

Querying and Transforming XML Documents

Alexandru Berlea + Helmut Seidl

Trier

April 5, 2003

I. Basics

- ◇ Words, Trees, ...
- ◇ Regular Sets
- ◇ Automata

II. Querying

- ◇ Specification
- ◇ Implementation
- ◇ Fxgrep

Part I

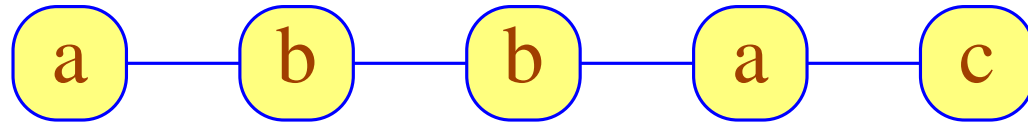
Basics.

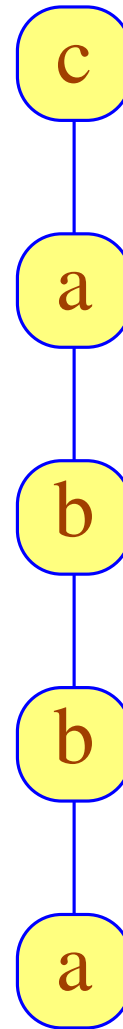
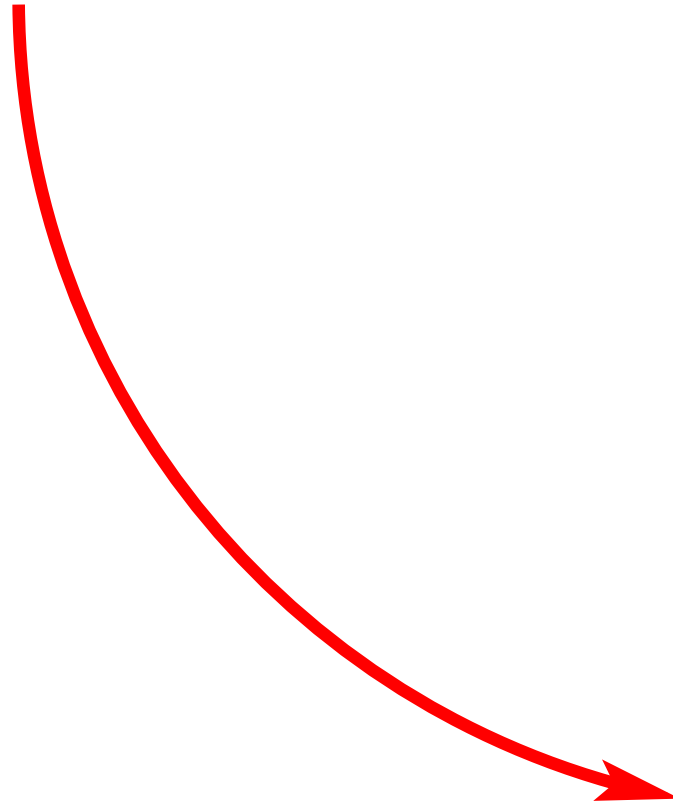
- ◇ Words
- ◇ Trees: Ranked — Un-ranked
- ◇ Grammars and related Formalisms
- ◇ Finite Automata and variations

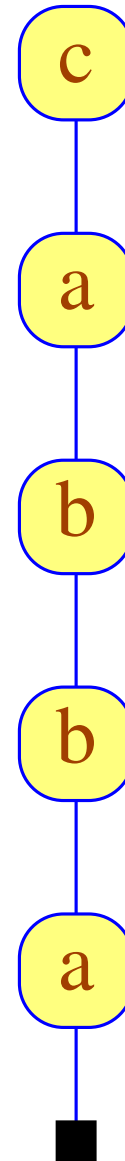
Finite words

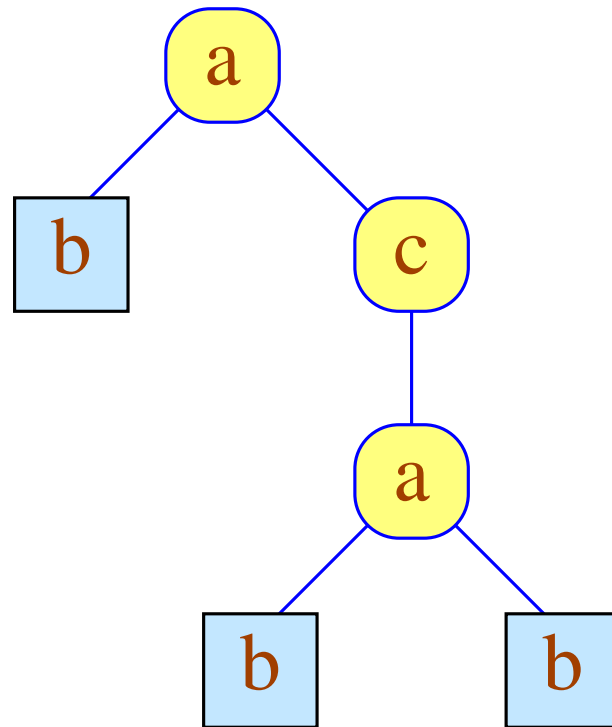
$$w = \text{abbac}$$

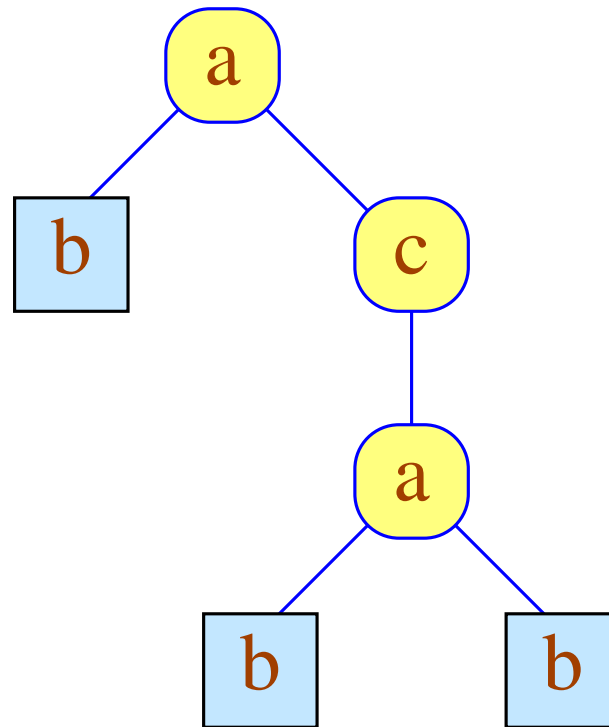
are monadic trees:



