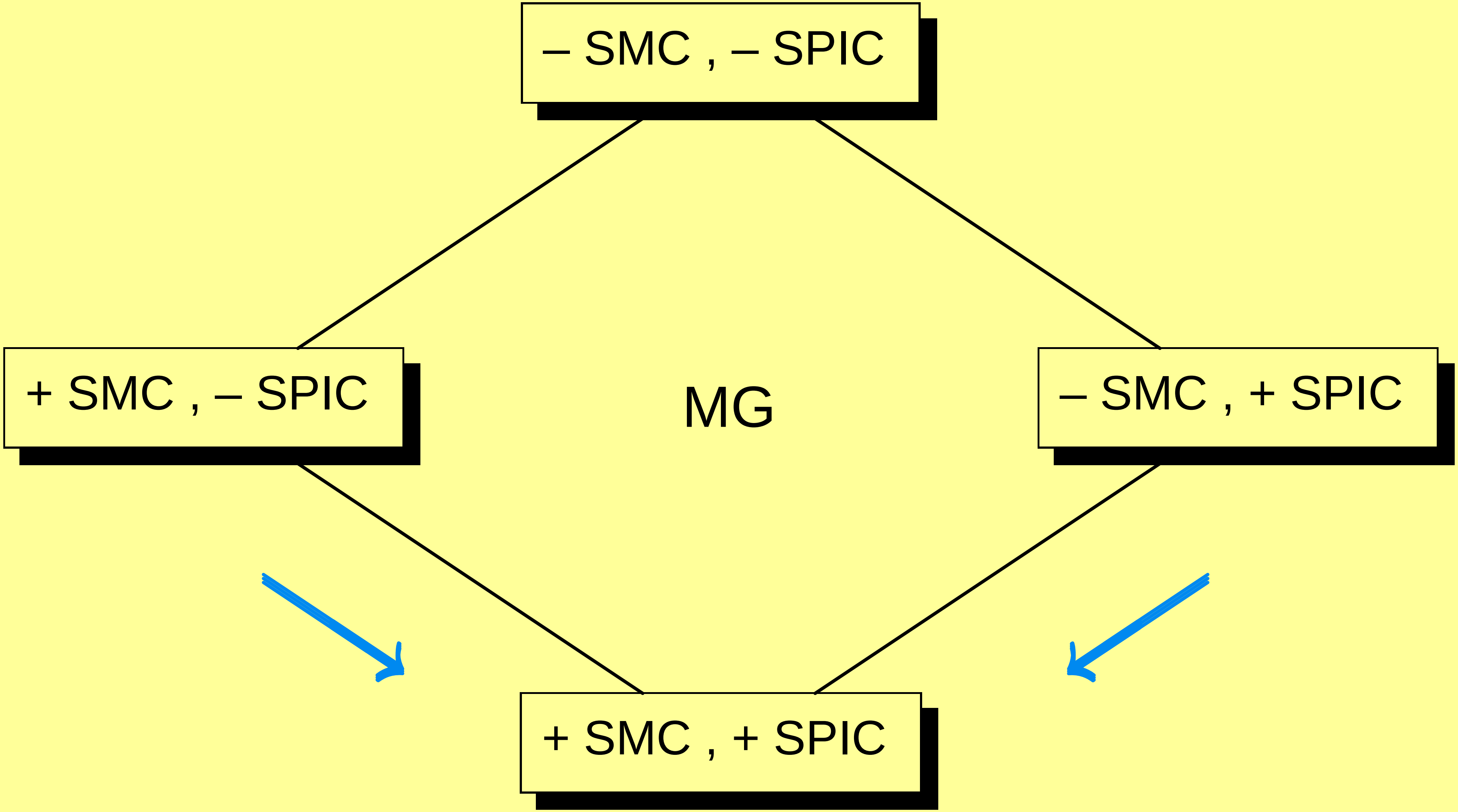
- each nonterminating rule is of the form $A \longrightarrow f(B)$ or $A \longrightarrow f(B, C)$,
- if $A \longrightarrow f(B, C)$, nonterminal B derives only simple terminal strings.

