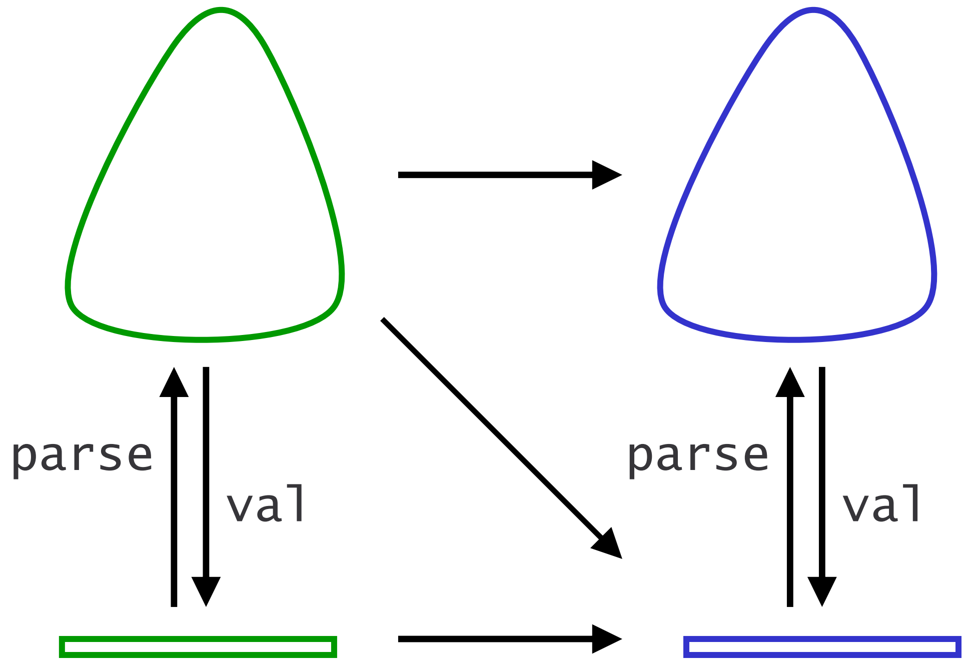
Tutorial on

Tree Transducers

Hendrik Jan Hoogeboom
LIACS Leiden

CSL/GAMES
Lausanne, sept 07

from tree to tree



symbolic /
syntax-directed translation

- compiler theory
- natural language
- document transformation

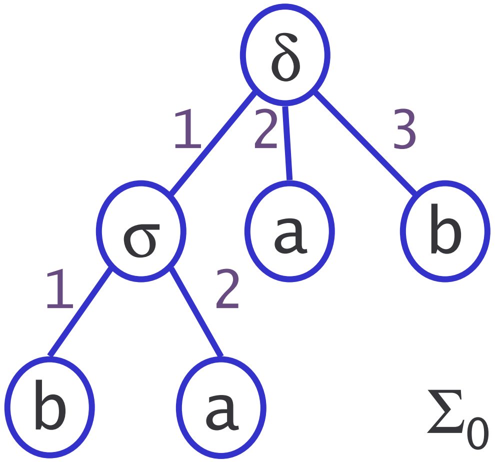
history

1960	Irons	syntax-directed translation
1968	Knuth	attribute grammar
1968	Thatcher& Rounds	top-down, bottom-up
1980	Aho& Ullman	tree-walking tr.
1985	Engelfriet& Vogler	macro tree tr.
2000	Milo& Suciu& Vianu	pebble tree tr. XML
	Fülöp& Vogler	book tree transducers
	Maneth	Tarragona lectures

contents

ranked trees ~ terms

[nested strings]



δ(σ(ab)ba)

ranked alphabet

(Σ,rank)