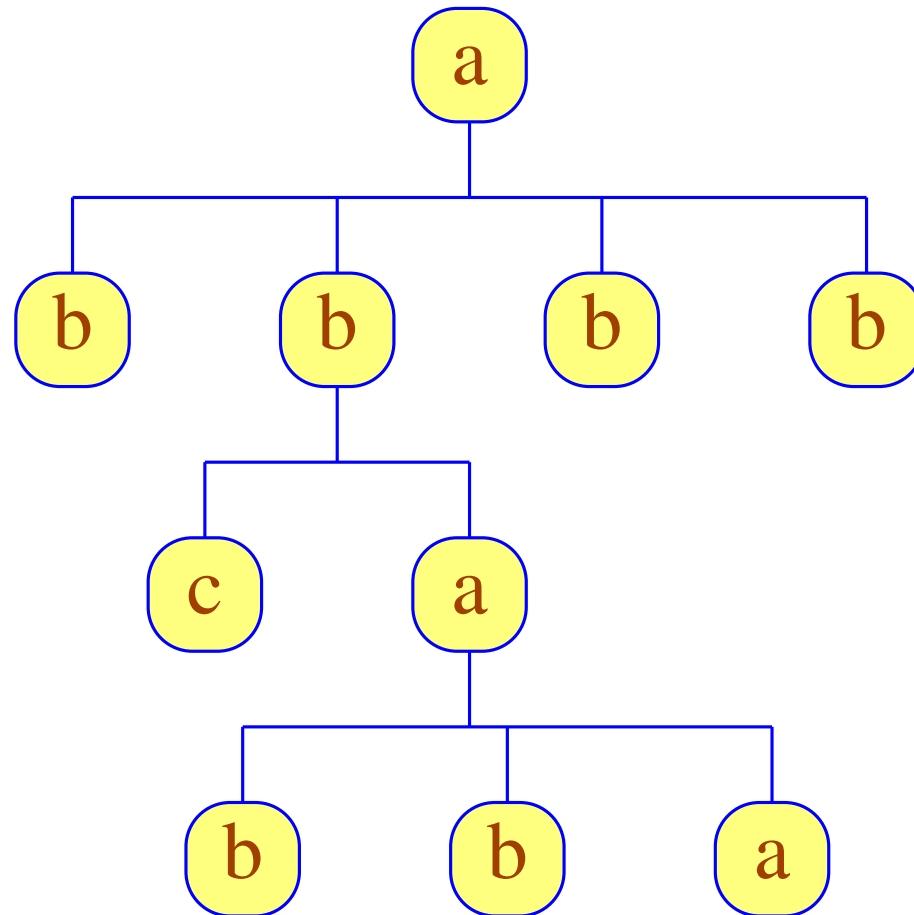


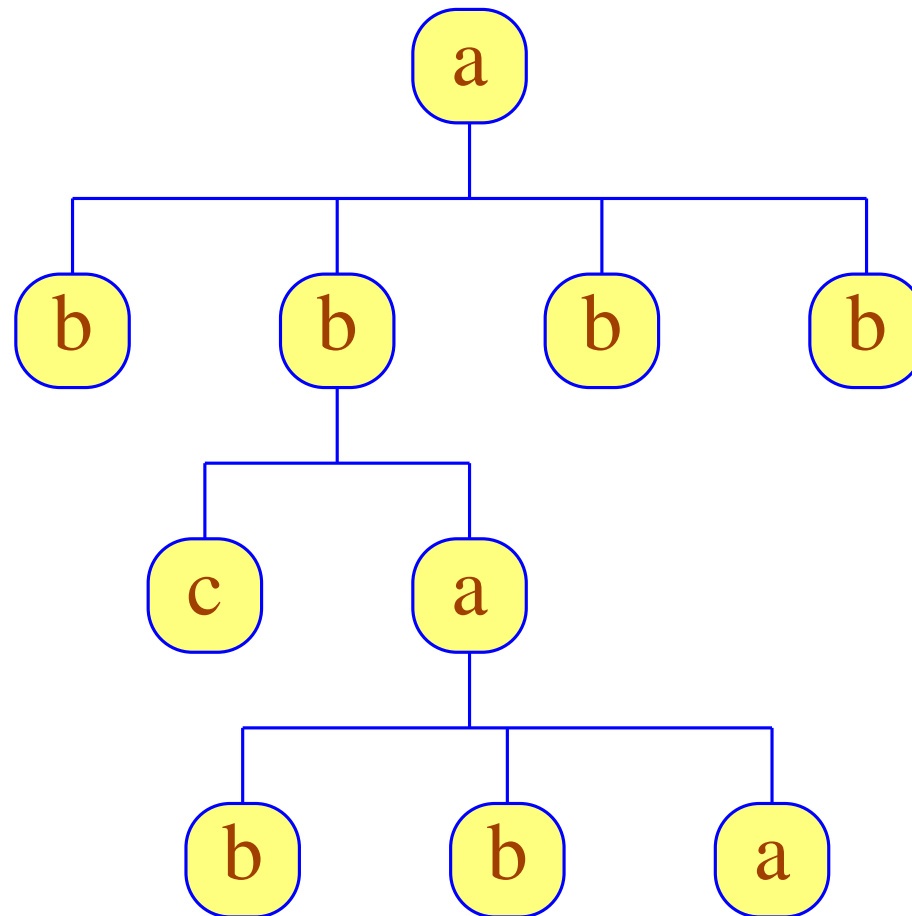






tree = term
internal node = operator application
leaf = constant

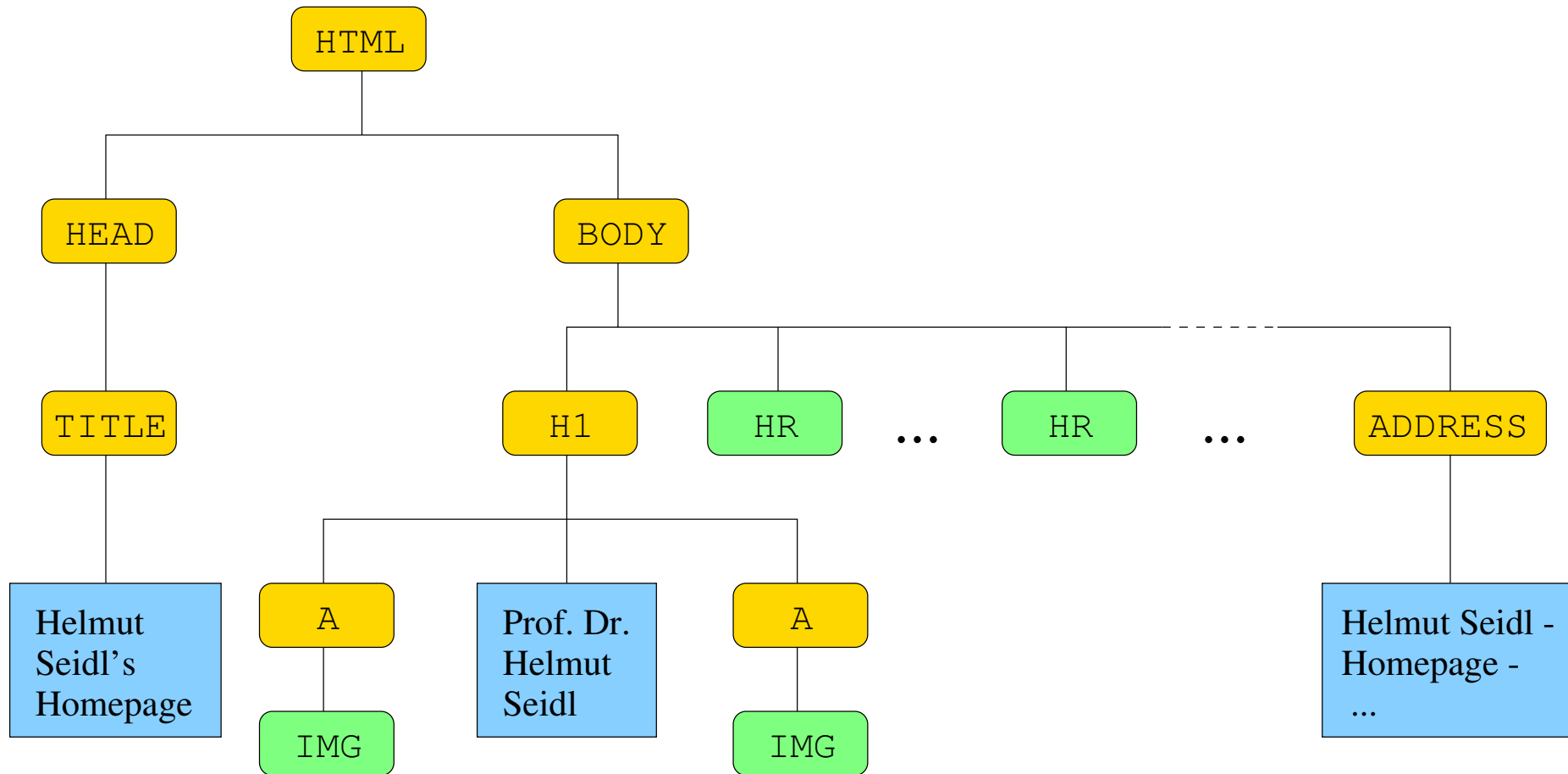




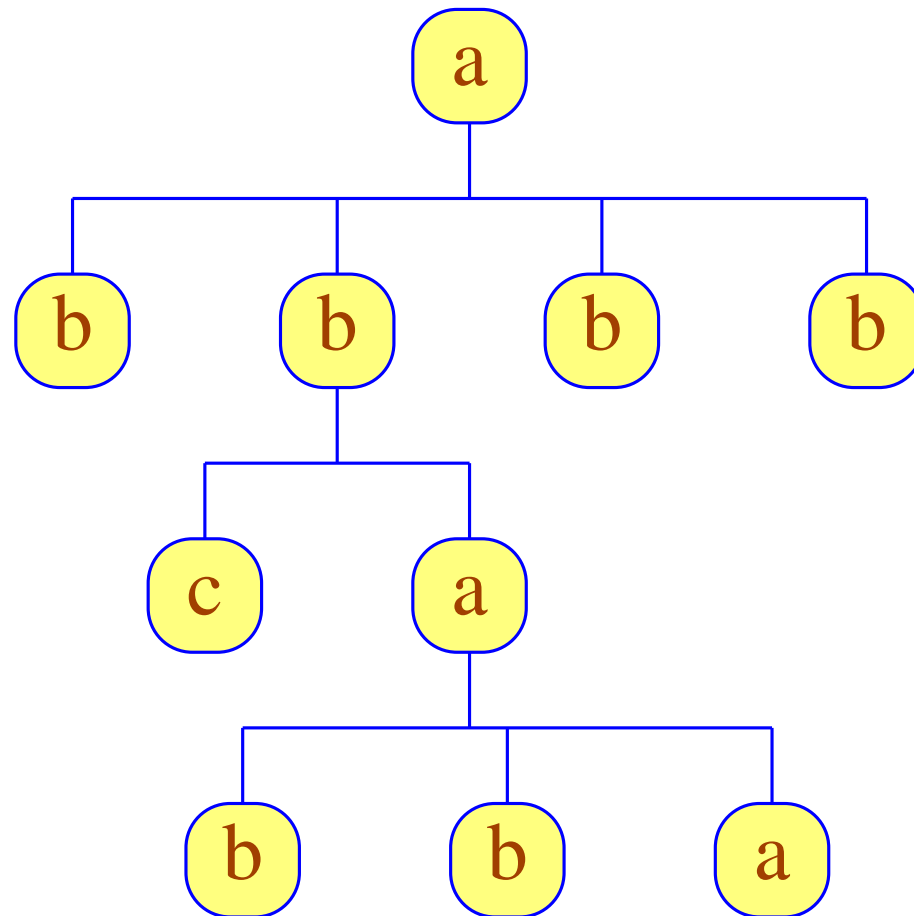
tree = document structure

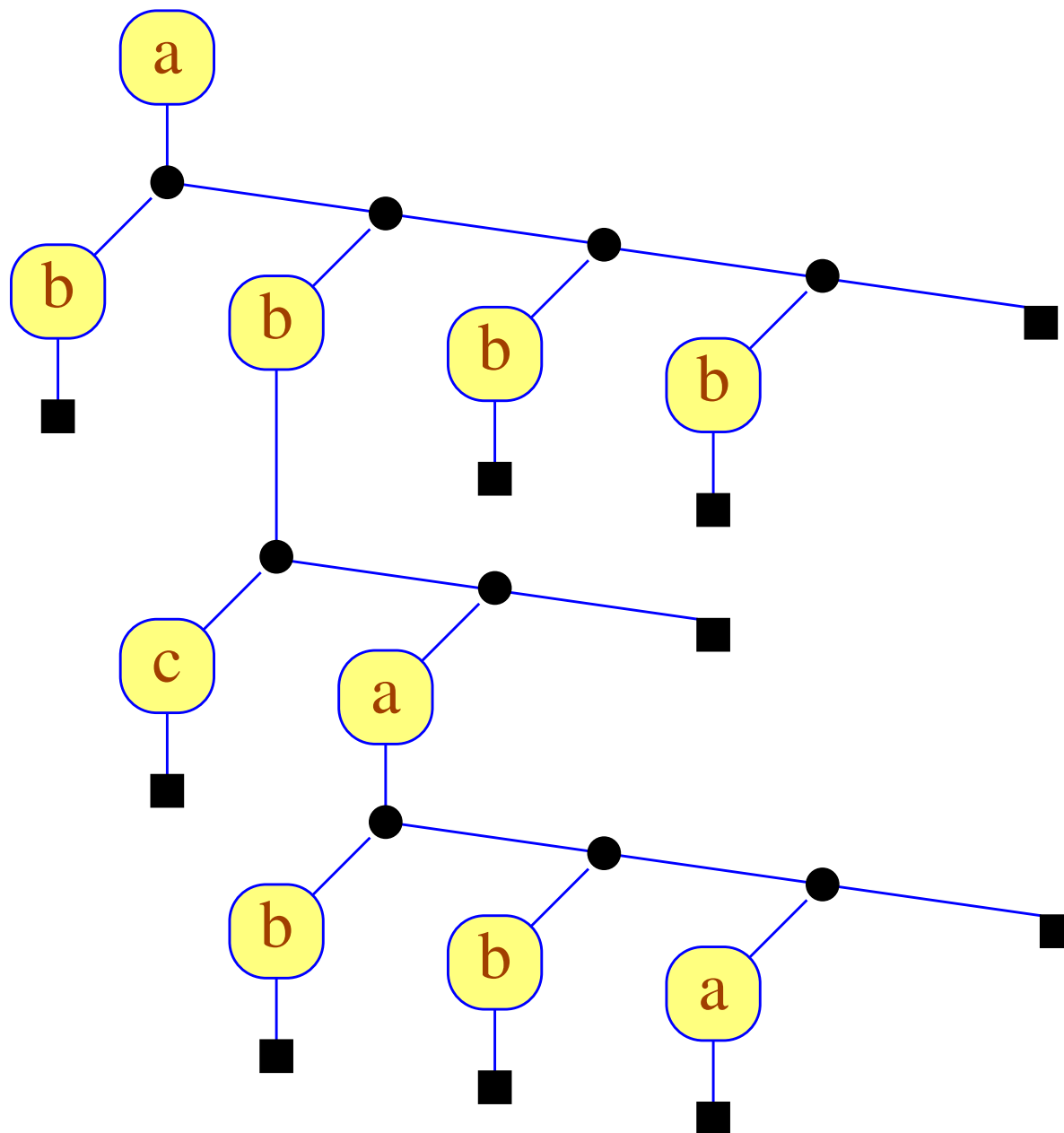
node = element

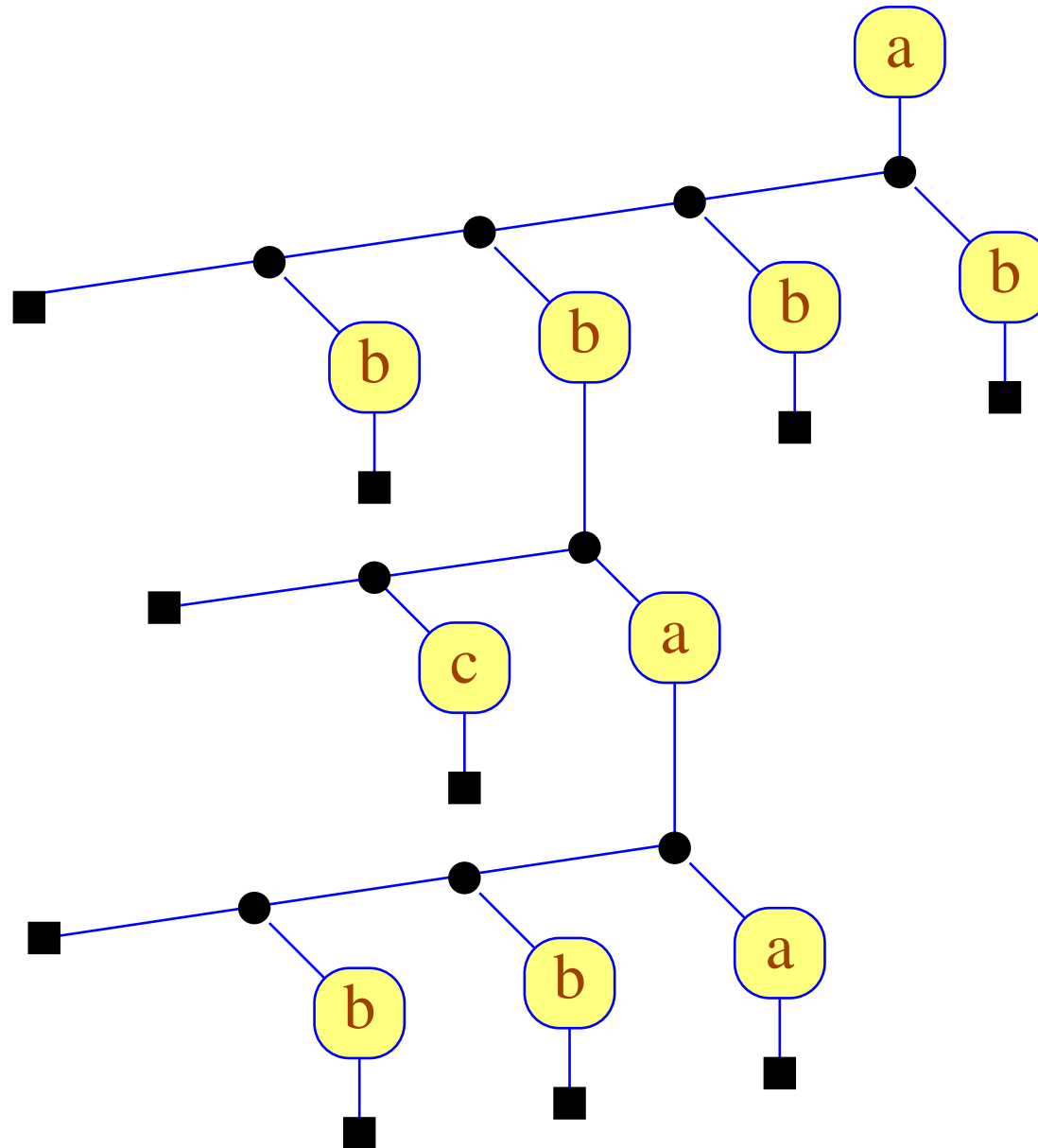
Trees: the Unranked Case (cont.) 7

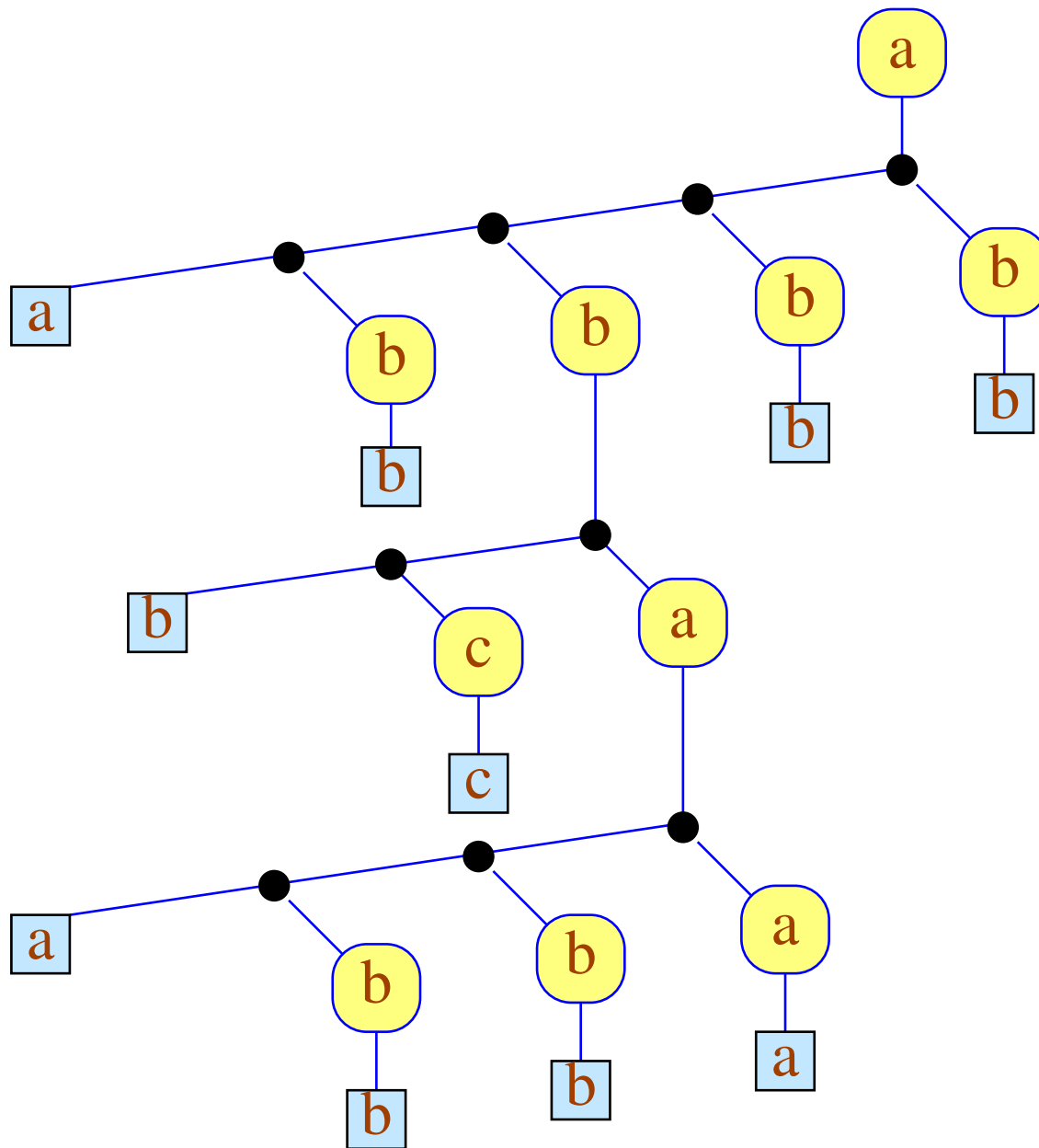


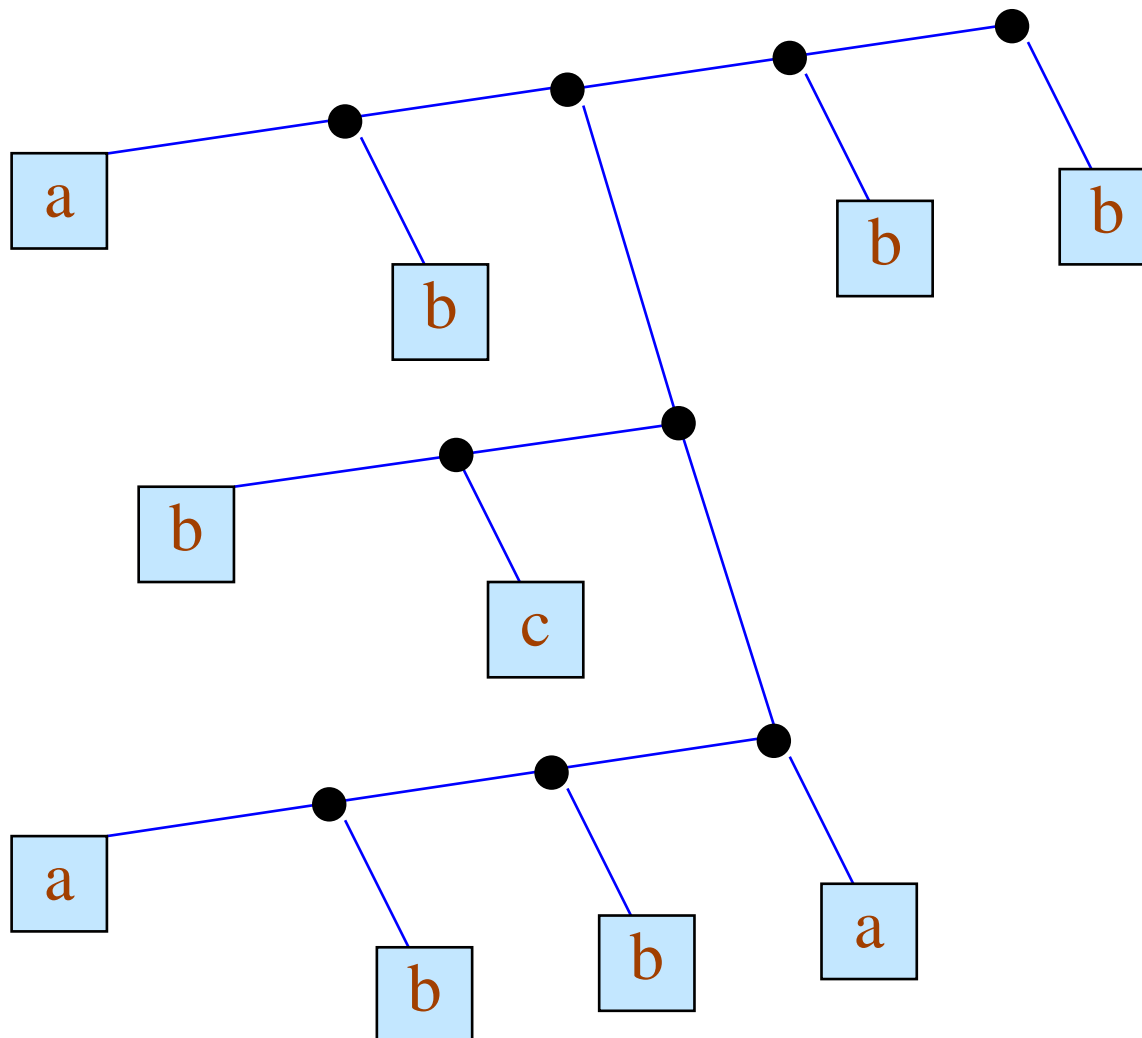
Unranked trees can be seen as special ranked trees ...

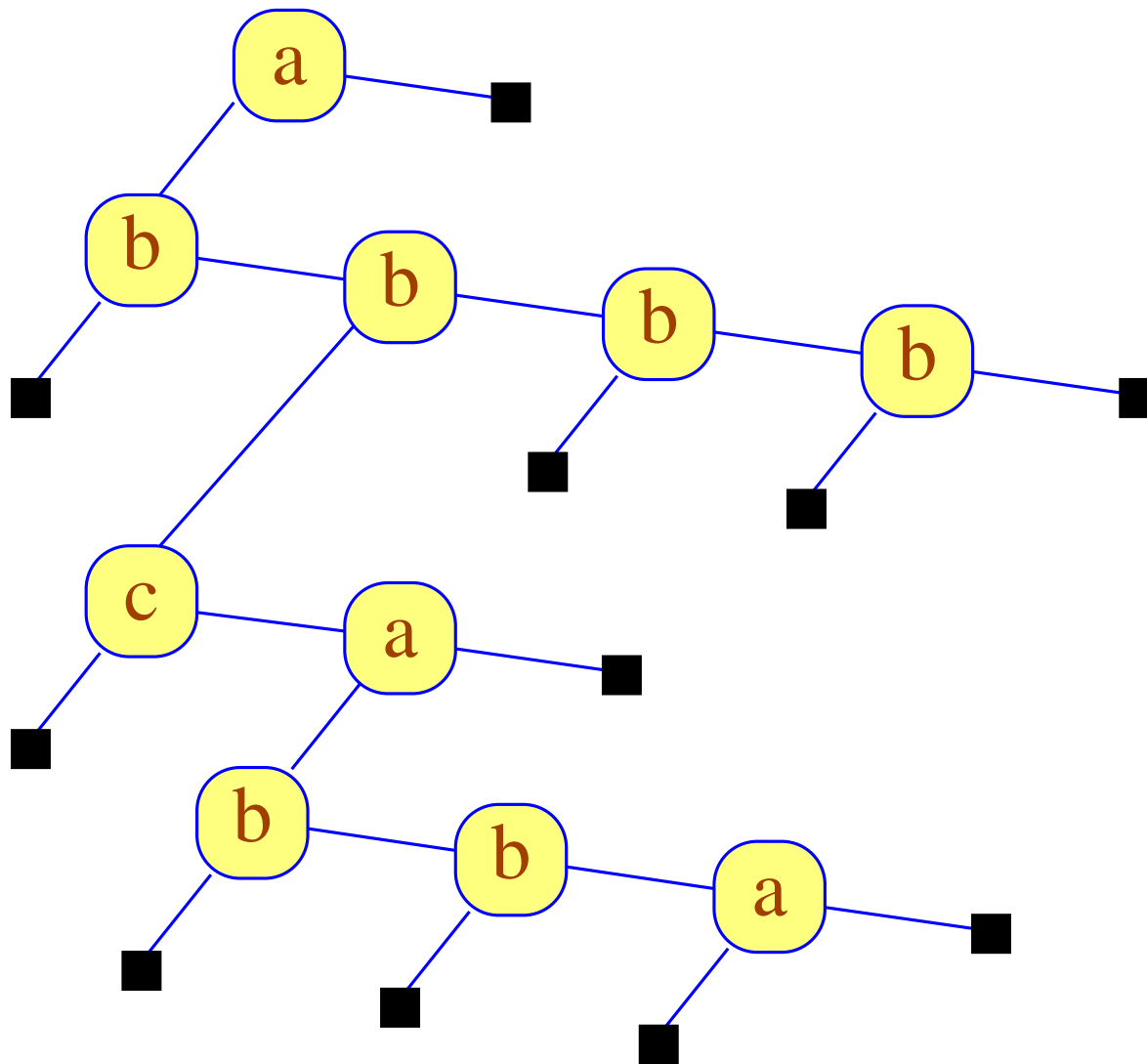












- ✧ There is **no unique** coding of unranked trees.
- ✧ Algorithms may differ in their **view** onto a tree.



Change view whenever needed :-)

Property of t : $P(t)$ (P monadic predicate)

Property of t : $P(t)$ (P monadic **predicate**)
: $t \in R$ (R a **set**)

Property of t : $P(t)$ (P monadic predicate)
: $t \in R$ (R a set)

Regular sets = simple class of properties

- grammars;
- regular expressions;
- fixpoint formulas;
- MSO logic.

- automata;
- automata;
- automata;
- ...

Grammar:

$$\begin{array}{l} A \longrightarrow aA \quad | \quad bB \\ B \longrightarrow aB \quad | \quad \epsilon \end{array}$$

$$A \Longrightarrow^* aaabaa$$

Expression: a^*ba^*

Fixpoint formula: $\mu x. ax \vee b(\mu x. ax \vee \epsilon)$

MSO formula:

Expression: a^*ba^*

Fixpoint formula: $\mu x. ax \vee b(\mu x. ax \vee \epsilon)$

MSO formula:

$$\exists x. \text{lab}_b(x) \wedge \forall y. x \neq y \Rightarrow \text{lab}_a(y)$$

Grammar:

$$A \longrightarrow a(A, A) \quad | \quad b(B)$$

$$B \longrightarrow a(B, B) \quad | \quad c$$

$$A \Longrightarrow^* a(b(a(c, c)), a(b(c), b(c)))$$

Expression:

$$(a(A, A))^{*A} \cdot_A b((a(B, B))^{*B}) \cdot_B c$$

Fixpoint formula:

$$\mu x. a(x, x) \vee b(\mu x. a(x, x) \vee c)$$

Expression:

$$(a(A, A))^{*A} \cdot_A b((a(B, B))^{*B}) \cdot_B c$$

Fixpoint formula:

$$\mu x. a(x, x) \vee b(\mu x. a(x, x) \vee c)$$

Warning:

In general, several $*_A, *_B, \dots$ needed !!!

Ranked Trees: Specification (cont.) 17

MSO formula:

$$(\forall x. \text{lab}_c(x) \vee \text{lab}_b(x) \vee \text{lab}_a(x)) \wedge$$

$$(\forall x. \text{lab}_c(x) \Rightarrow \exists y. y < x \wedge \text{lab}_b(y)) \wedge$$

$$(\forall x. \forall y. \text{lab}_b(x) \wedge \text{lab}_b(y) \wedge x \leq y \Rightarrow x = y)$$

Ranked Trees: Specification (cont.) 18

Ingredients:

- ◇ rank information for a, b, c ;
- ◇ successor relations $\text{succ}_1, \text{succ}_2, \dots$
- ◇ ancestor relation \leq
(definable from successor relations).

Grammar:

$$A \longrightarrow \langle a \rangle A^+ \langle /a \rangle \quad | \quad \langle b \rangle B^* \langle /b \rangle$$

$$B \longrightarrow \langle a \rangle B^* \langle /a \rangle$$

$$A \implies^* \langle a \rangle \langle b / \rangle \langle a \rangle \langle b / \rangle \langle b / \rangle \langle a / \rangle \langle /a \rangle$$

Grammar:

$$A \longrightarrow a\langle A^+ \rangle \quad | \quad b\langle B^* \rangle$$

$$B \longrightarrow a\langle B^* \rangle$$

$$A \Longrightarrow^* a\langle b\langle \rangle a\langle b\langle \rangle b\langle \rangle \rangle$$

Grammar:

$$A \longrightarrow a\langle A^+ \rangle \quad | \quad b\langle B^* \rangle$$

$$B \longrightarrow a\langle B^* \rangle$$

$$A \Longrightarrow^* a\langle ba\langle bb \rangle \rangle$$

Expression:

$$(a\langle A^+ \rangle)^{*A} \cdot_A b\langle a\langle B^* \rangle^{*B} \rangle \cdot_B a$$

Fixpoint formula:

$$\mu x. a\langle x^+ \rangle \vee b\langle \mu x. a\langle x^* \rangle \rangle$$

Expression:

$$(a\langle A^+ \rangle)^{*A} \cdot_A b\langle a\langle B^* \rangle^{*B} \rangle \cdot_B a$$

Fixpoint formula:

$$\mu x. a\langle x^+ \rangle \vee b\langle \mu x. a\langle x^* \rangle \rangle$$

Ingredients:

- ◇ Word regular expressions for contents;
- ◇ Tree regular operators for nesting.

MSO formula:

$$(\forall x. \text{lab}_b(x) \vee \text{lab}_a(x)) \wedge$$

$$(\forall x. \text{lab}_a(x) \Rightarrow \exists y. \text{lab}_b(y) \wedge (y < x \vee x < y)) \wedge$$

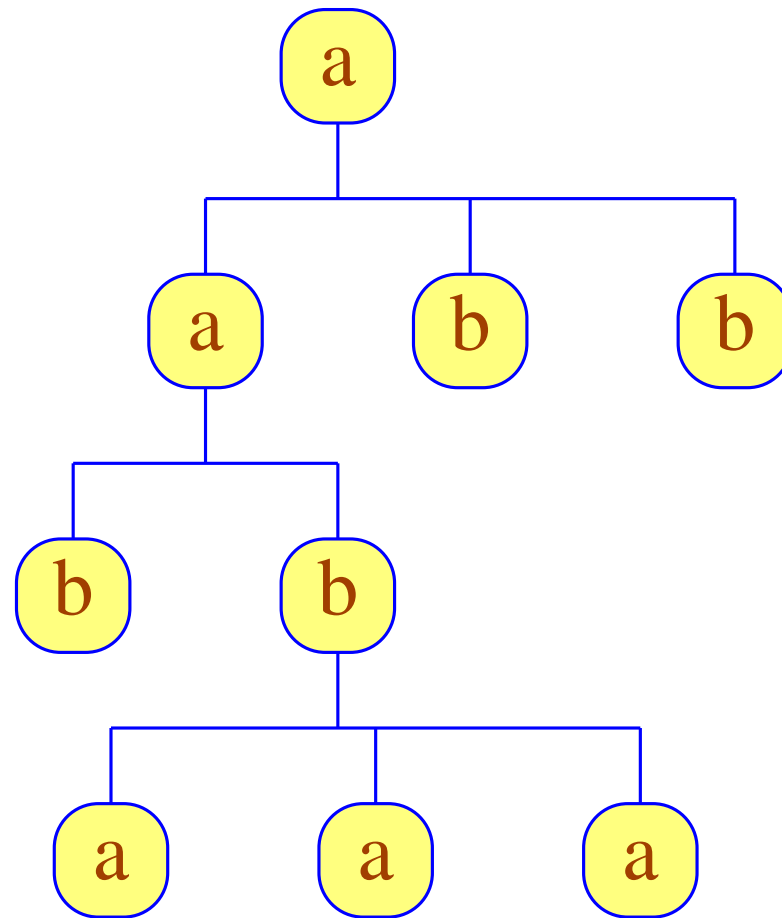
$$(\forall x. \forall y. \text{lab}_b(x) \wedge \text{lab}_b(y) \wedge x \leq y \Rightarrow x = y)$$

Ingredients:

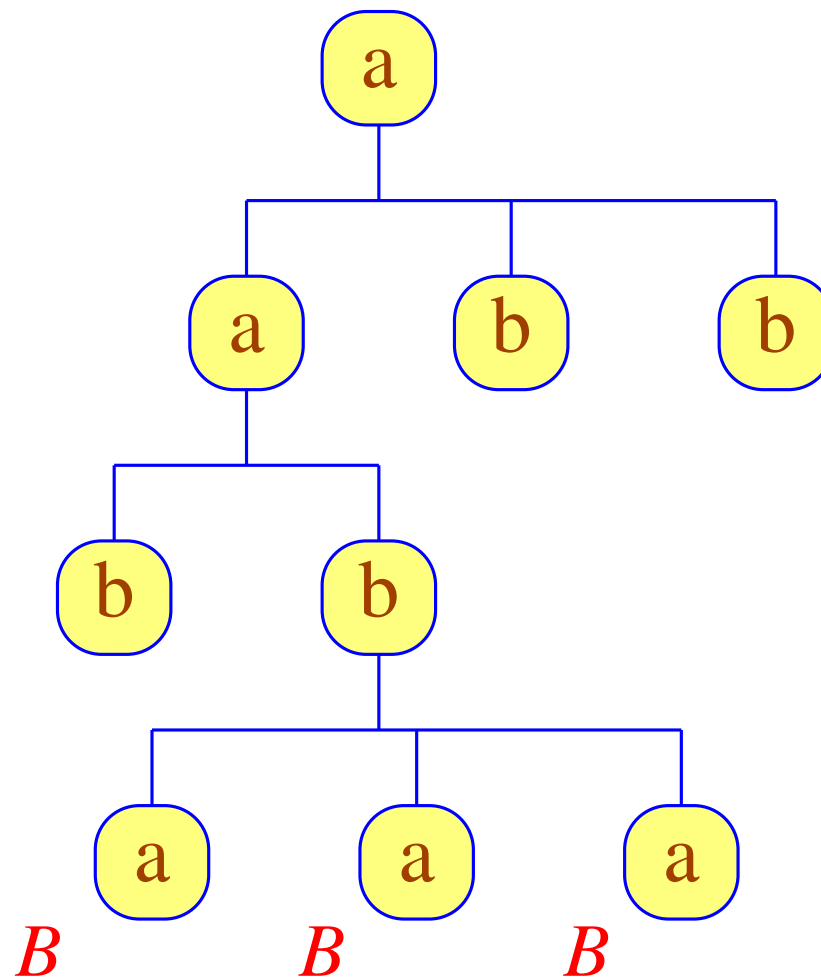
- ◇ left sibling relation;
- ◇ ancestor relation \leq .

Non-deterministic finite automaton \approx grammar

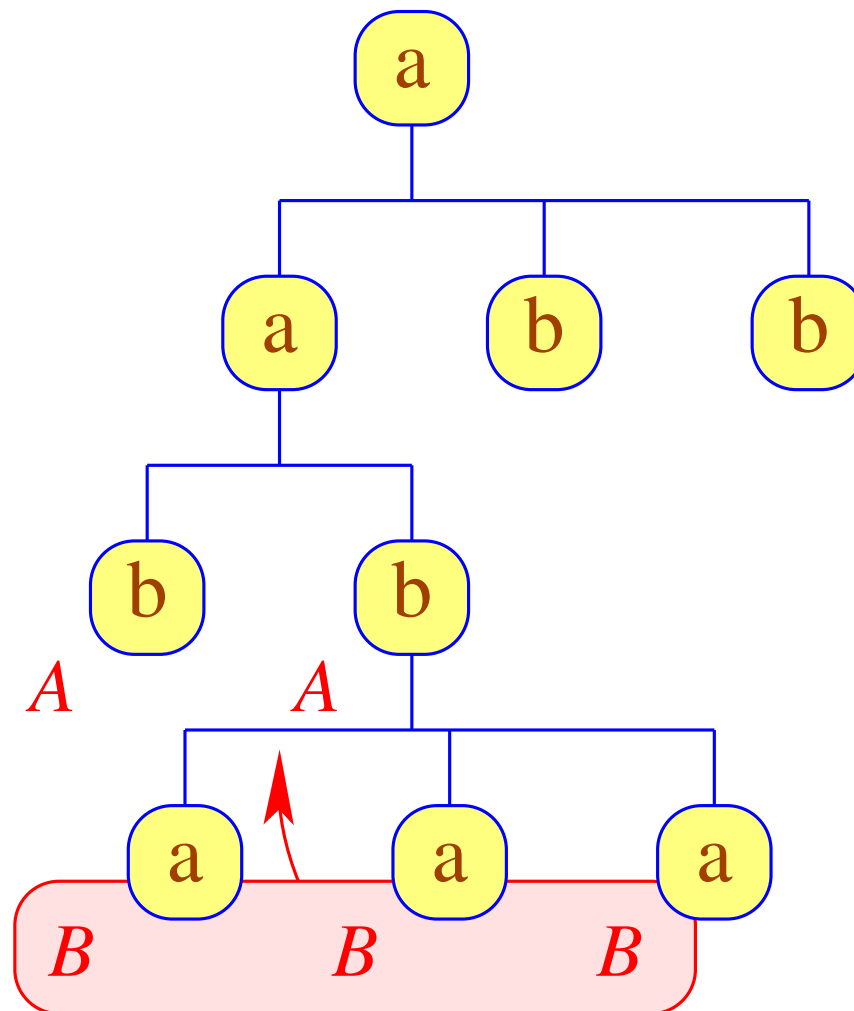
Non-deterministic finite automaton \approx grammar