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

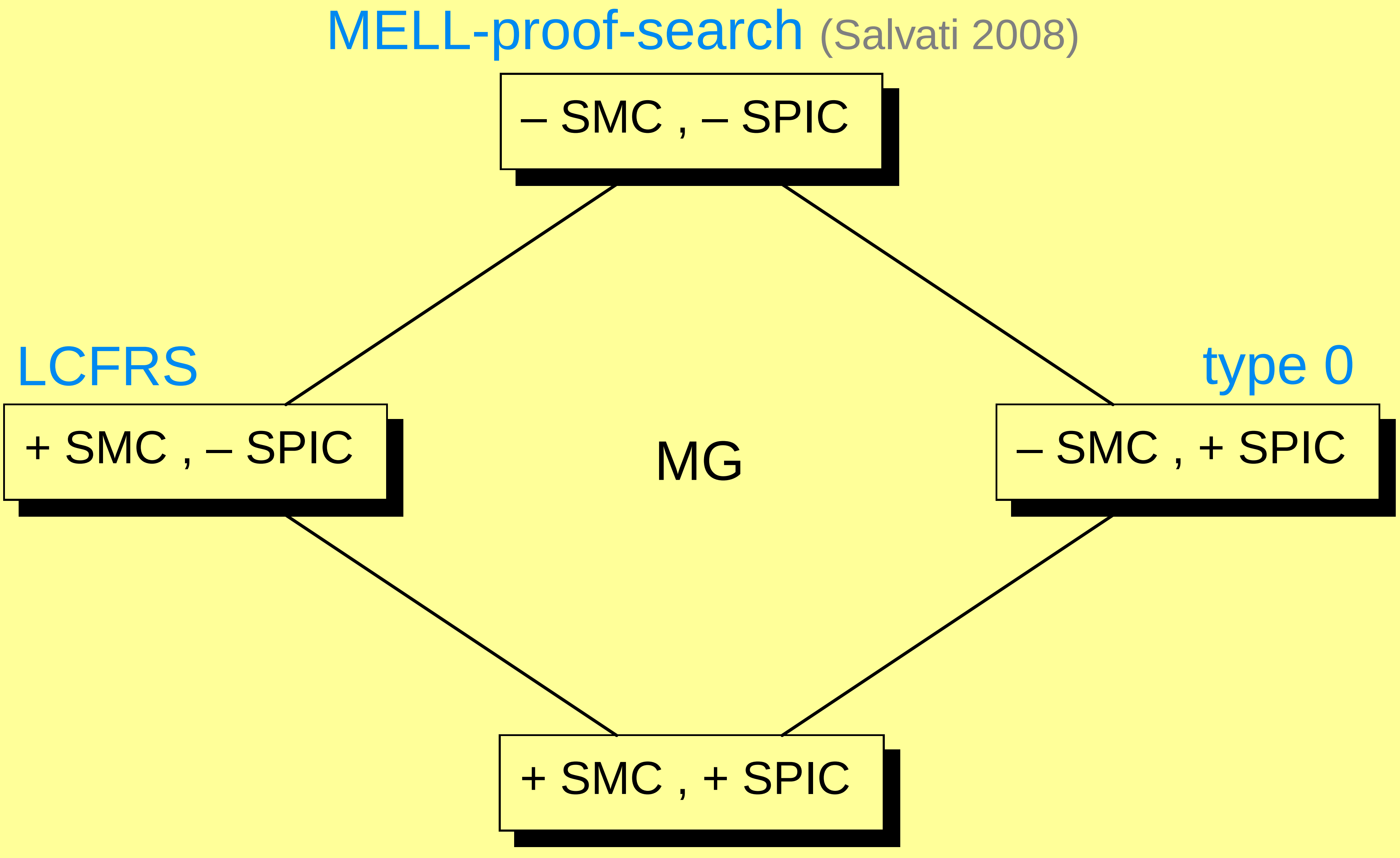
$$\left\{ w_1 \cdots w_n\, z_n\, w_n \cdots z_1\, w_1\, z_0\, w_n{}^R \cdots w_1{}^R \;\middle|\; w_i \in \{a, b\}^+, z_n \cdots z_0 \text{ Dyck word} \right\}$$