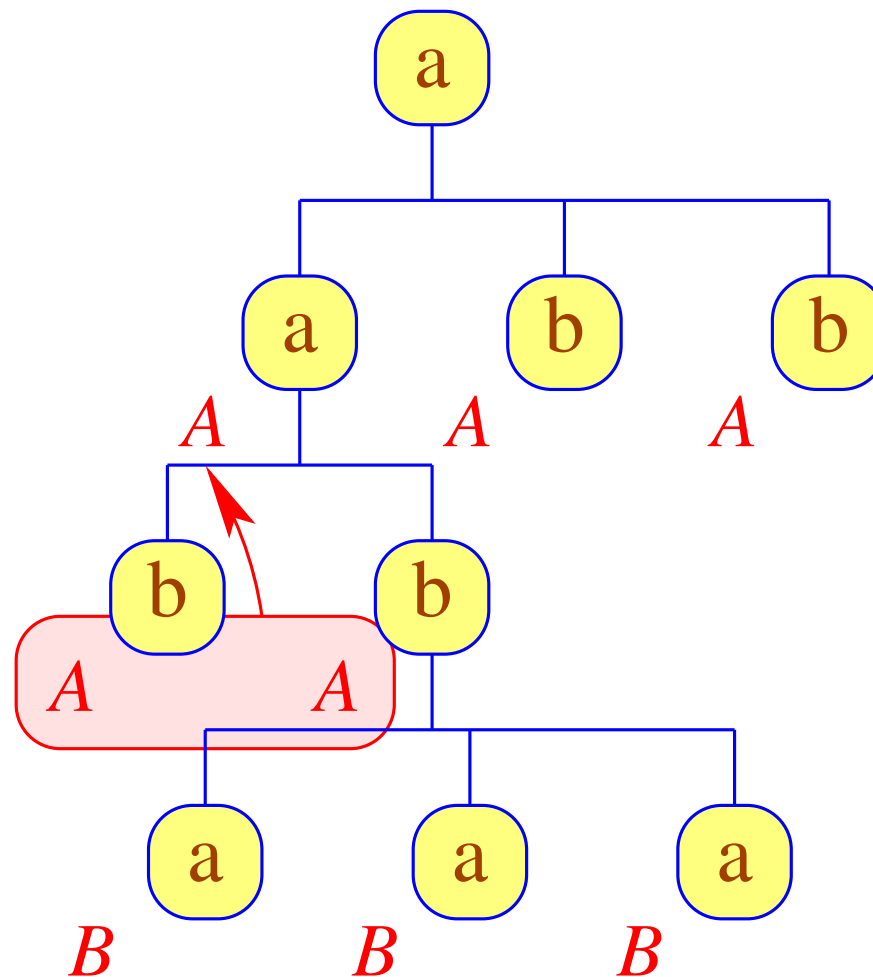
Non-deterministic finite automaton \approx grammar



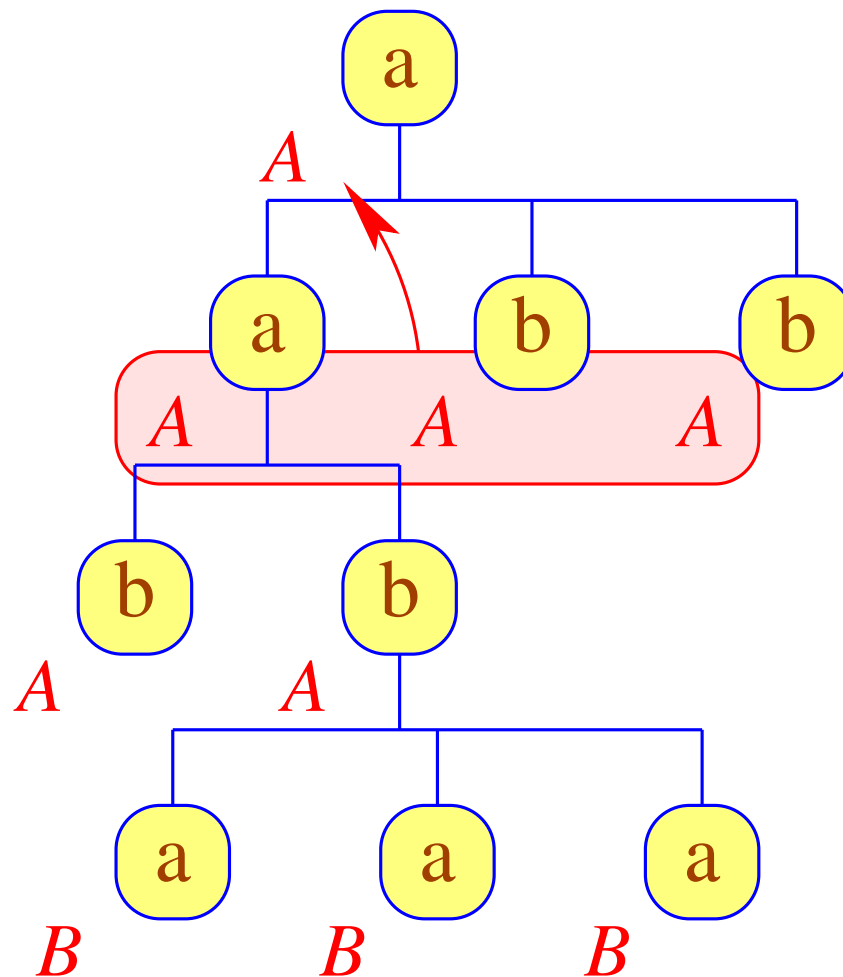
Non-deterministic finite automaton \approx grammar



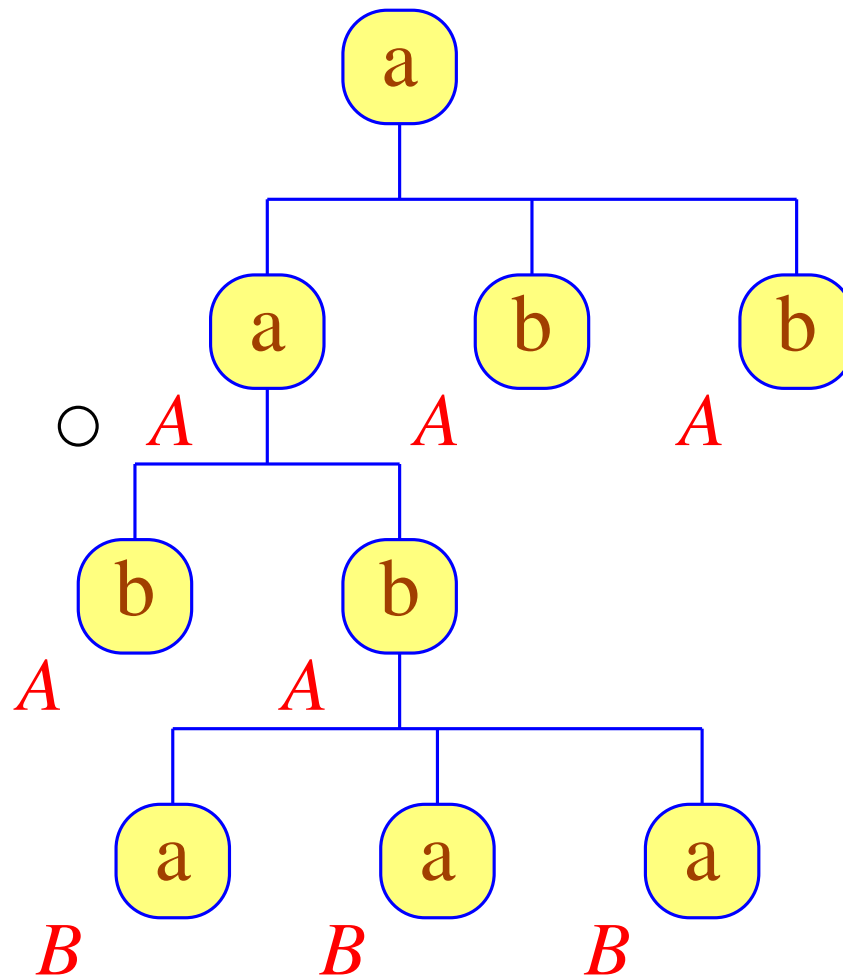
Non-deterministic finite automaton \approx grammar



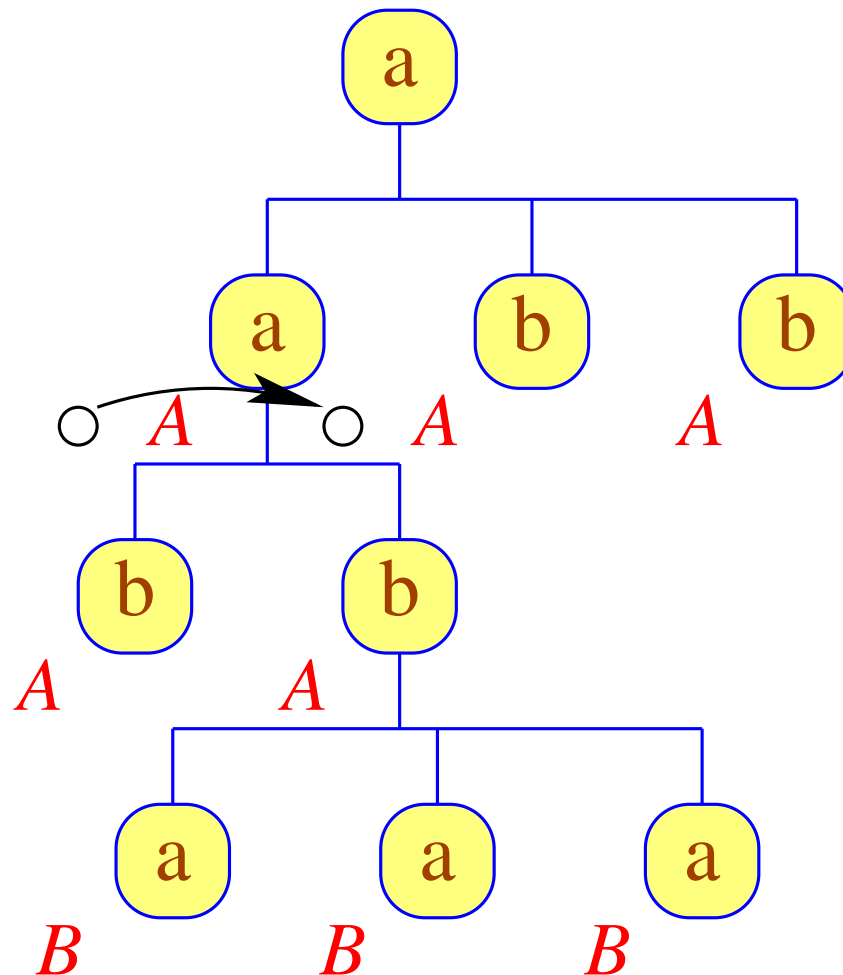
Non-deterministic finite automaton \approx grammar



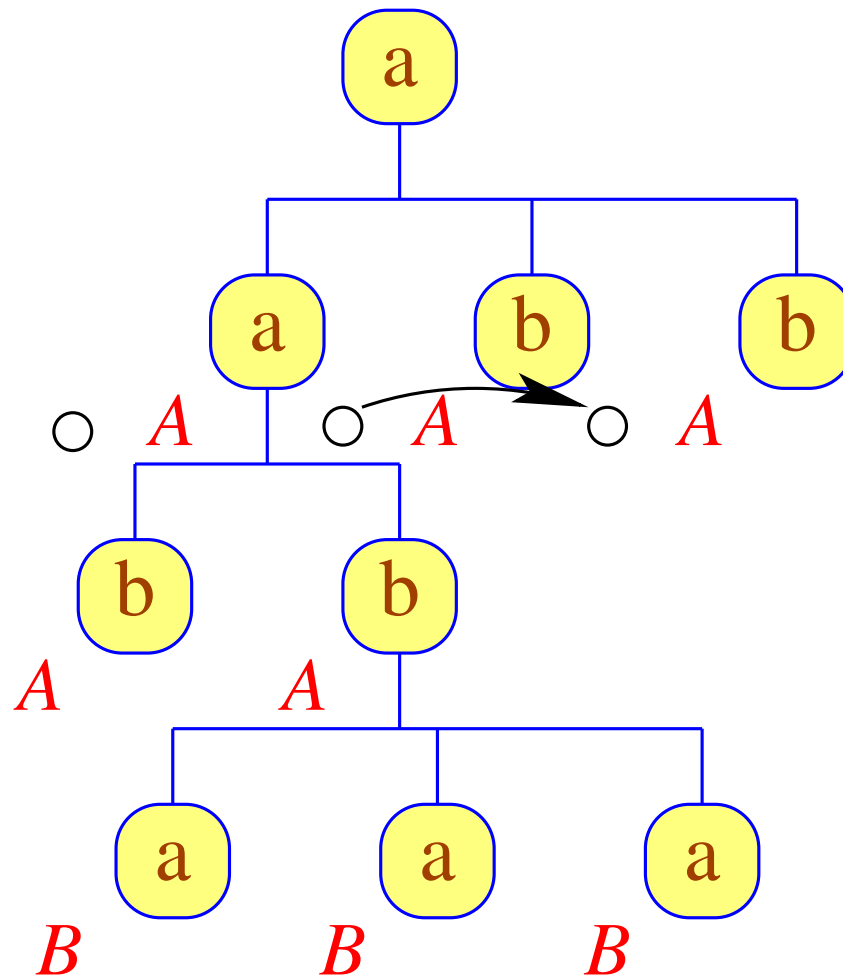
For testing of containment in regular set for successors we use a **word** automaton ...



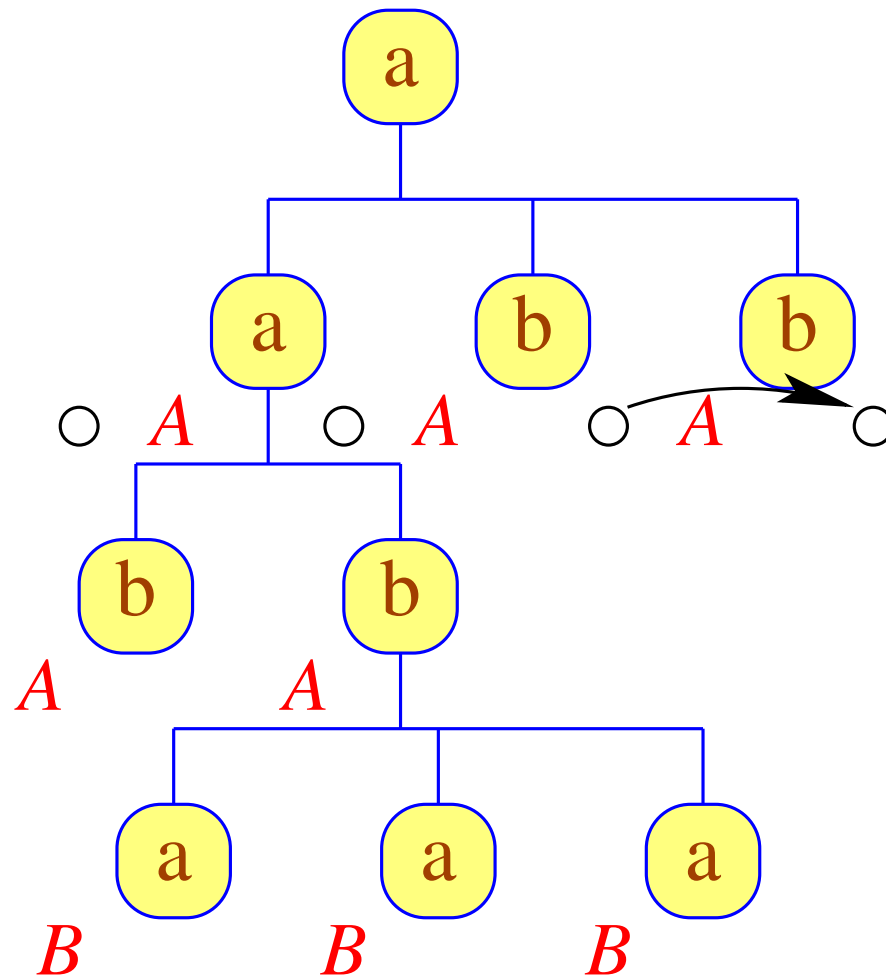
For testing of containment in regular set for successors we use a **word** automaton ...



For testing of containment in regular set for successors we use a **word** automaton ...



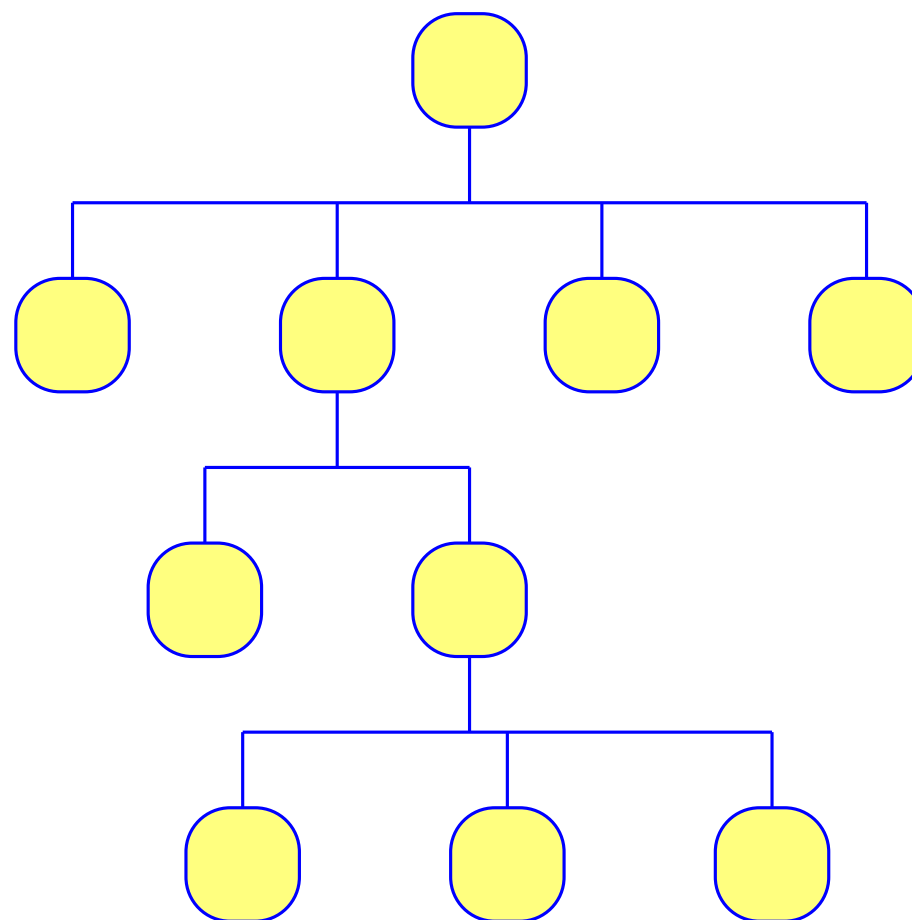
For testing of containment in regular set for successors we use a **word** automaton ...



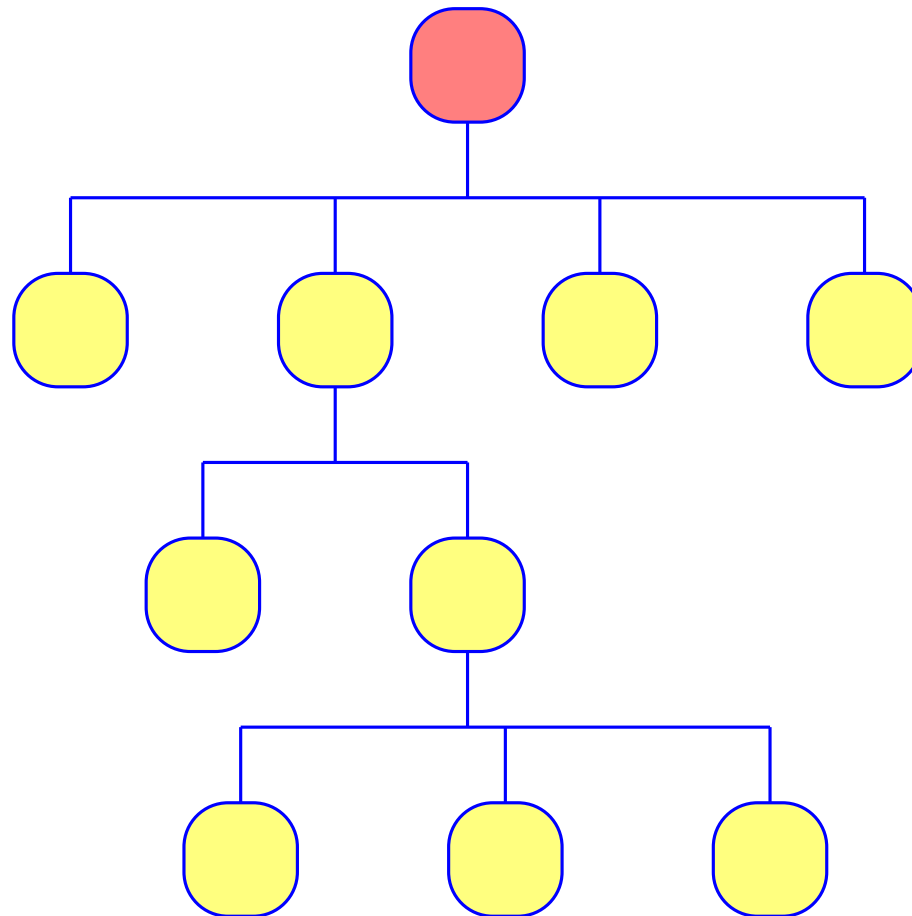
Implementation:

Specification	automaton
grammar	polynomial
reg. expression	polynomial
fixpoint formula	exponential
MSO formula	non-elementary

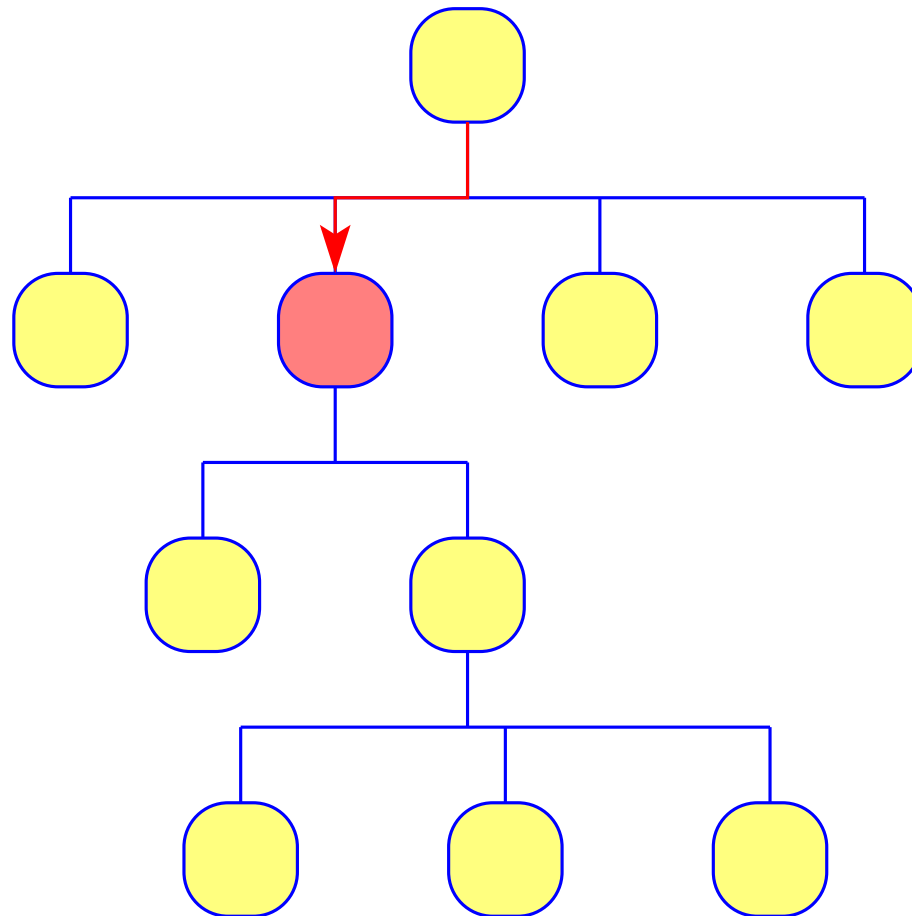
Tree walking automaton:



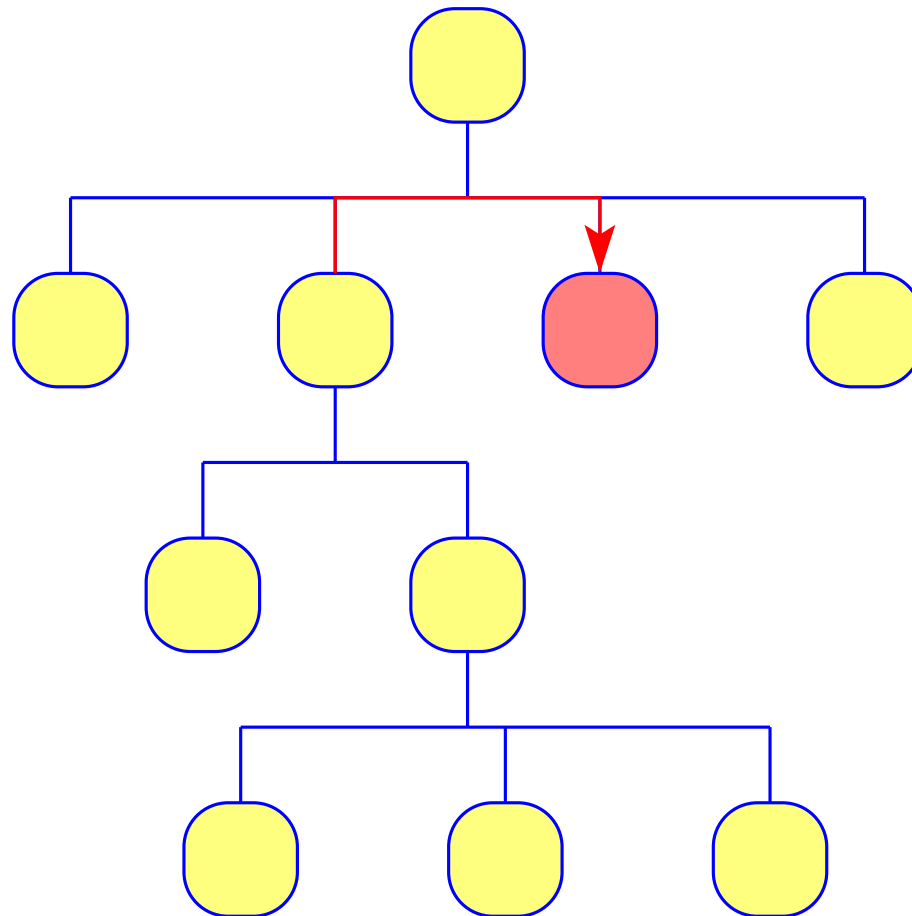
Tree walking automaton:



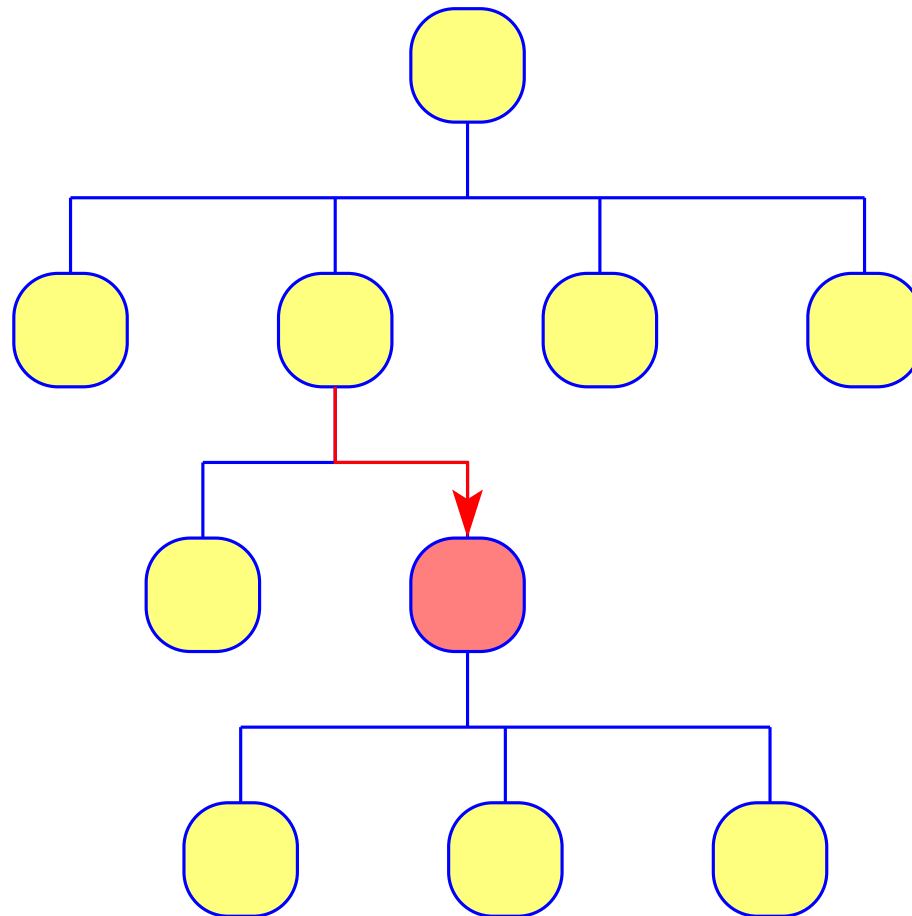
Tree walking automaton:



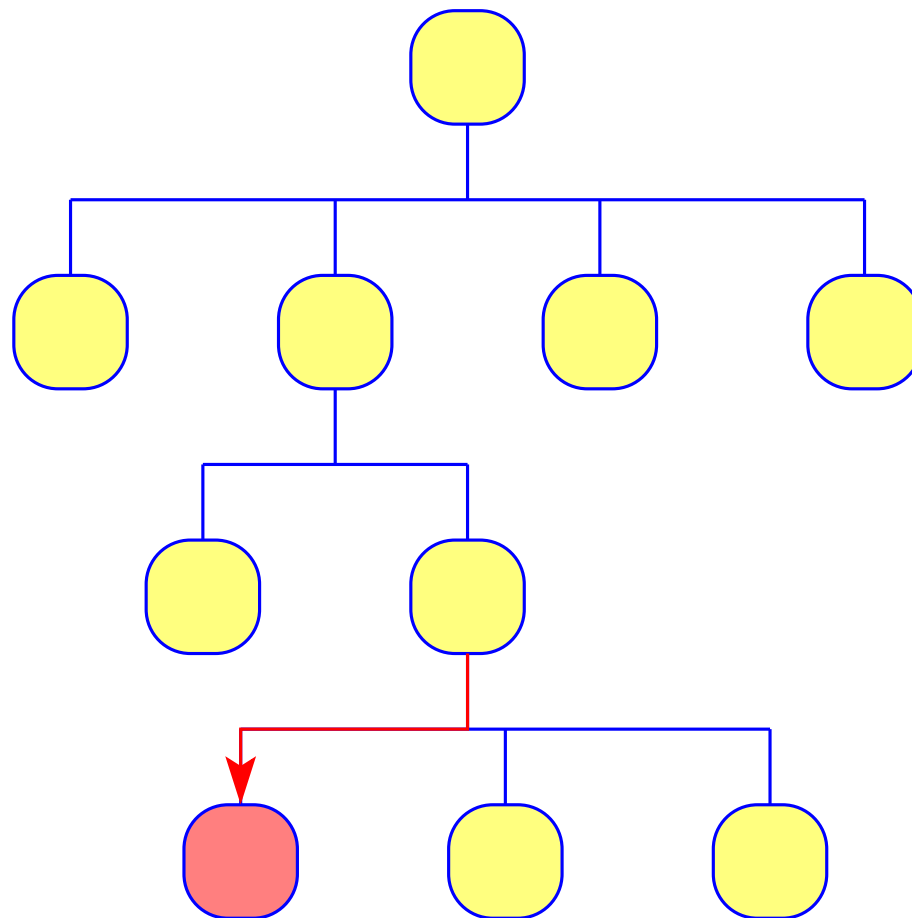
Tree walking automaton:



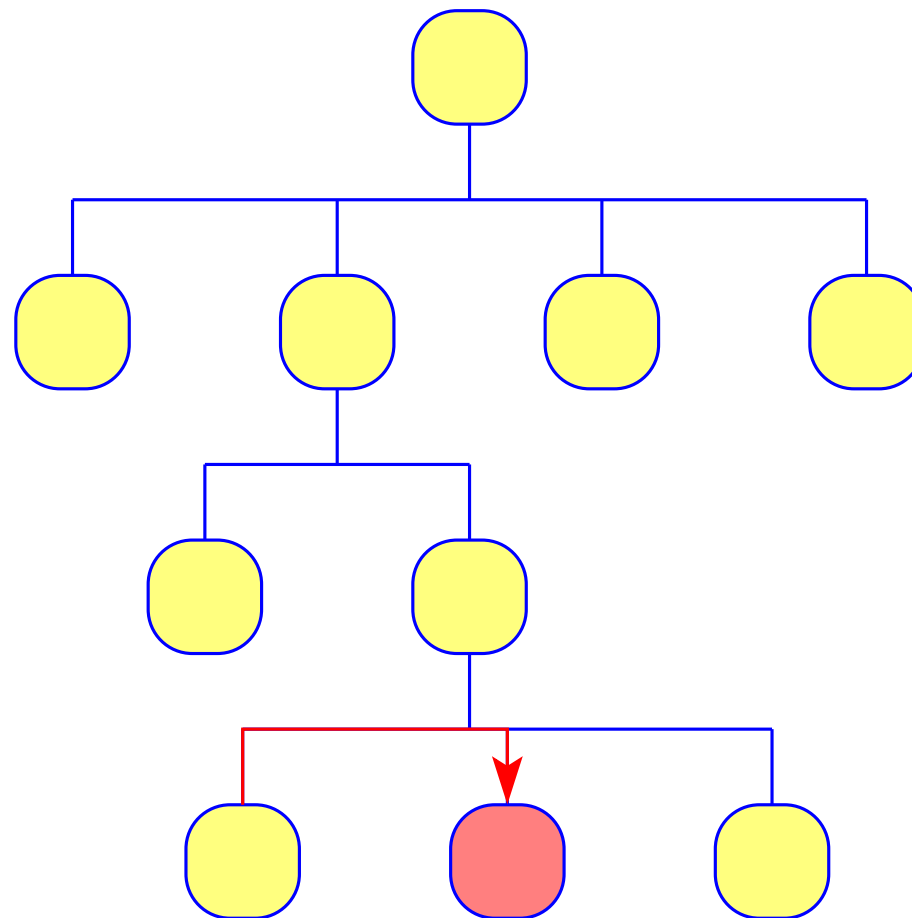
Tree walking automaton:



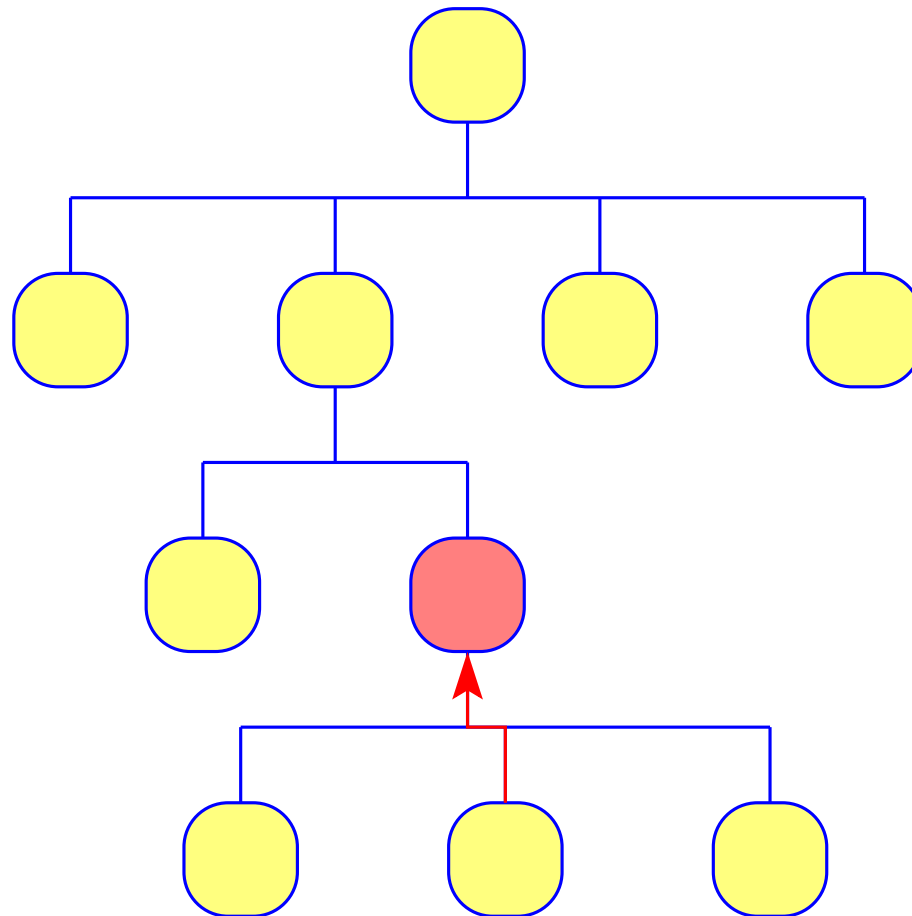
Tree walking automaton:



Tree walking automaton:



Tree walking automaton:



... always reads the original labels

⇒ catapillar

Brüggemann-Klein/Wood

... always reads the original labels

\implies catapillar

Brüggemann-Klein/Wood

... leaves state at ancestors

\implies general

Kamimura/Slutzki 1981

... always reads the original labels

\implies catapillar

Brüggemann-Klein/Wood

... leaves state at ancestors

\implies general

Kamimura/Slutzki 1981

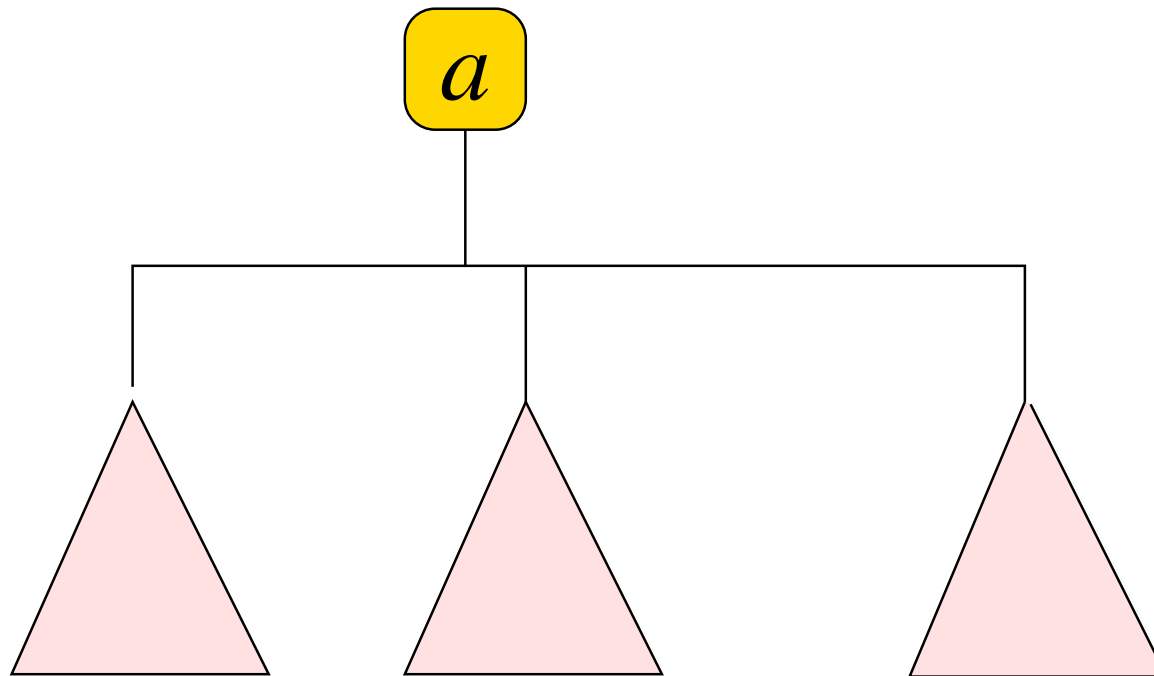
... leaves state at ancestors +
performs a single left-to-write scan

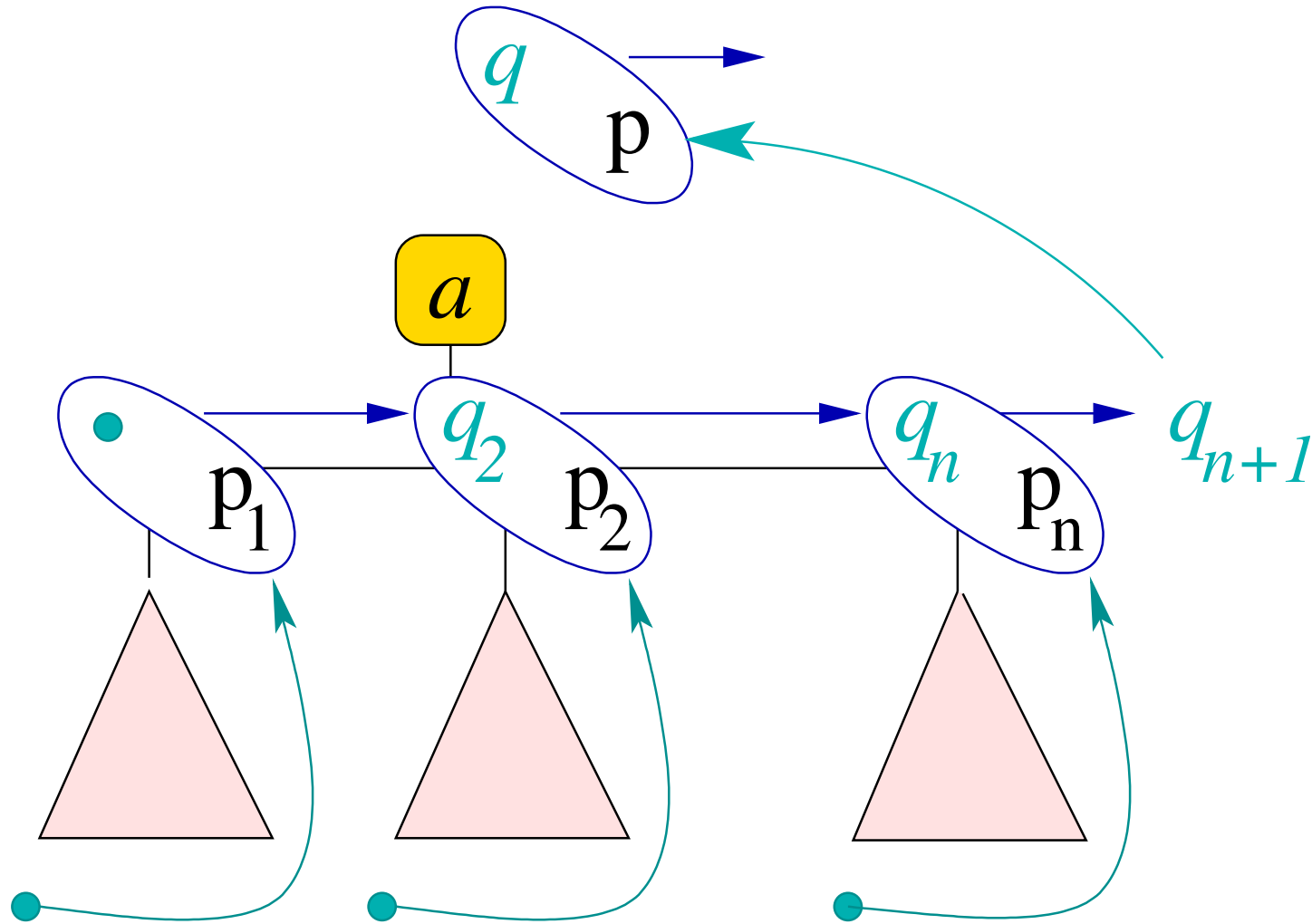
\implies pushdown tree automaton

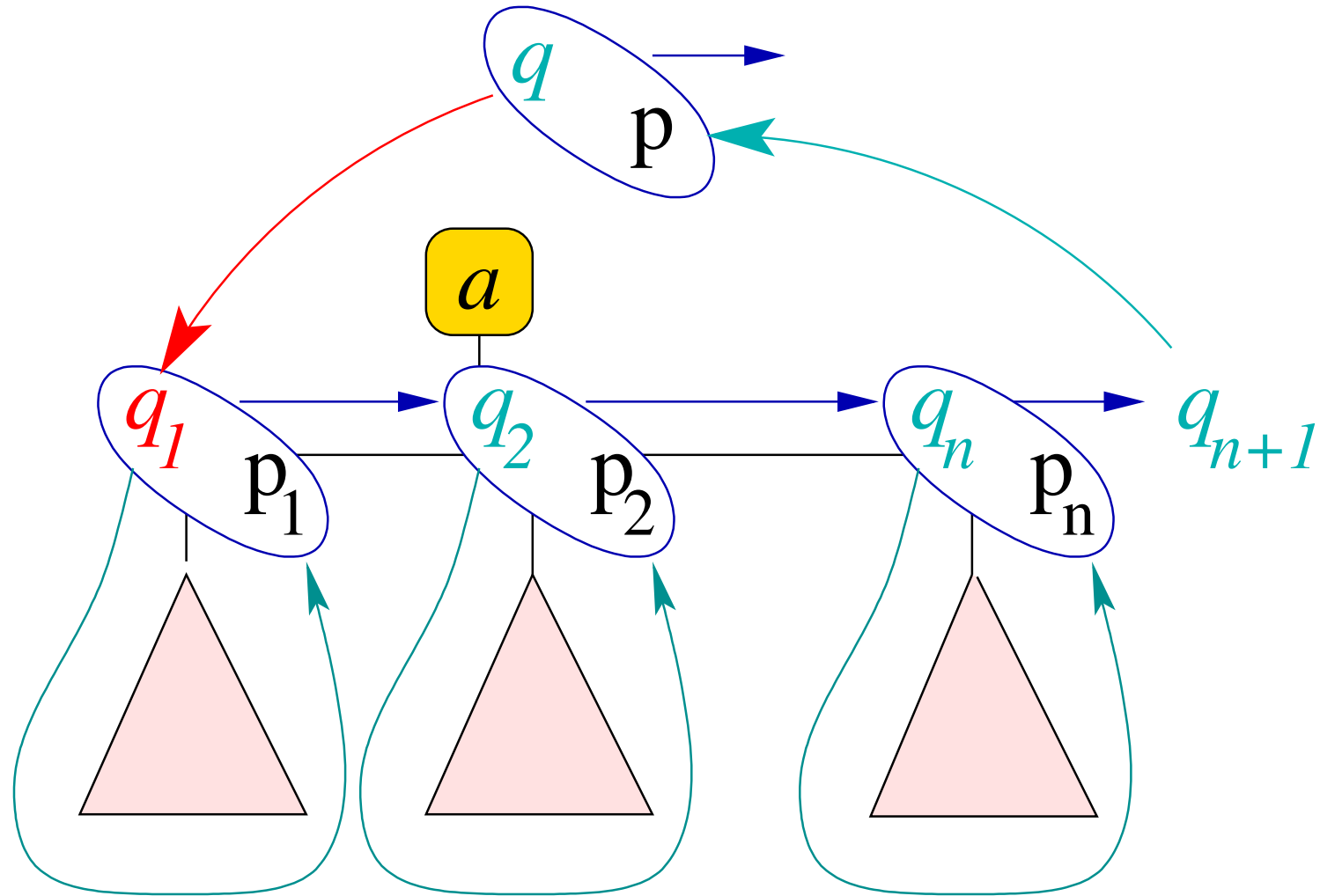
Neumann/S. 1998

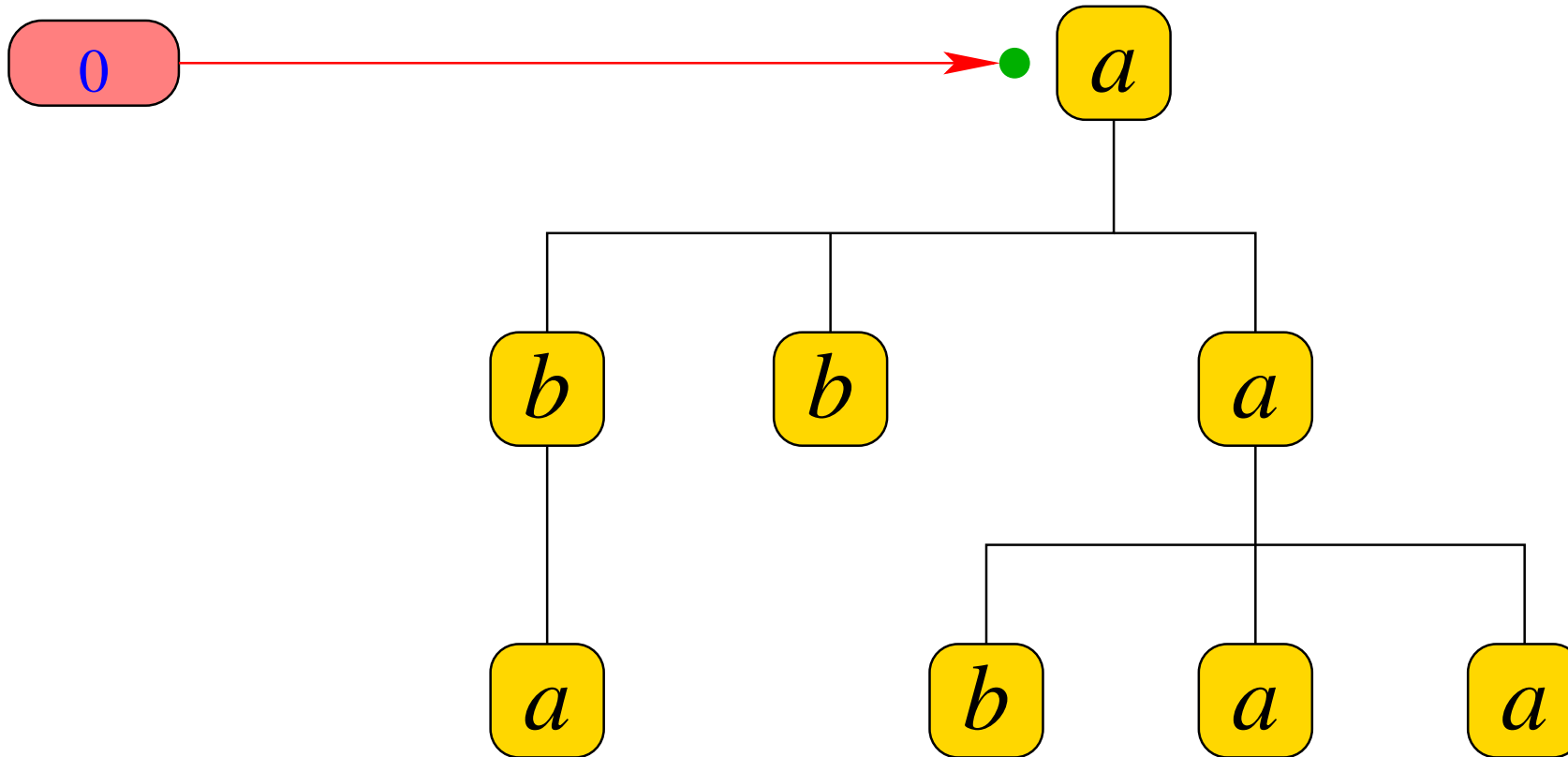
- ◇ Ordinary tree automata are **recursive** procedures ...
- ◇ perform a **post-order** traversal over the tree ...
 - ⇒ use a pushdown :-)
 - ⇒ but use it just to resume computation :-)
- ◇ Information is propagated
 - **left-to-right** and
 - **bottom-up**.