Indexed Grammar
LCFRS(1,2)
$L_{example}$
LCFRS

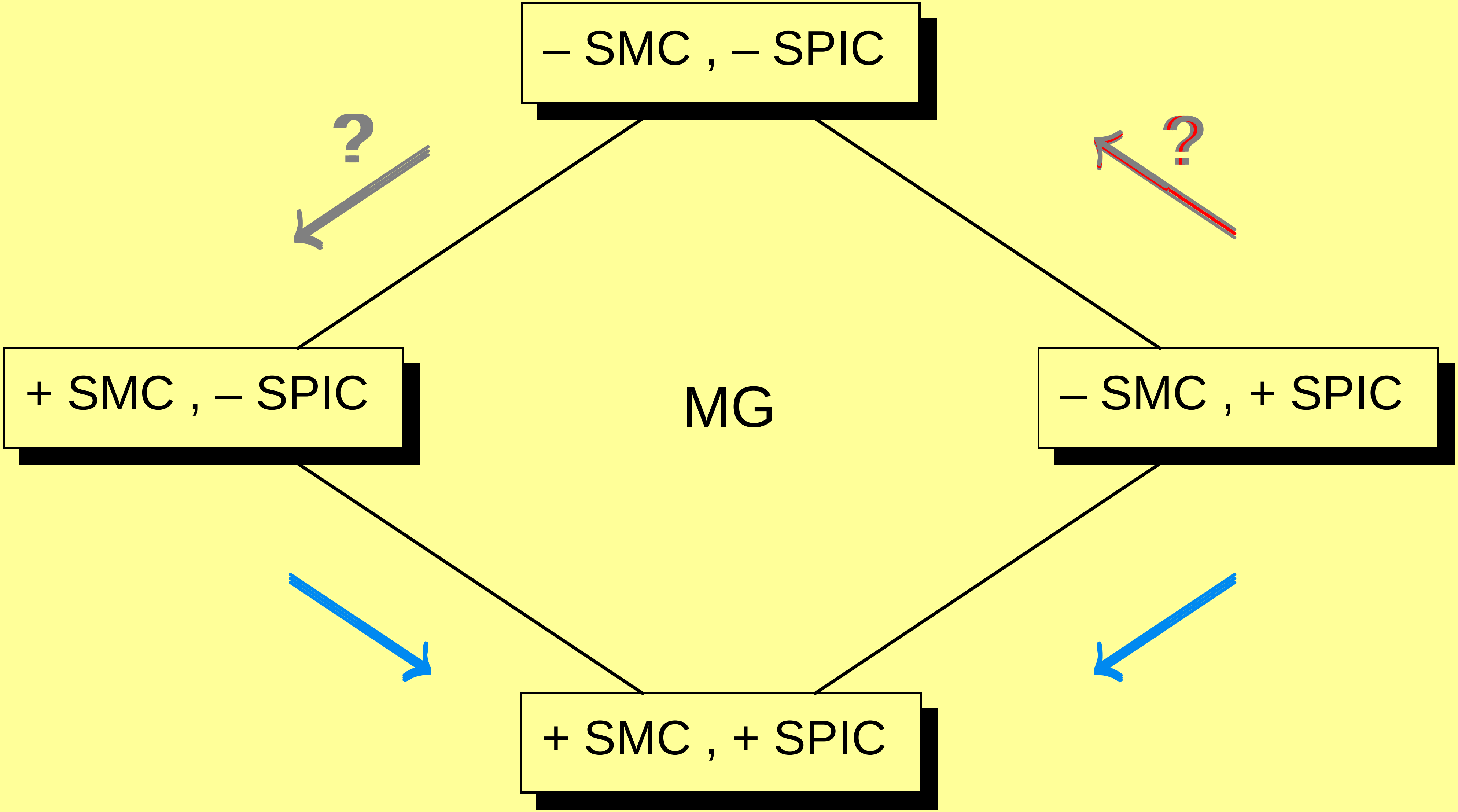
SMC and SPIC — restricting the move-operator domain



SMC and SPIC — restricting the move-operator domain



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 */

koj_i	$kogo_j$	$kakvo_k$	t_i	e	pital	t_j	t_k
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:* $^{\triangle}x$, $^{\triangle}y$, $^{\triangle}z$, . . .

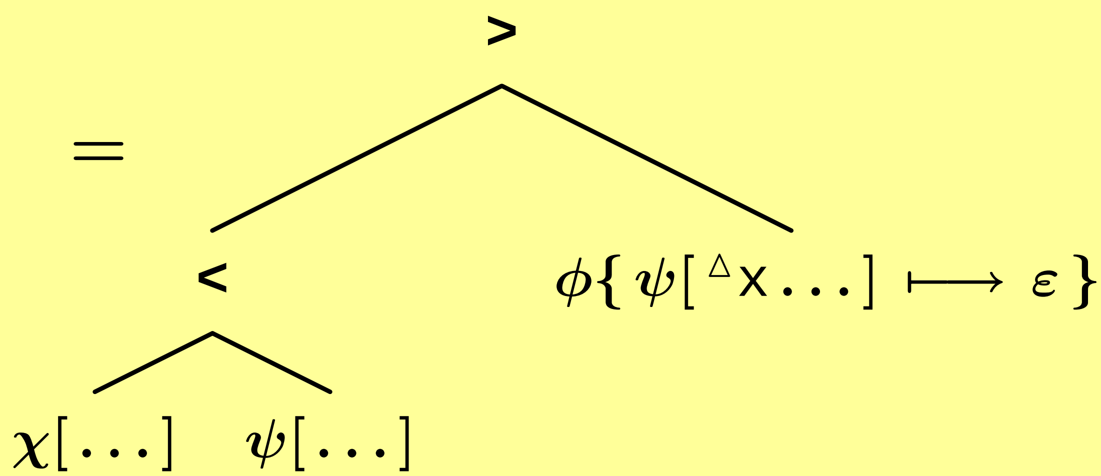
  c(luster)-licensors: $^{\triangledown}x$, $^{\triangledown}y$, $^{\triangledown}z$, . . .