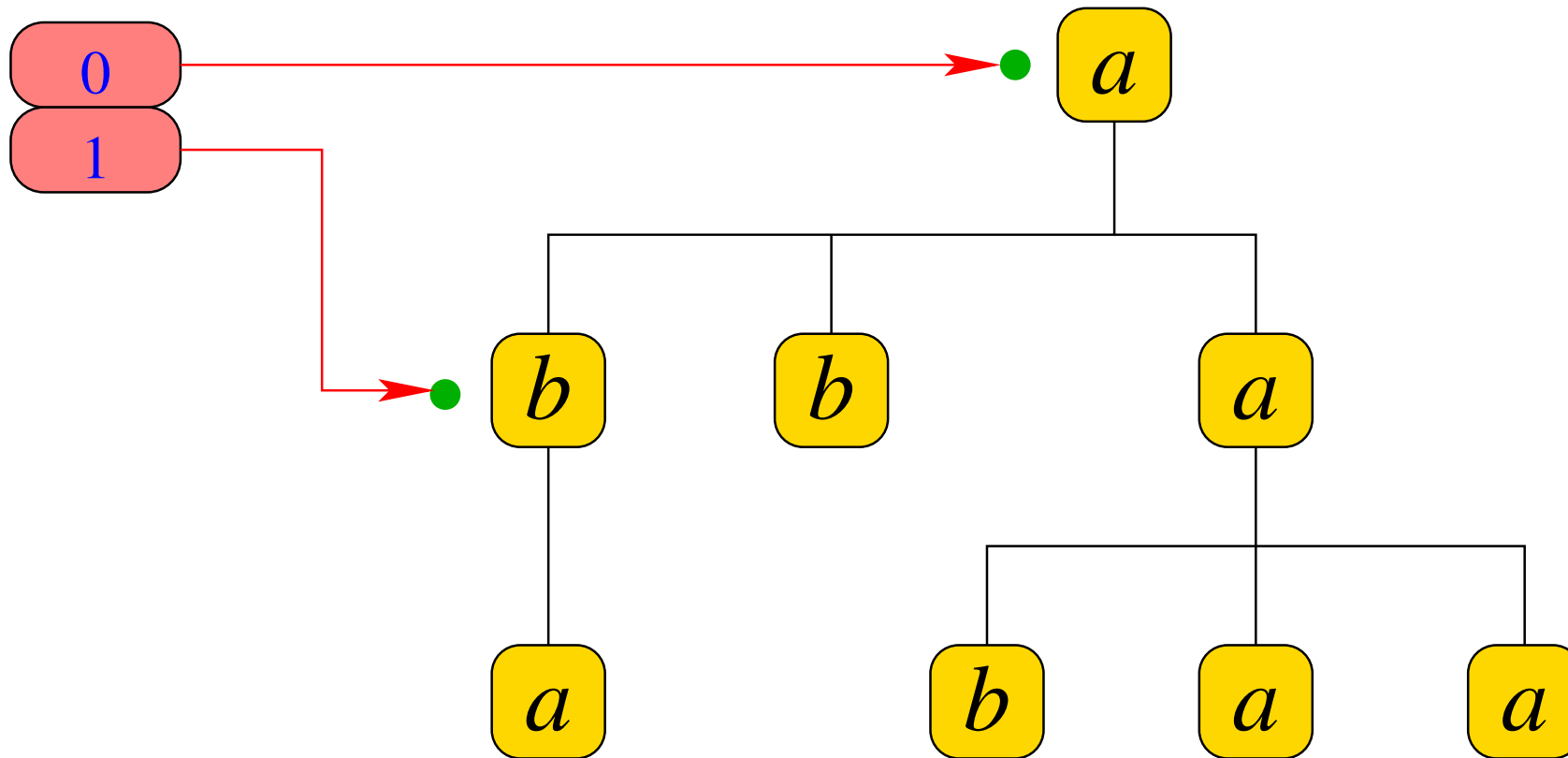
- ◇ Ordinary tree automata are **recursive** procedures ...
- ◇ perform a **post-order** traversal over the tree ...
 - ⇒ use a pushdown :-)
 - ⇒ but use it just to resume computation :-)
- ◇ Information is propagated
 - **left-to-right** and
 - **bottom-up**.
- Idea:** Propagate information also **downward** ...
 - ⇒ pushdown tree automata