Structure building functions

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

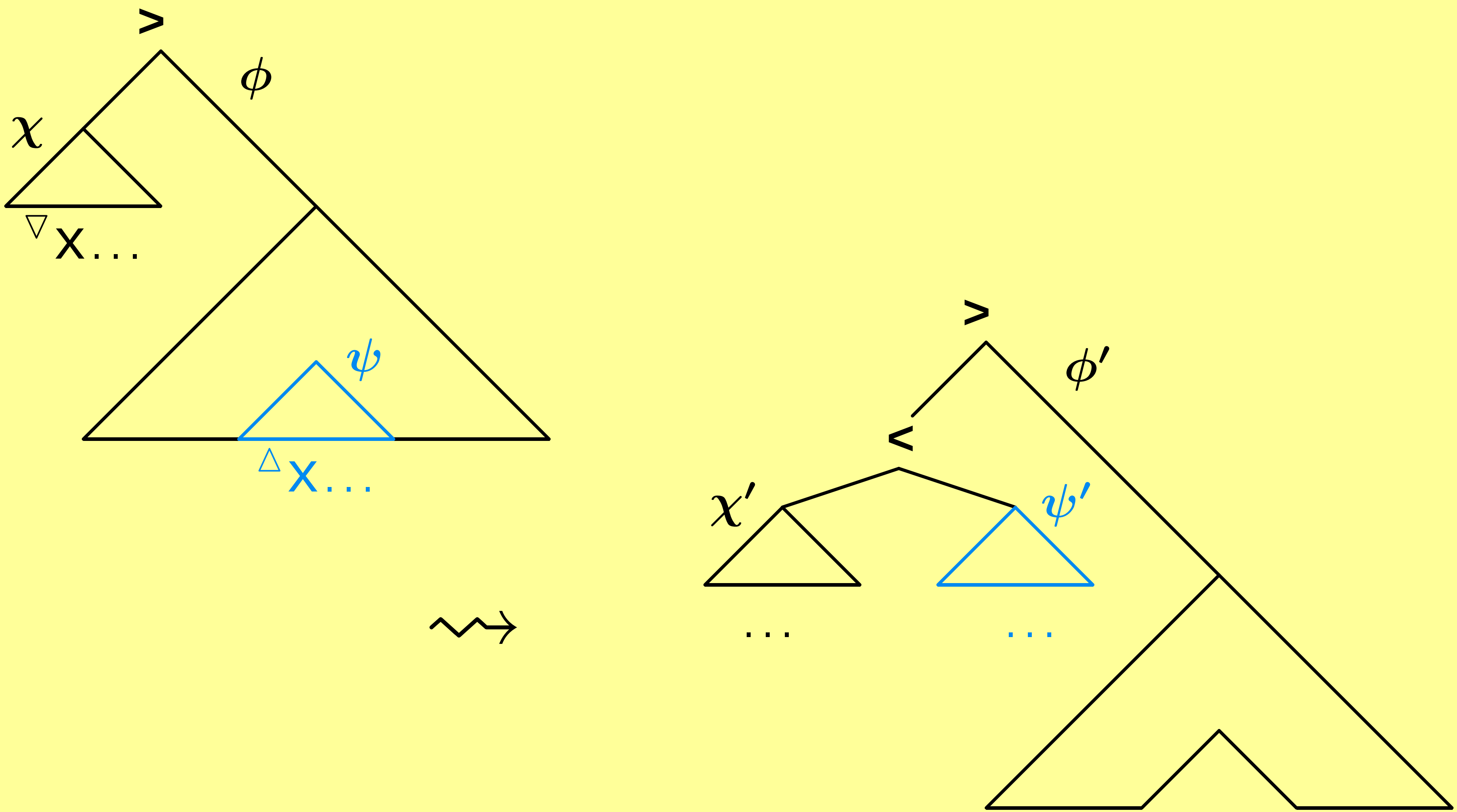
- $\phi \in \text{Domain(cluster)} :\Longleftrightarrow$
  - The highest specifier χ of ϕ displays c-licensor $^{\triangledown}$x
  - there is a (unique [SMC]) maximal projection ψ within ϕ that displays the corresponding c-licensee $^{\triangle}$x