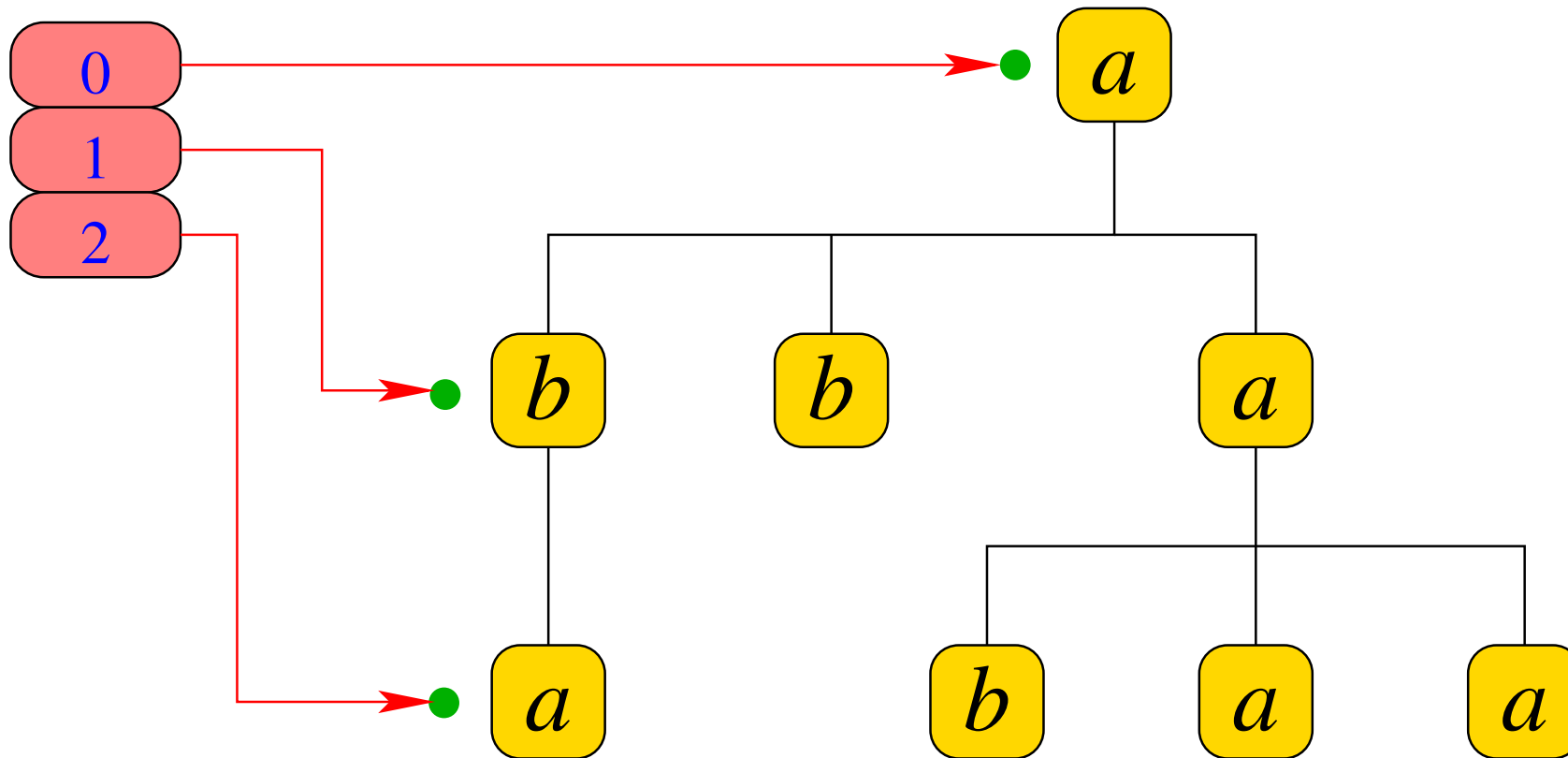


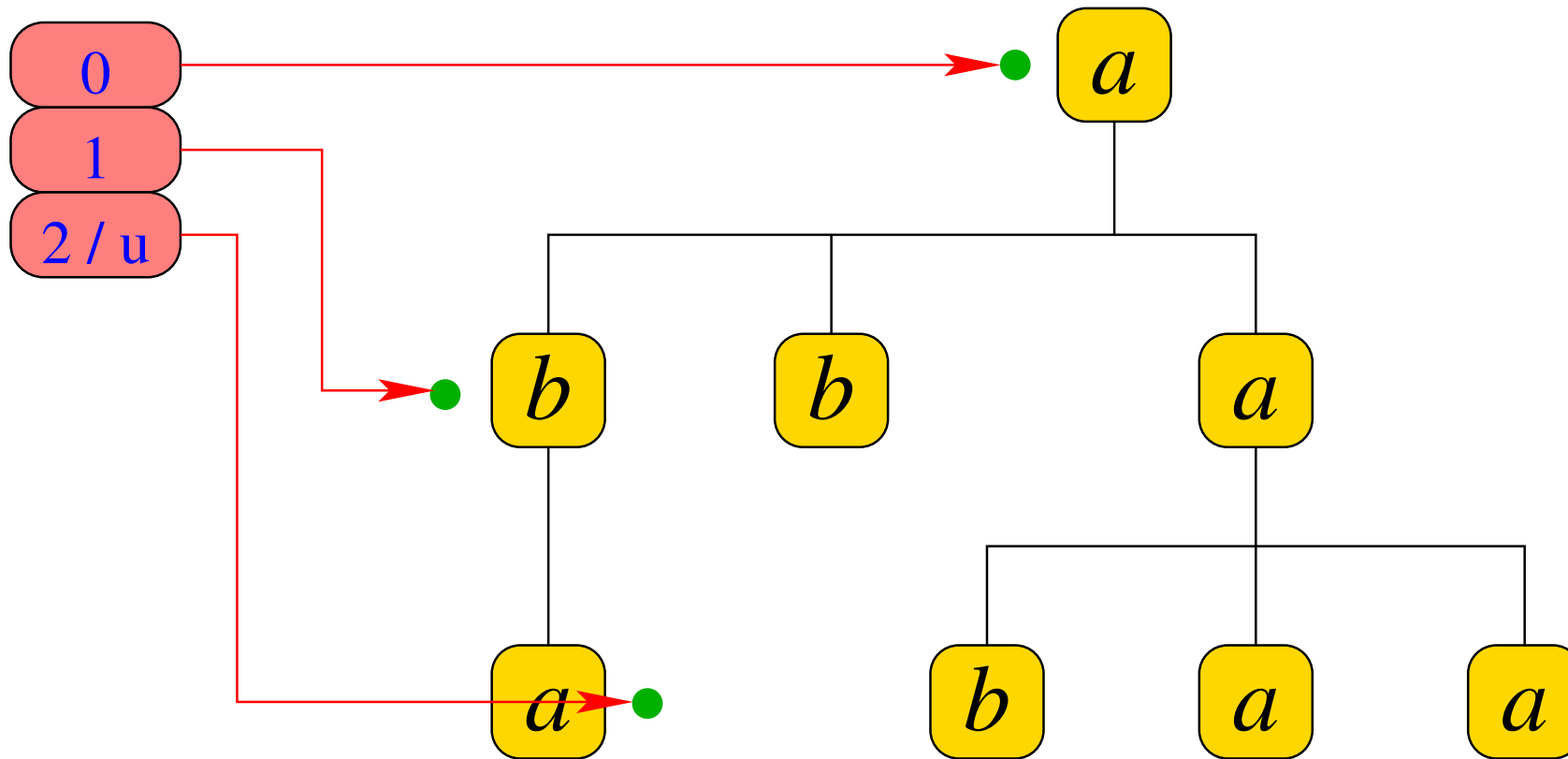


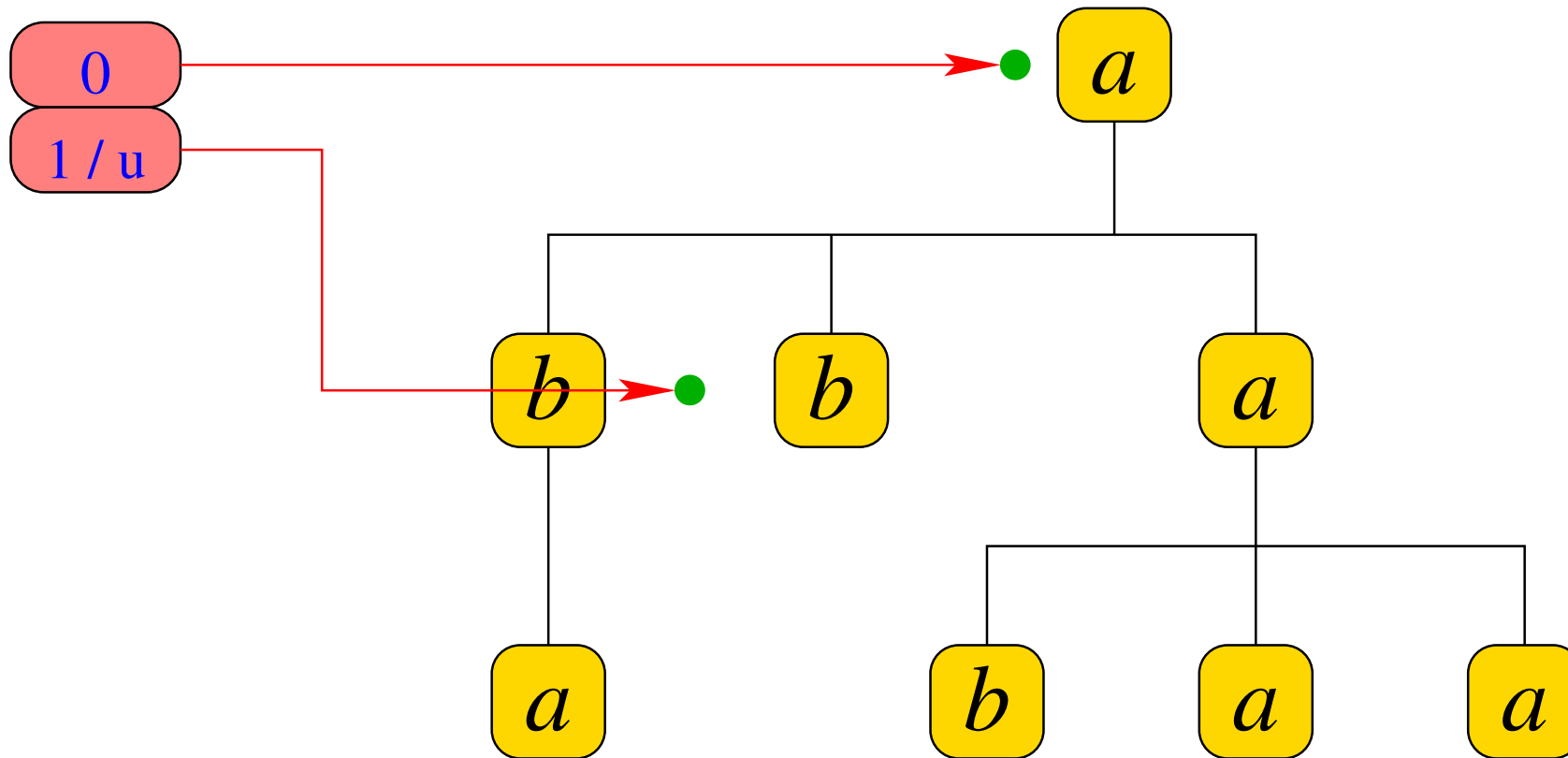


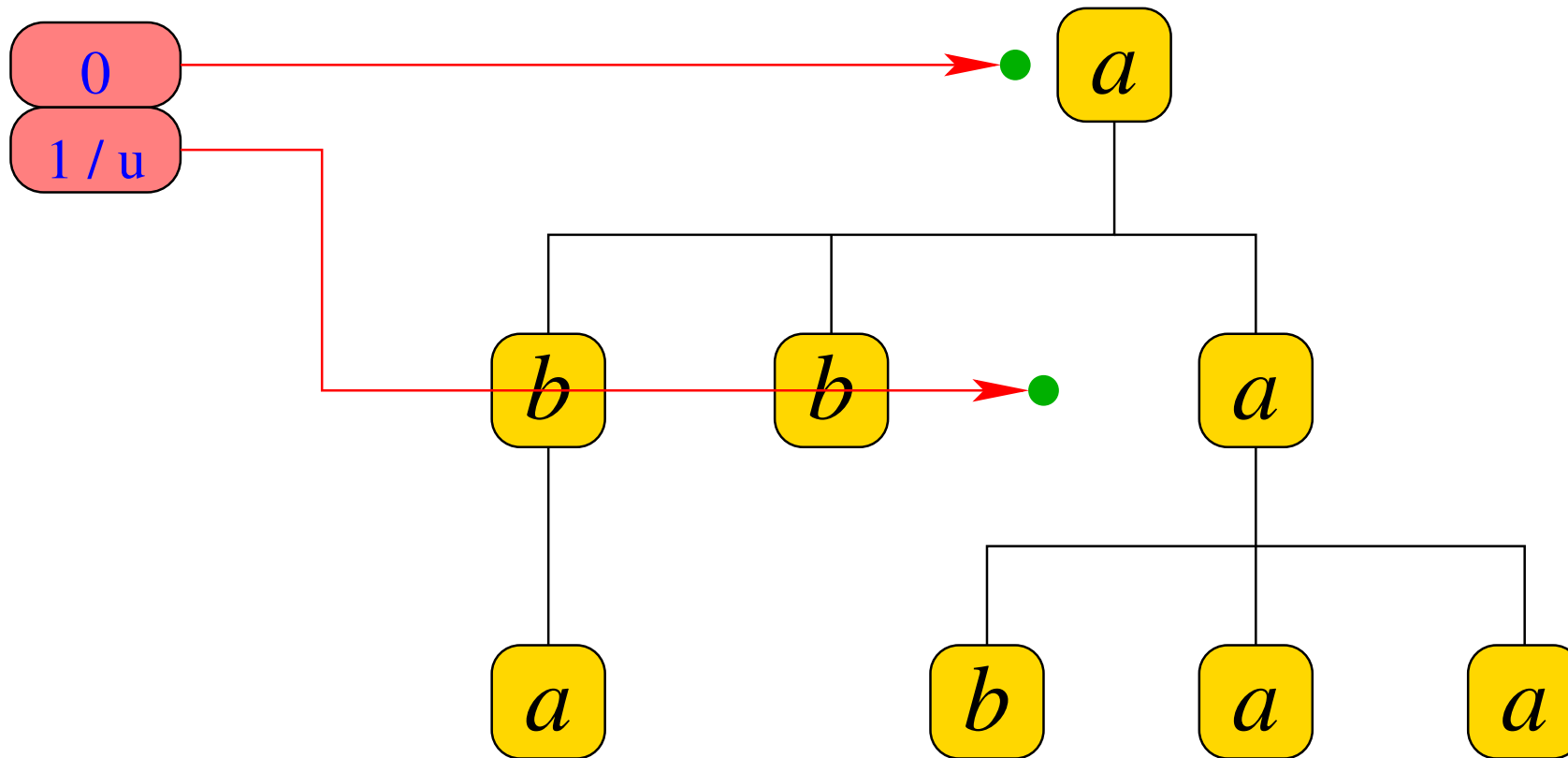


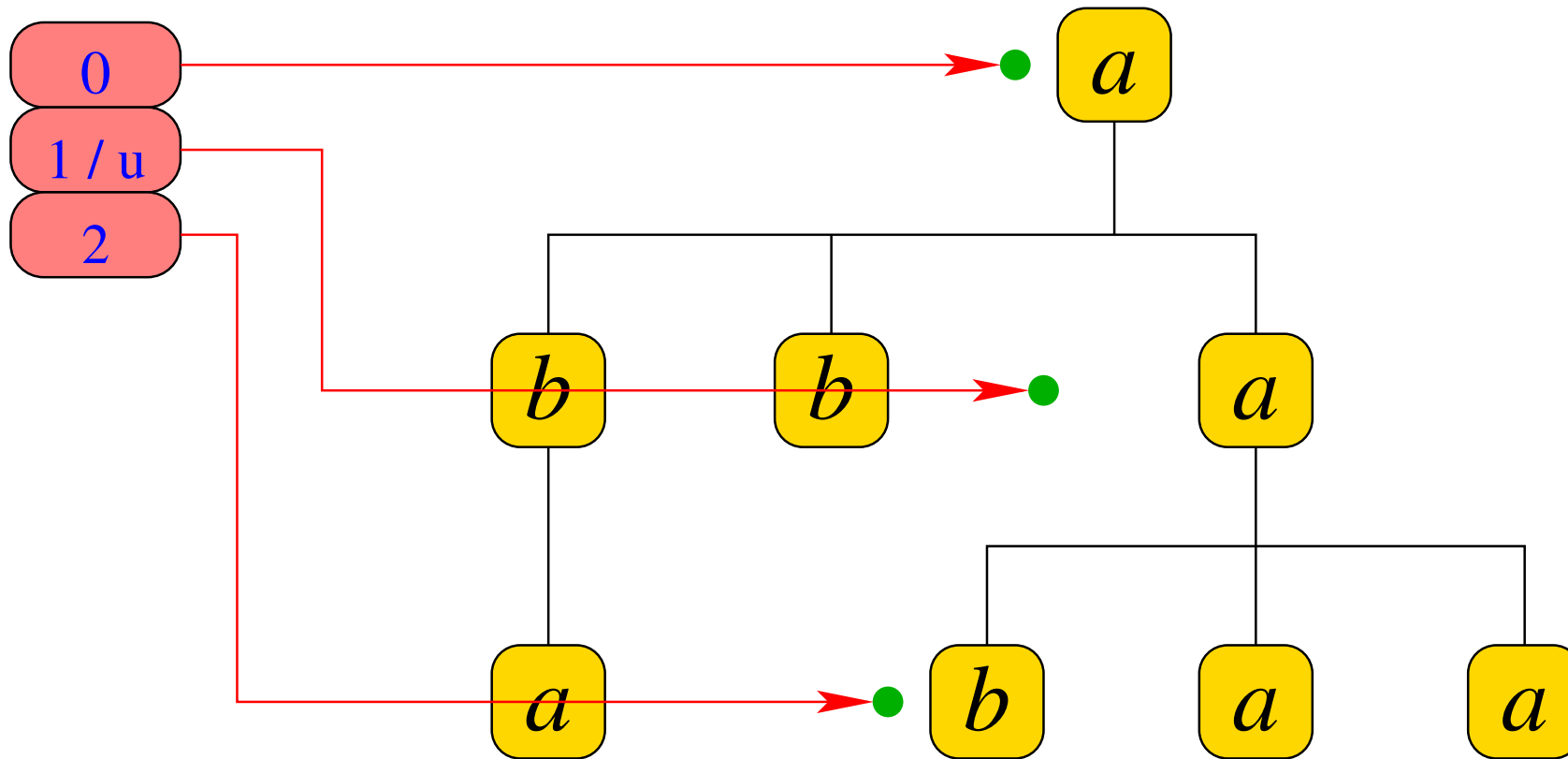


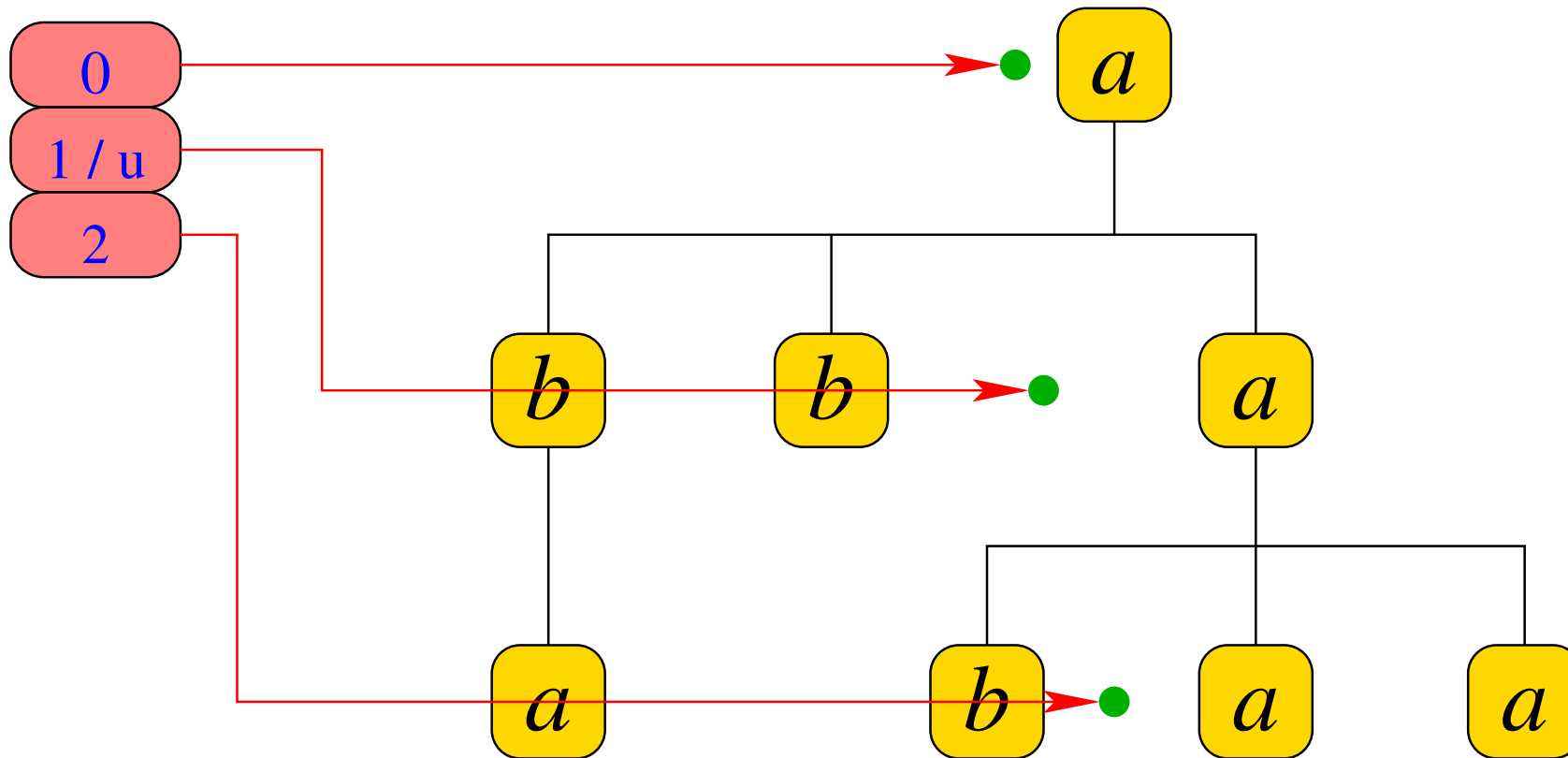


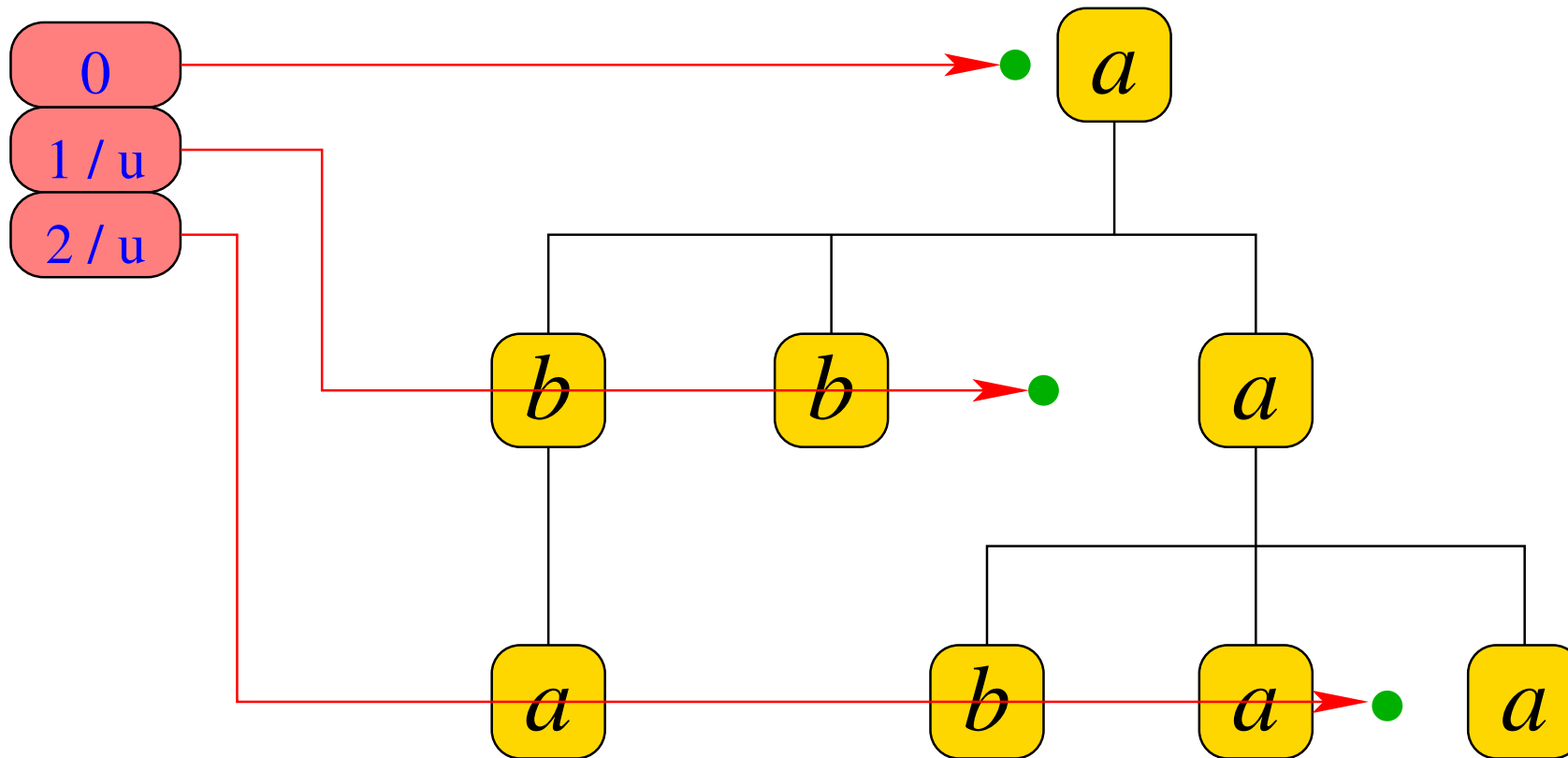


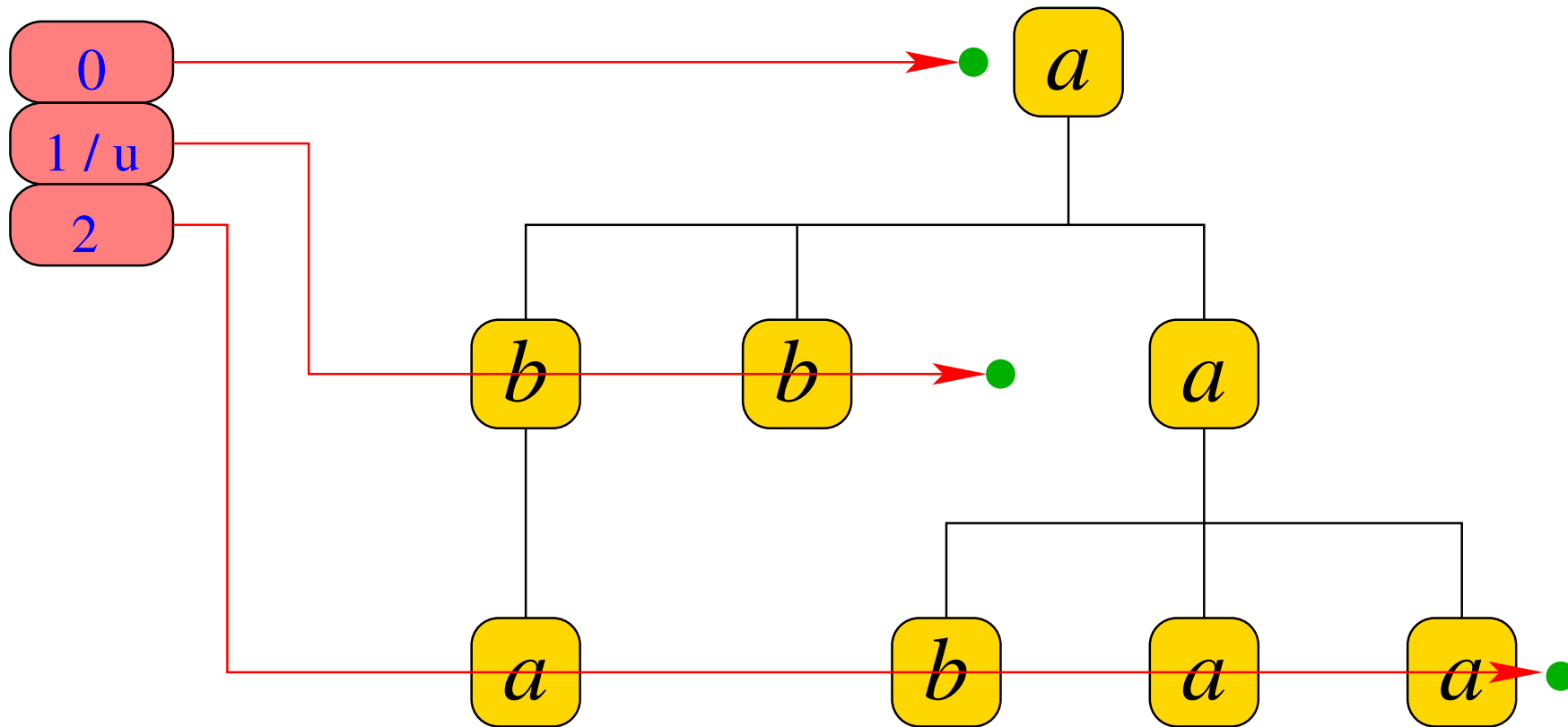


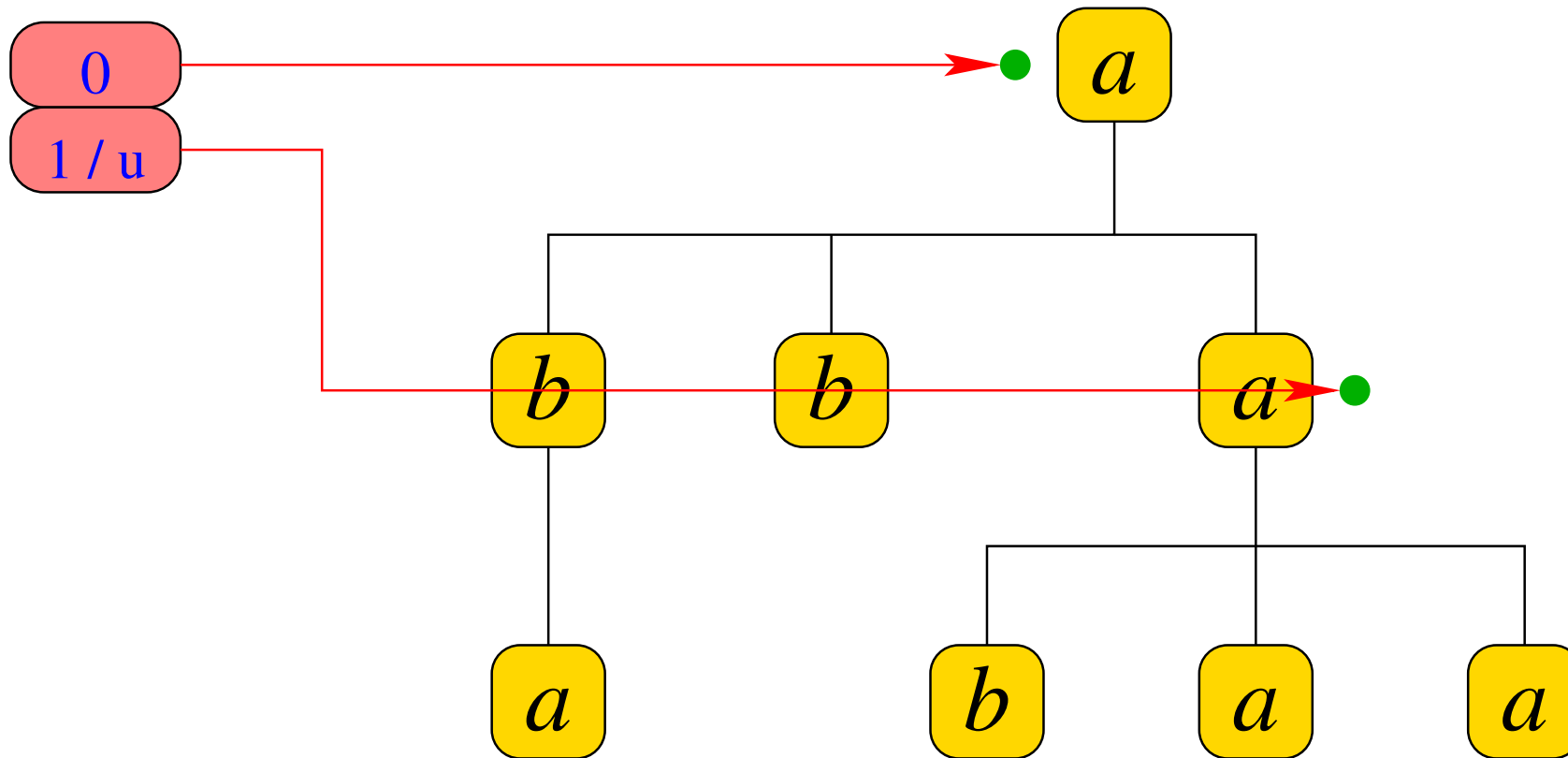


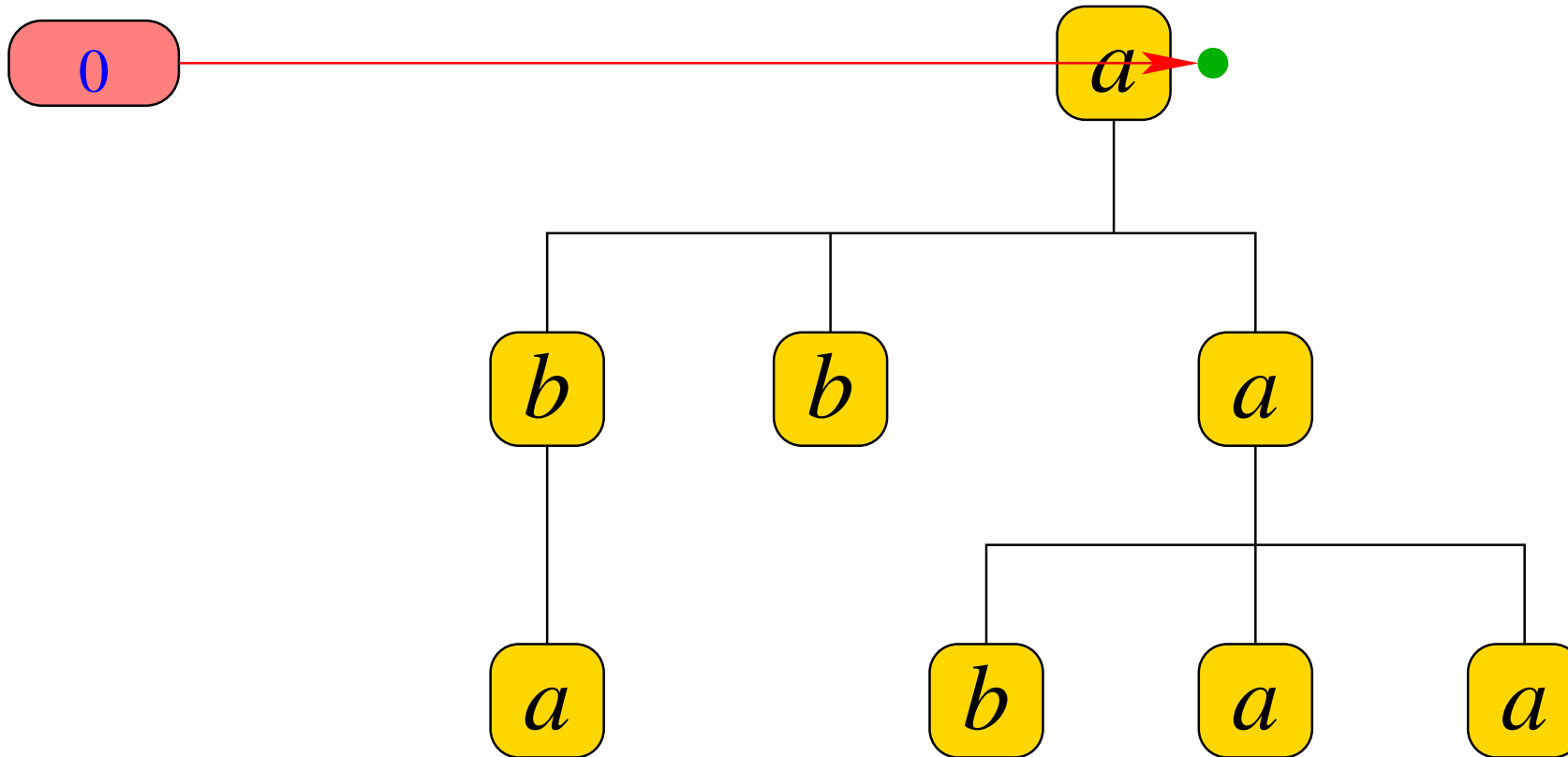












Expressiveness:

catapillars \subseteq tree walking
= finite automata

Expressiveness:

catapillars \subseteq tree walking
= finite automata

Emptiness:

Model	Complexity
catapillars	PSPACE hard
tree walking	DEXPTIME complete
tree pushdown	quadratic
finite automaton	linear

General Idea: powerset construction

⇒ works for **bottom-up** automata

⇒ is inherently **exponential**

General Idea: powerset construction

⇒ works for bottom-up automata

⇒ is inherently exponential

Costs:

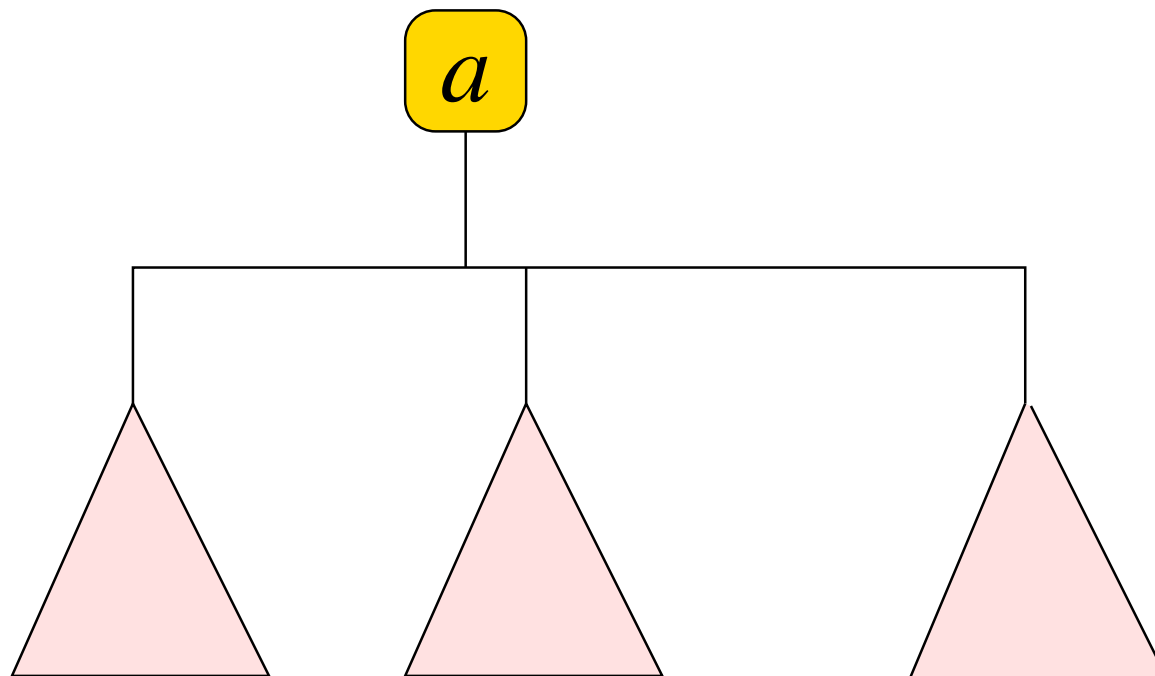
finite automata	2^n
catapillars	???
tree walking	2^{n^2}

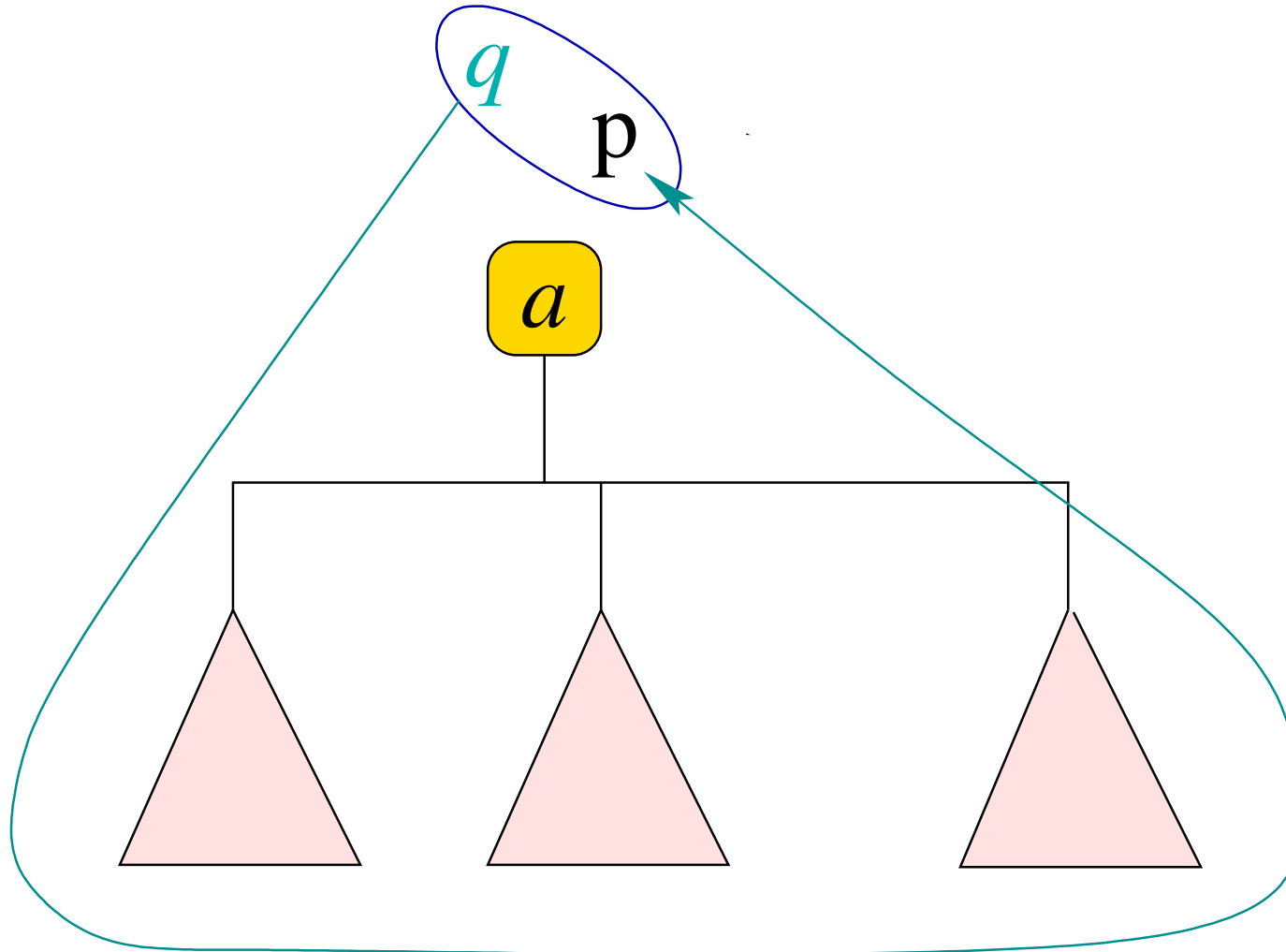
Remarks:

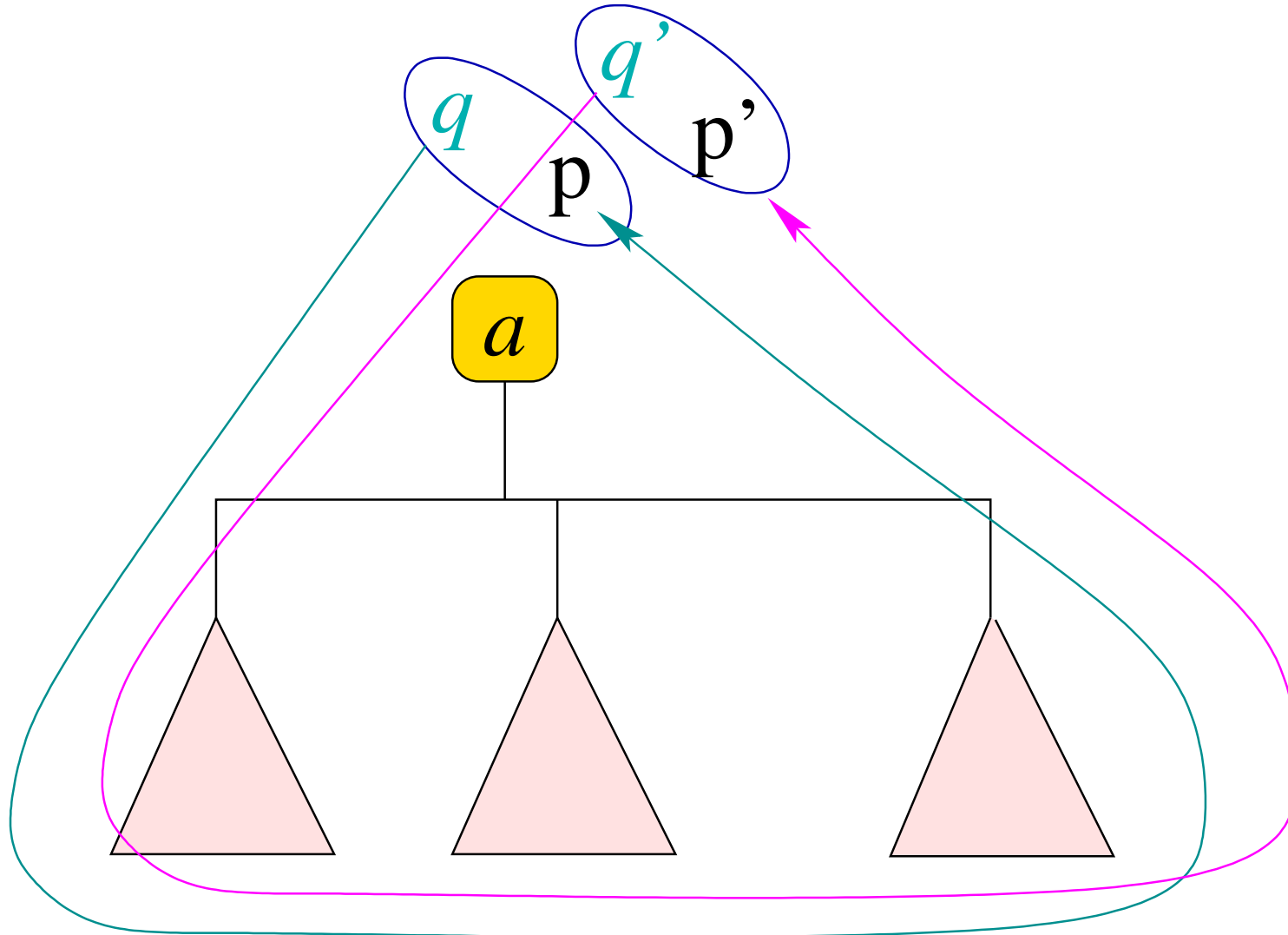
- ◇ Determinization also works for **alternating** push-down tree automata :-)
- ◇ When starting from a grammar, deterministic push-down tree automata only may have size 2^n .