- cluster(ϕ) $=$



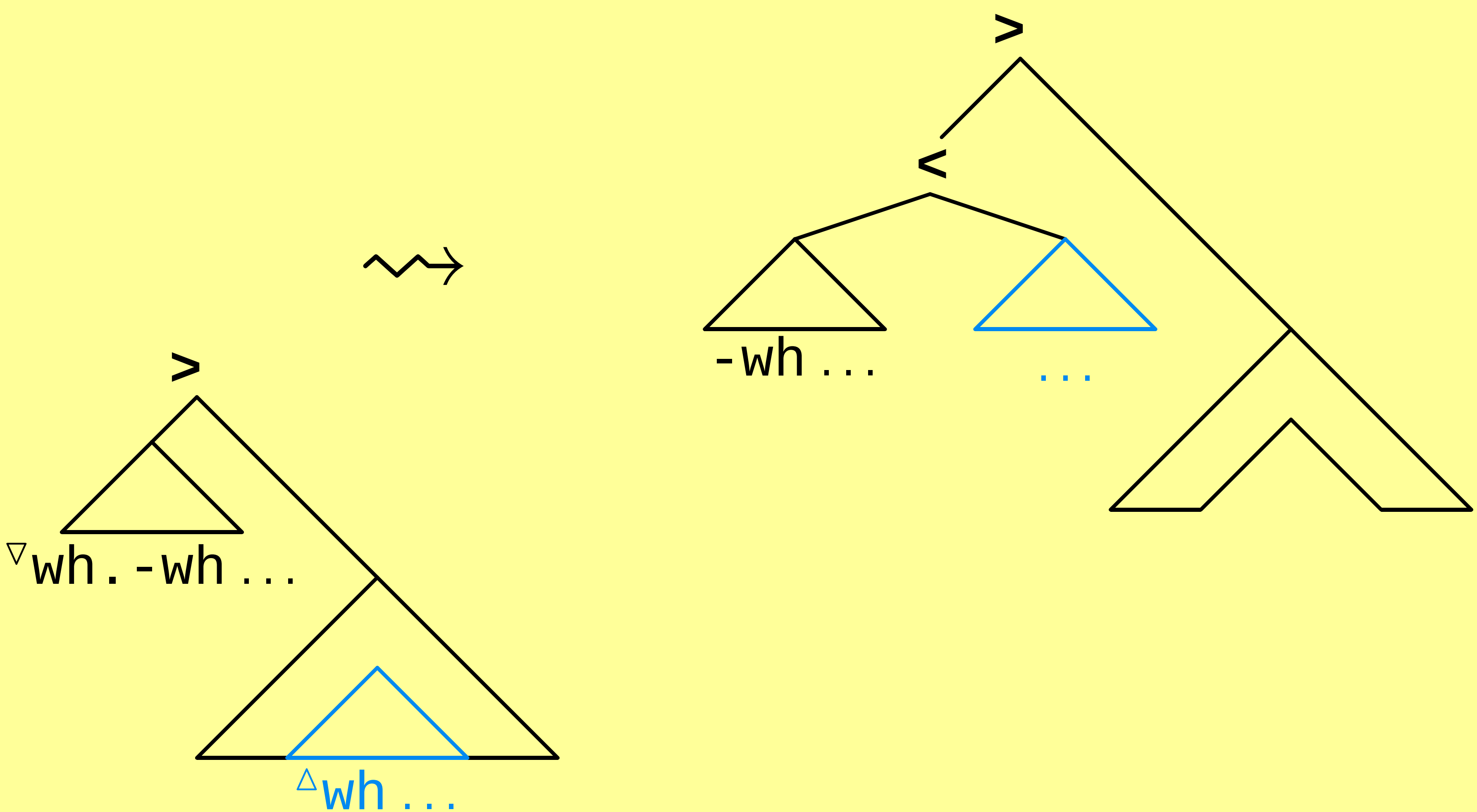
Structure building functions

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



- 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