Idea for pushdown tree automata:

- ◇ subset construction ... **but**
- ◇ take care that **down**- and **up**-moves match:





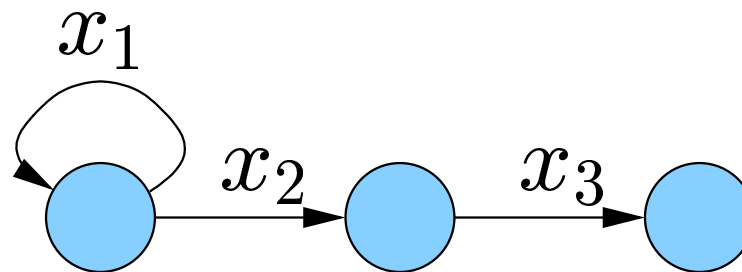


Remark:

Determinization preserves traversals :-)

Idea:

→ translate regular expression $r = x_1^*x_2x_3$
into finite automaton A_r :



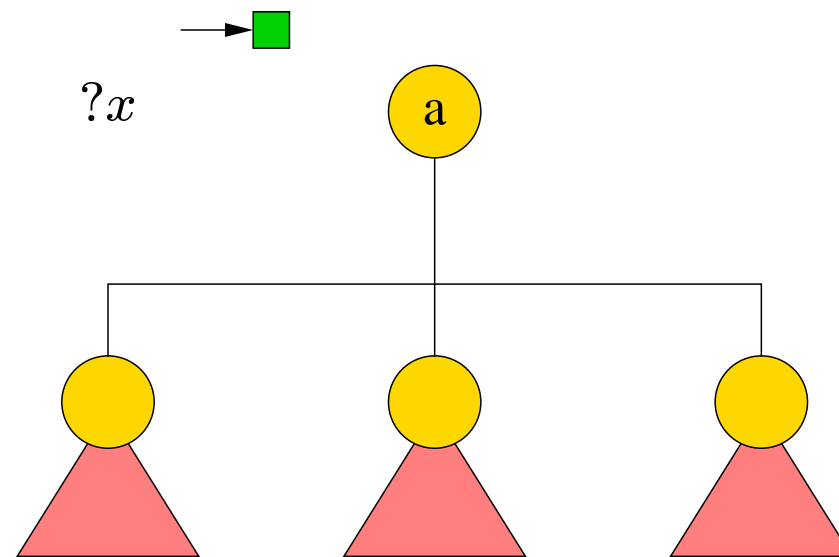
⇒ horizontal forest transitions consume
non-terminals :-)

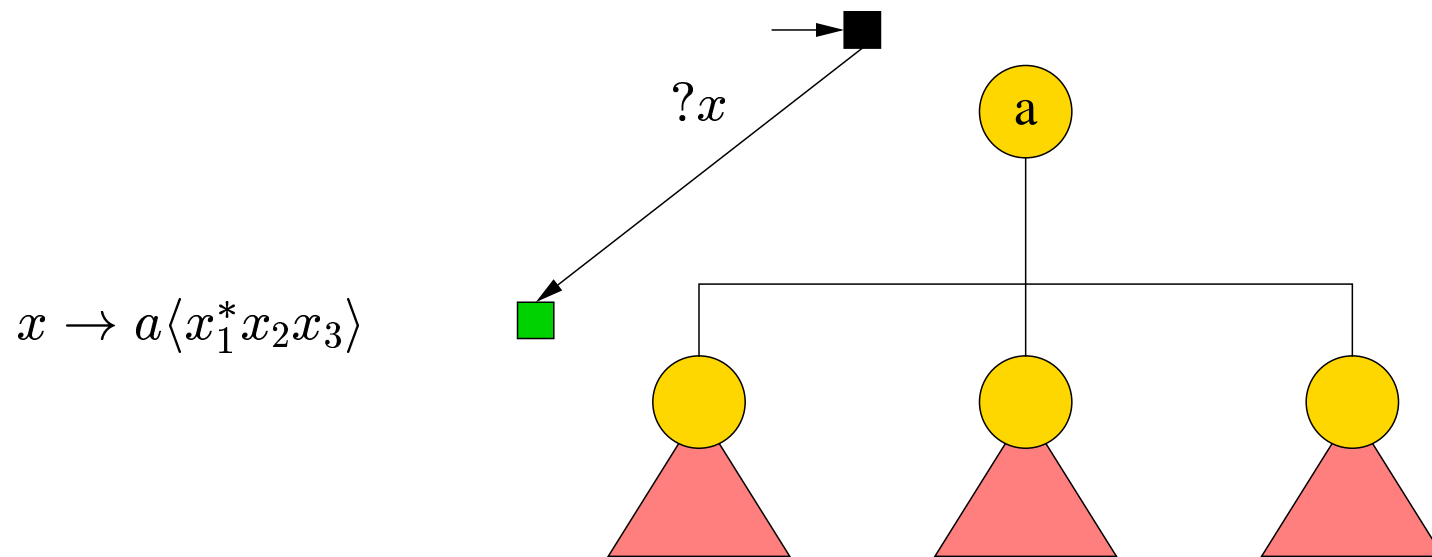
→ translate rule $x \rightarrow a\langle r \rangle$ into **down**- and **up**-transitions:

$$(q, a, q_0) \in \text{down} \quad (q_f, a, x) \in \text{up}$$

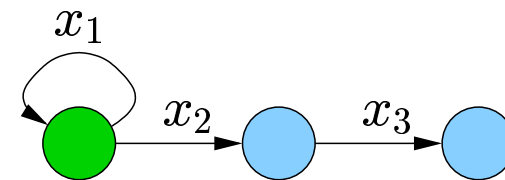
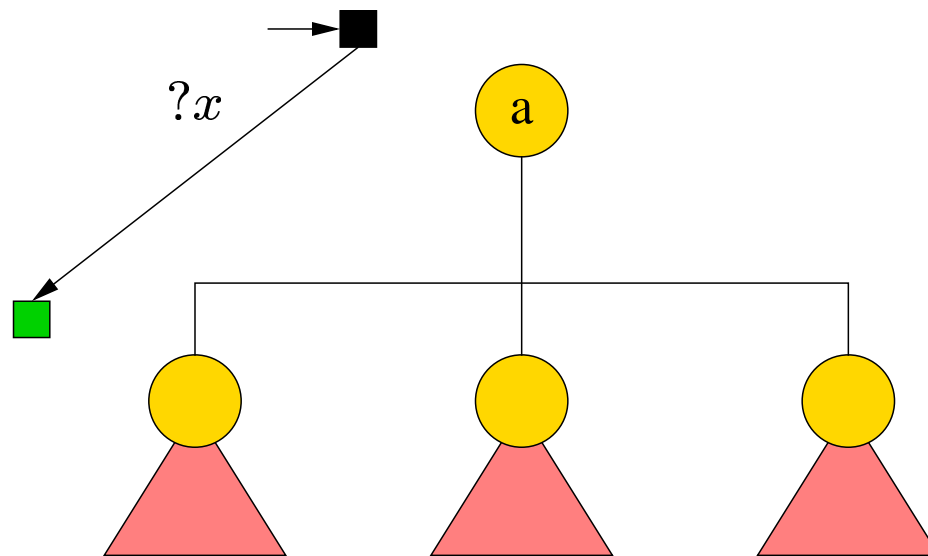
where:

q	$=$	forest state with x -transition;
q_0	$=$	initial state of A_r ;
q_f	$=$	final state of A_r

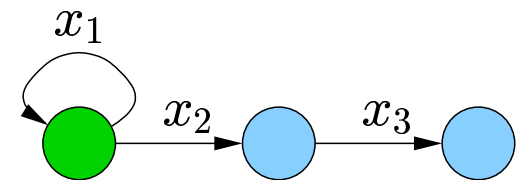
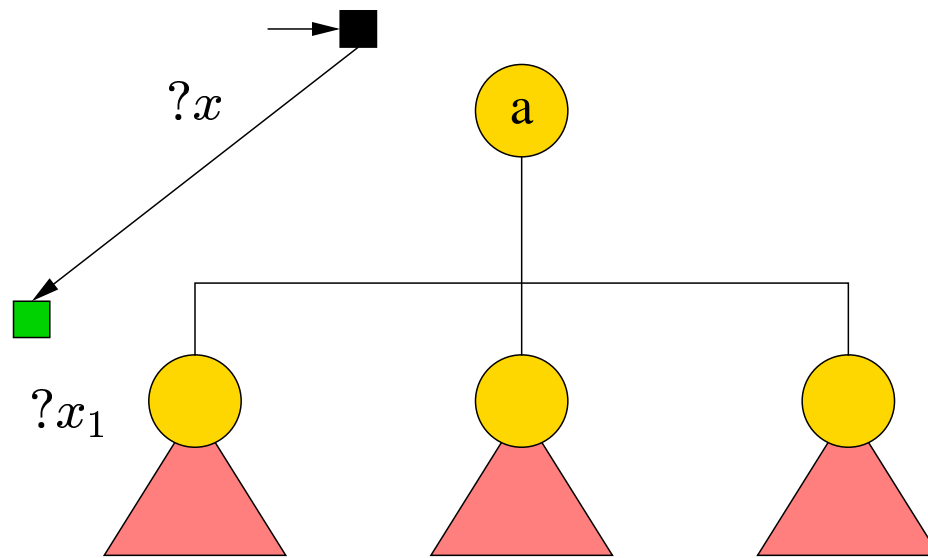




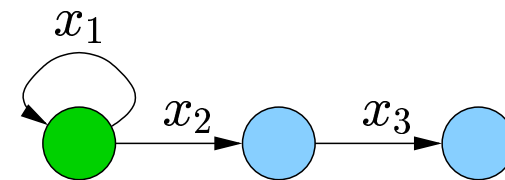
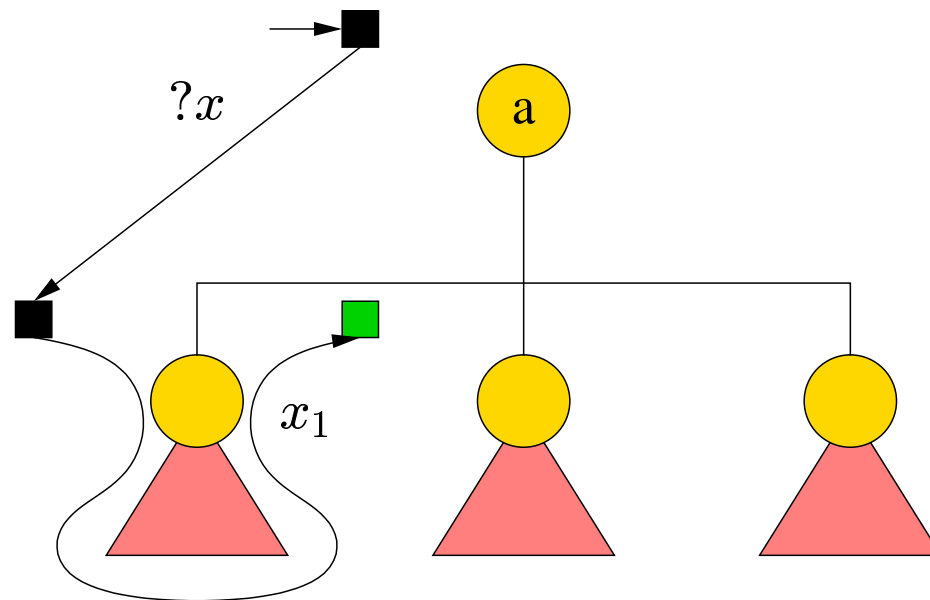
$$x \rightarrow a \langle x_1^* x_2 x_3 \rangle$$



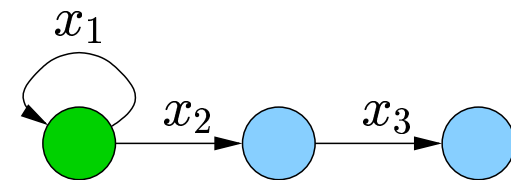
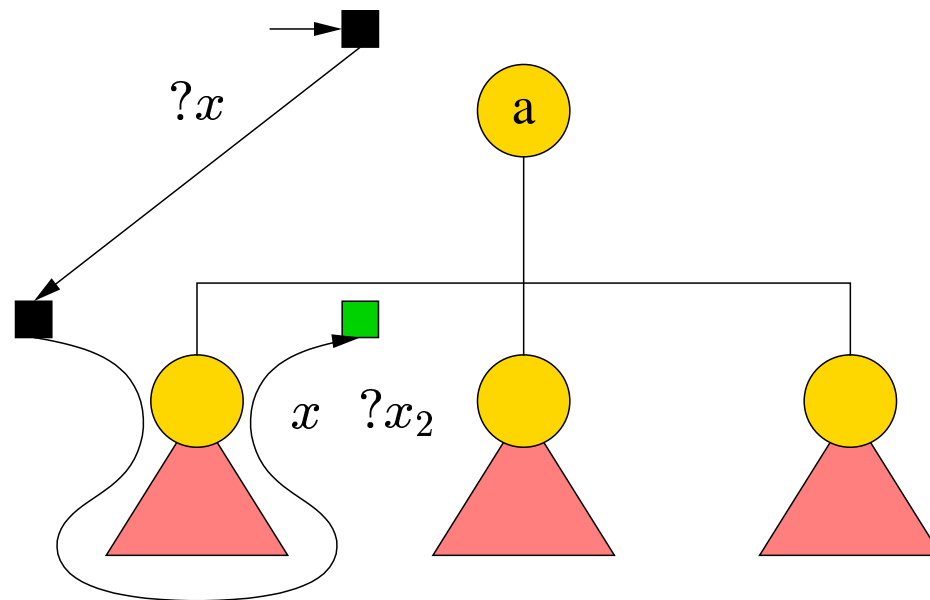
$$x \rightarrow a \langle x_1^* x_2 x_3 \rangle$$



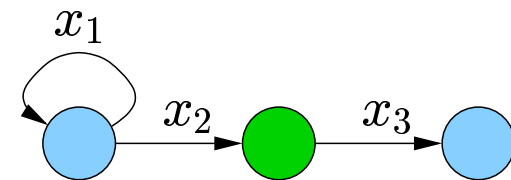
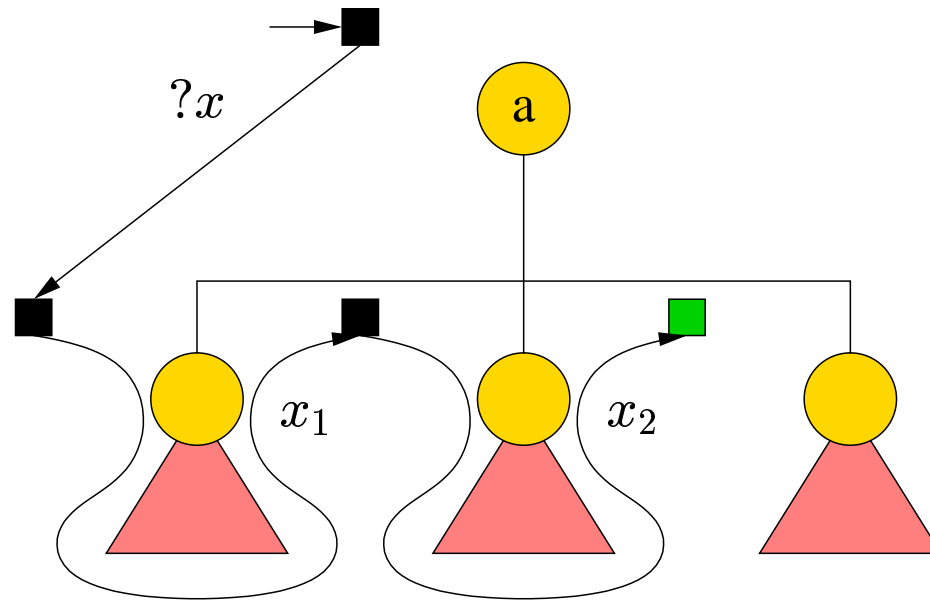
$$x \rightarrow a \langle x_1^* x_2 x_3 \rangle$$



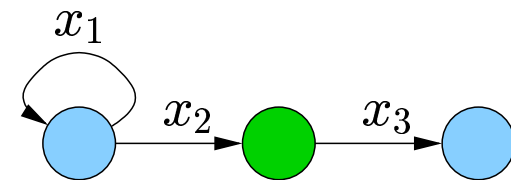
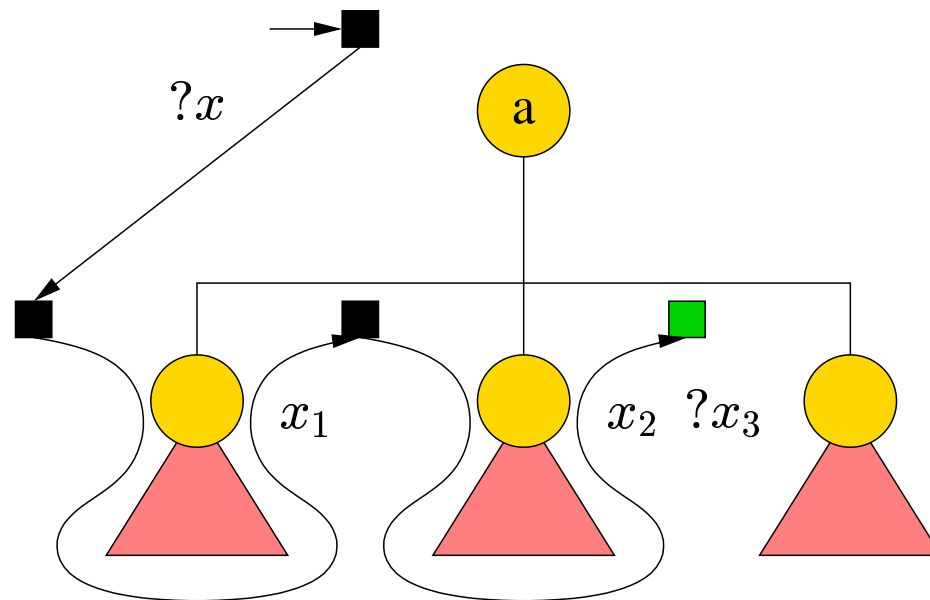
$$x \rightarrow a \langle x_1^* x_2 x_3 \rangle$$



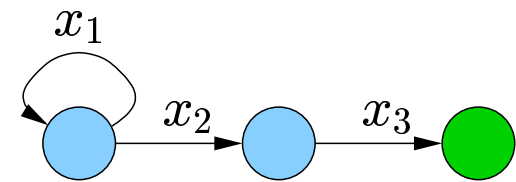
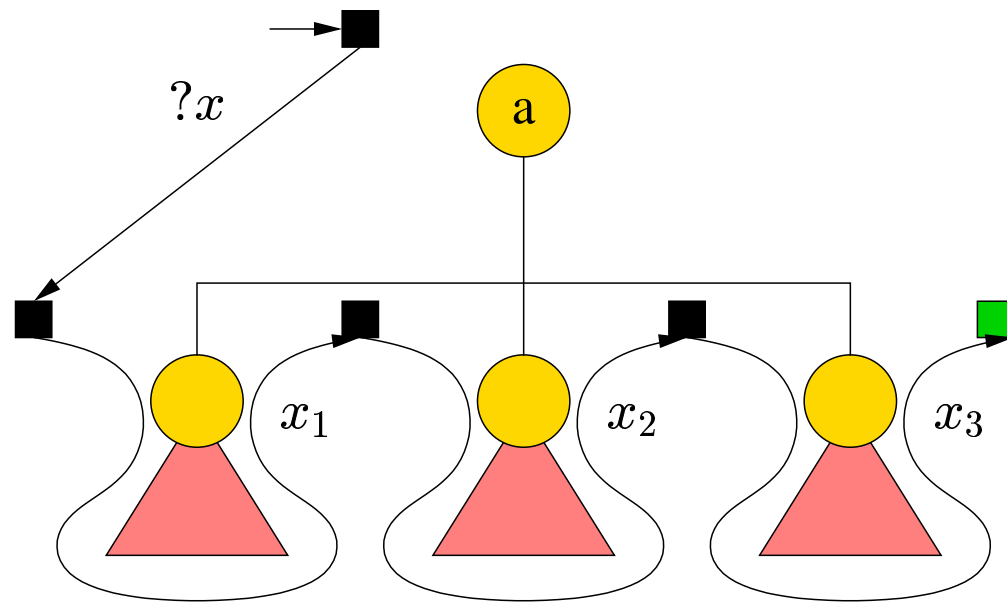
$$x \rightarrow a \langle x_1^* x_2 x_3 \rangle$$



$$x \rightarrow a \langle x_1^* x_2 x_3 \rangle$$



$$x \rightarrow a \langle x_1^* x_2 x_3 \rangle$$



Part II

Querying.

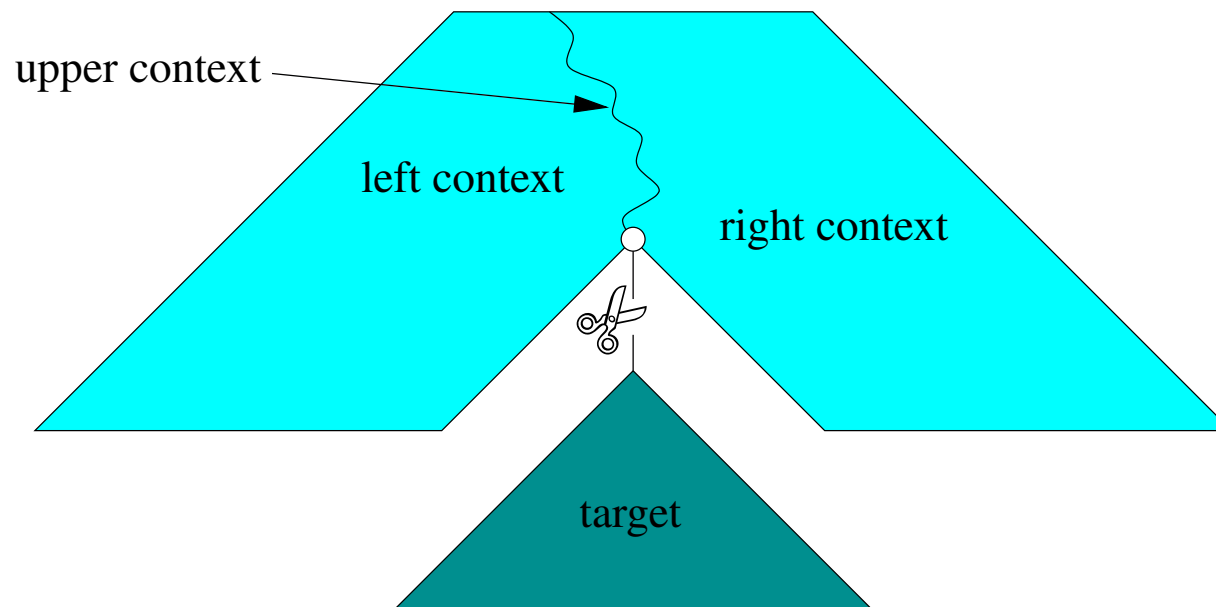
Querying a tree t \approx searching for tuples of subtrees

Querying a tree t	\approx	searching for tuples of subtrees
Monadic Querying	\approx	matching
	\approx	searching for subtrees

Querying a tree t	\approx	searching for tuples of subtrees
Monadic Querying	\approx	matching
	\approx	searching for subtrees
Binary Querying	\approx	selecting
	\approx	searching for next subtrees

Matching = Location of **sub-documents** which

- ◇ satisfy a **structural** condition (**what** ?)
- ◇ occur in a specified **context** (**where** ?)



Structure: a regular language ...

Structure: a regular language ...

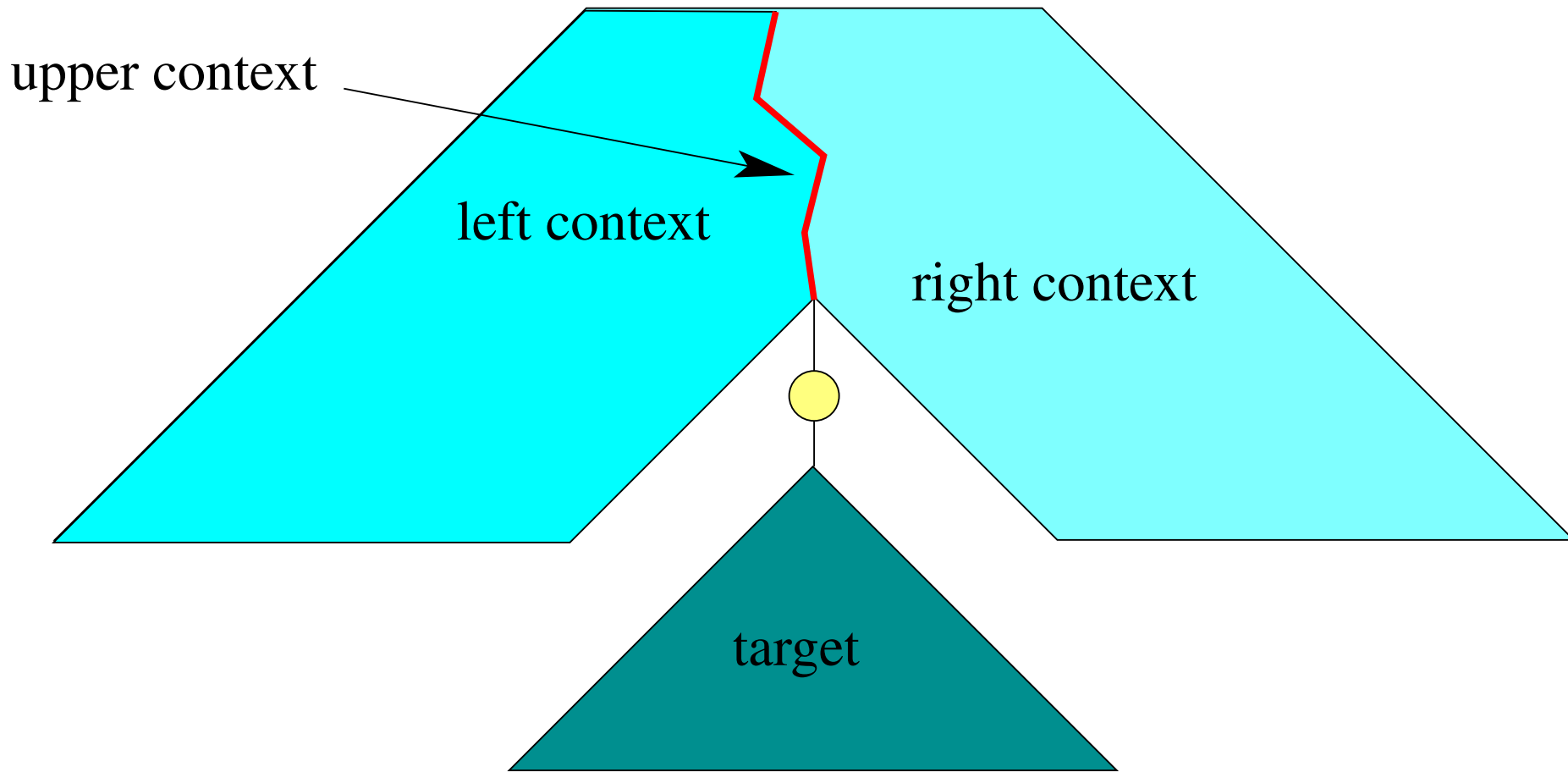
Context:

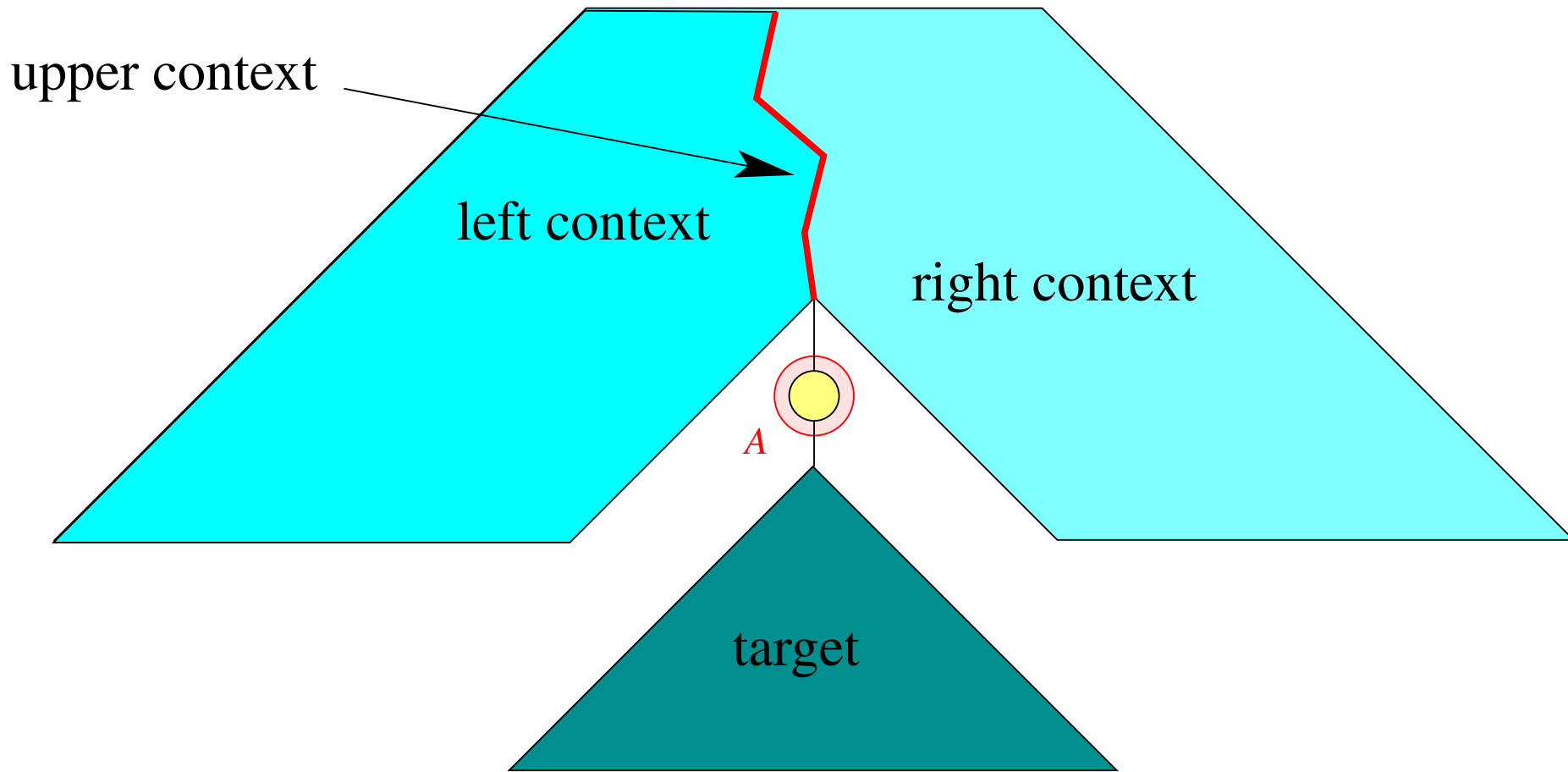
- ◇ Sequence of ancestors; more generally:
- ◇ Regular expression; more generally:

Structure: a regular language ...

Context:

- ◇ Sequence of ancestors; more generally:
 - ◇ Regular expression; more generally:
 - ◇ A tree grammar G together with a variable A :
 - G describes context plus structure
 - A describes structure relative to context
- ⇒ may speak about neighbors as well :-)





... more generally:

Structure + Context = grammar G with
Boolean combination of variables ...

... more generally:

Structure + Context = grammar G with
Boolean combination of variables ...

- allows conjunction / negations of structure;
- allows conjunction / negations of context :-)

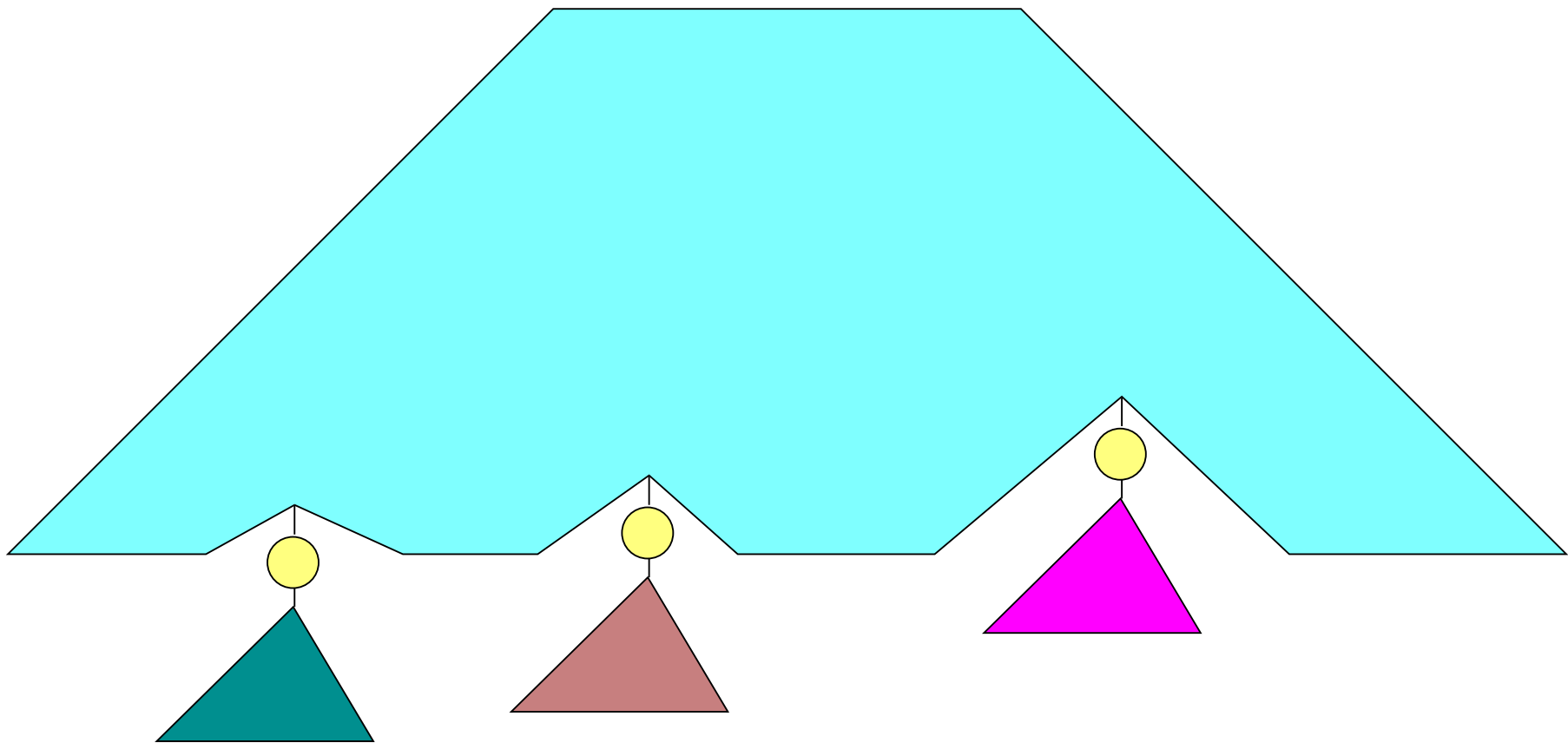
... more generally:

Structure + Context = grammar G with
Boolean combination of variables ...

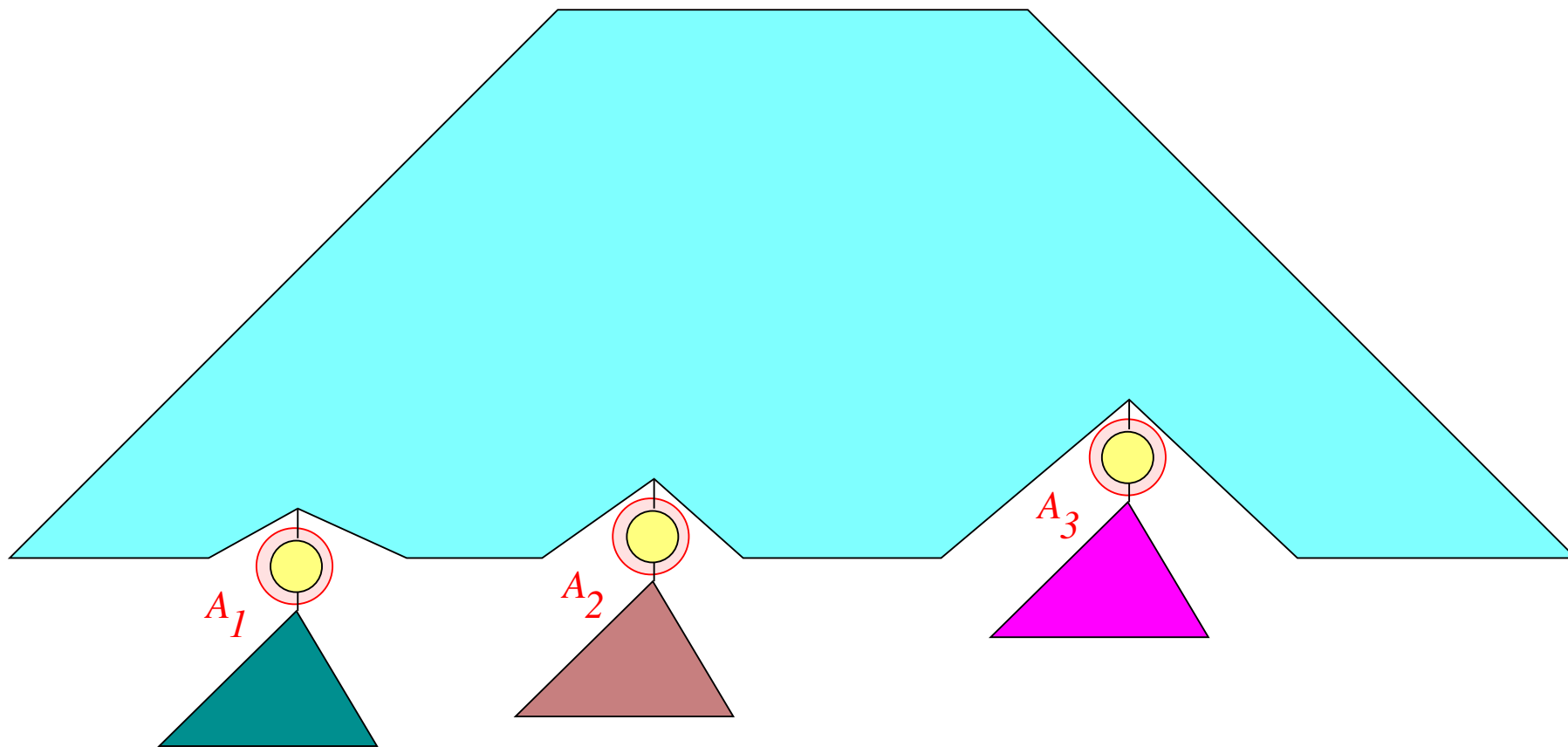
- allows conjunction / negations of structure;
- allows conjunction / negations of context :-)
- at no extra implementation cost :-))

Tree grammar G plus sequence A_1, \dots, A_k :

Tree grammar G plus sequence A_1, \dots, A_k :



Tree grammar G plus sequence A_1, \dots, A_k :



Idea: two **pushdown** tree automata

- ◇ The first runs **right-to-left**
⇒ checks right upper context and structure
- ◇ The second runs **left-to-right**
⇒ checks left context and signals match.

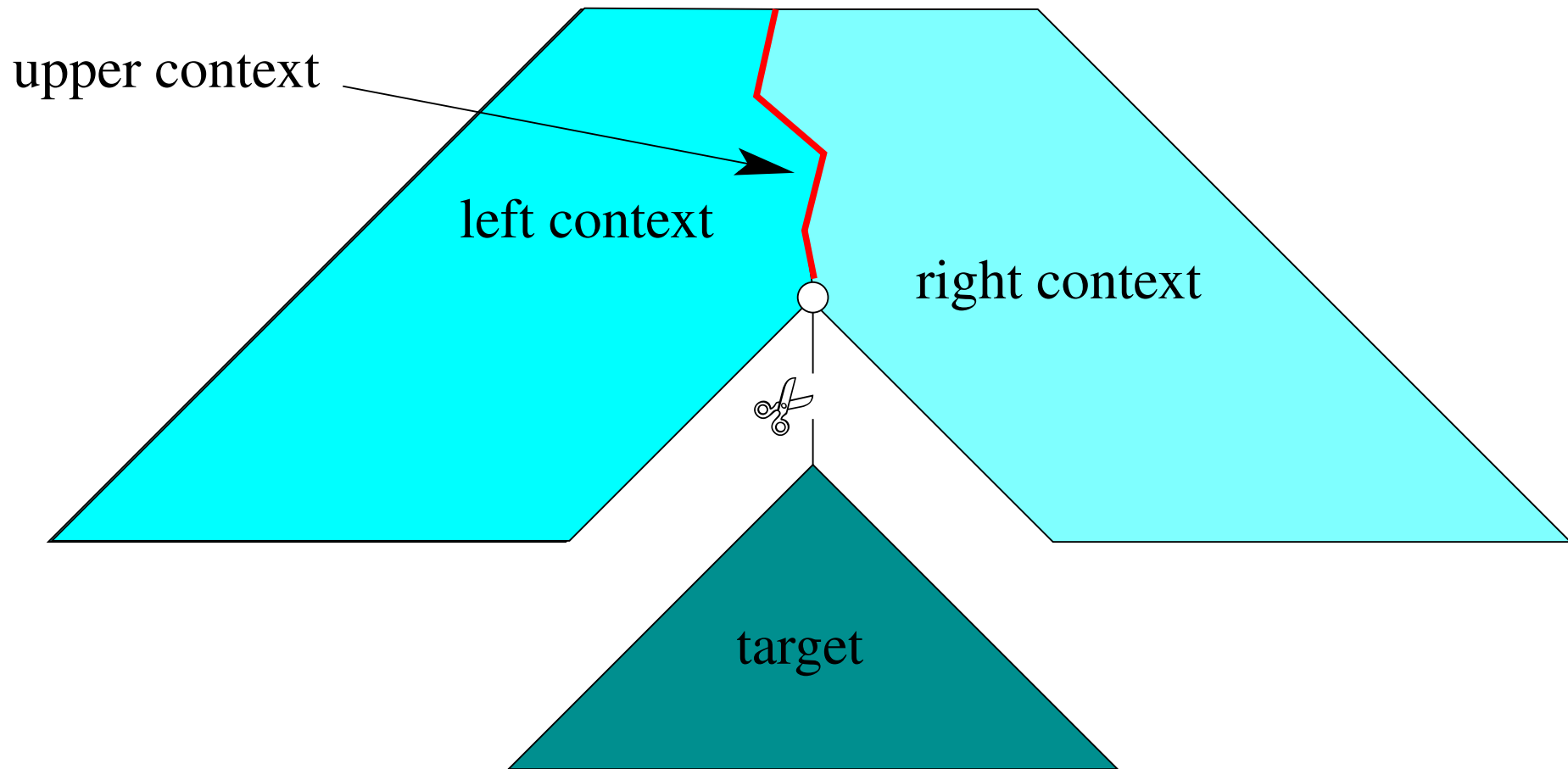
Idea: two **pushdown** tree automata

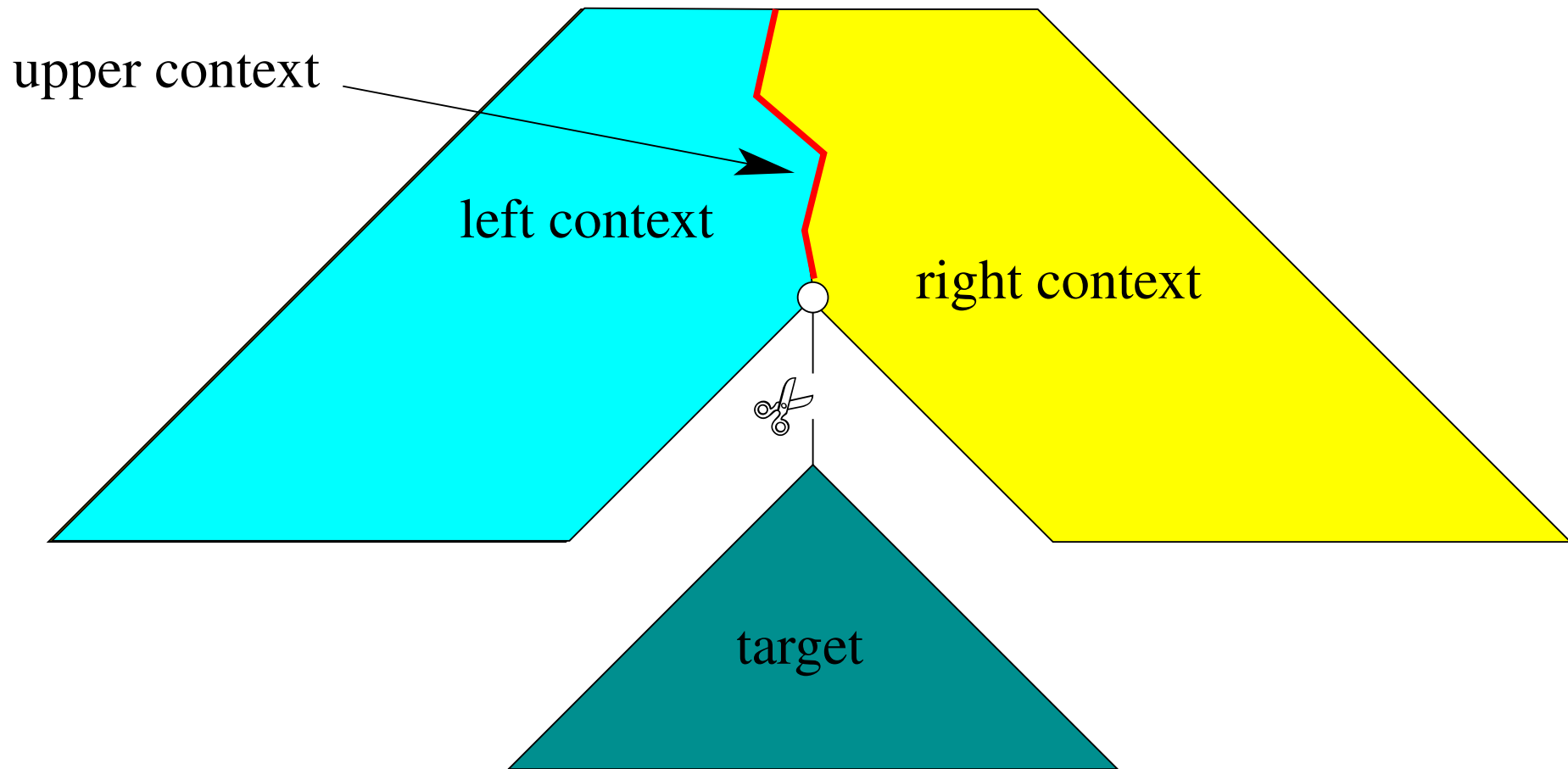
- ◇ The first runs **right-to-left**
⇒ checks right upper context and structure

- ◇ The second runs **left-to-right**
⇒ checks left context and signals match.

... **in particular:**

One pass suffices if only **left upper context** matters.





Goals:

- ◇ Allow wild cards,
- ◇ Allow external predicates (at least on tags / text),
- ◇ Avoid state explosion,
- ◇ Be reasonably efficient :-)

Goals:

- ◇ Allow wild cards,
- ◇ Allow external predicates (at least on tags / text),
- ◇ Avoid state explosion,
- ◇ Be reasonably efficient :-)

Idea: Construct automata **by need**:

- ◇ No possibly useless pre-computation.
- ◇ States/transitions constructed at first occurrence, then cached in a hash table.

- ⇒ ... very few states are constructed!
- ⇒ ... **querying** as fast as **parsing**.

Link:

[http://www.informatik.uni-trier.de/
~aberlea](http://www.informatik.uni-trier.de/~aberlea)