rank : $\Sigma \to \mathbb{N}$

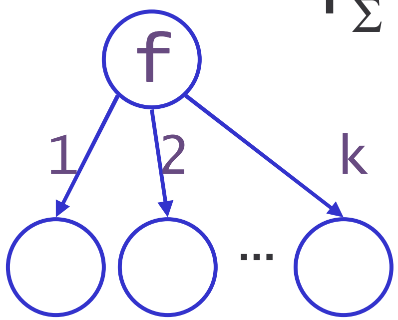
Σ_k rank k

Σ_0 = {a,b}

Σ_2 = {δ}

Σ_3 = {σ}

T_Σ trees over Σ

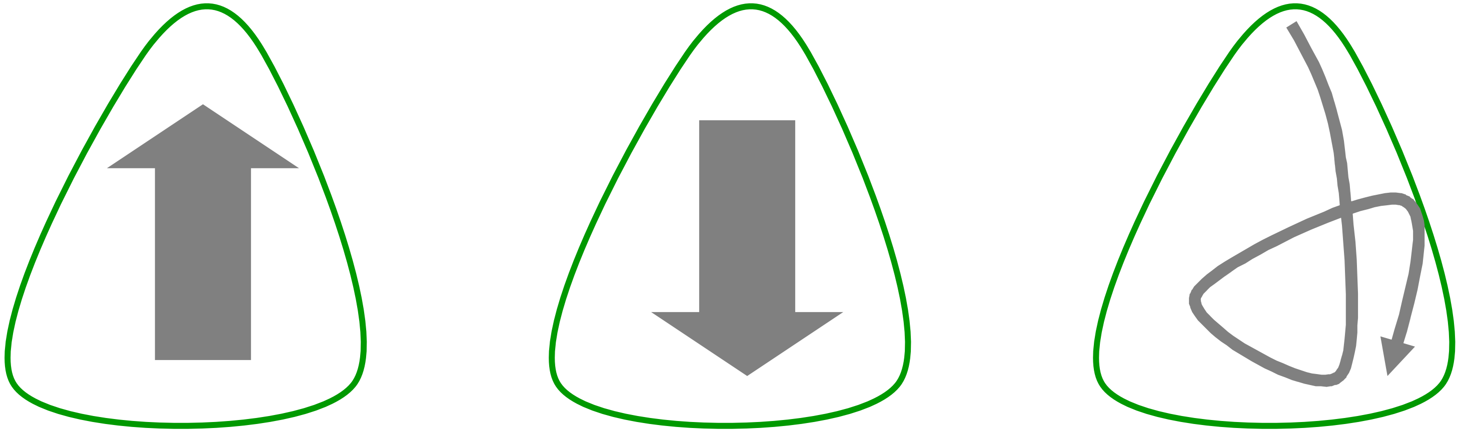


$f \in \Sigma_k$

$f(x_1 x_2 \ldots x_k)$

$f x_1 x_2 \ldots x_k$

tree automata



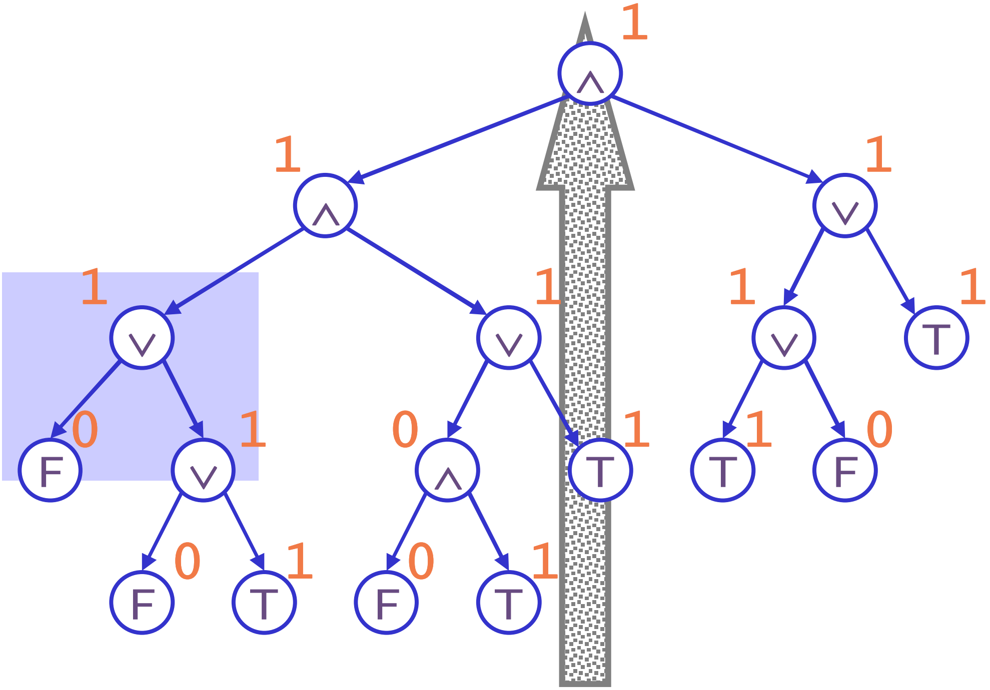
bottom-up *evaluation*

top-down *grammatical*

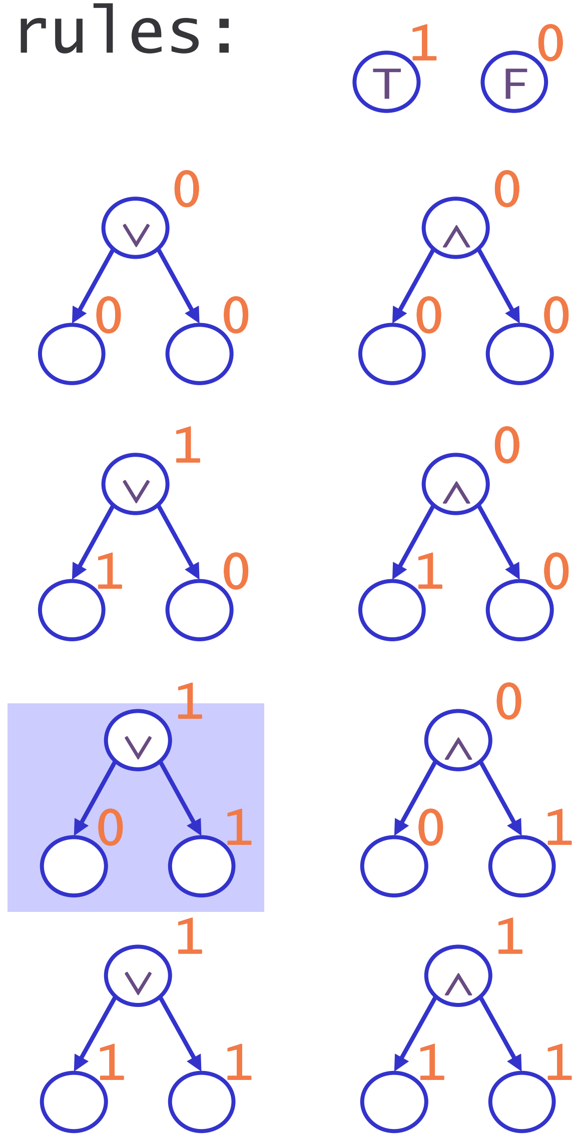
tree-walking *navigation*

‘parallel’

bottom-up

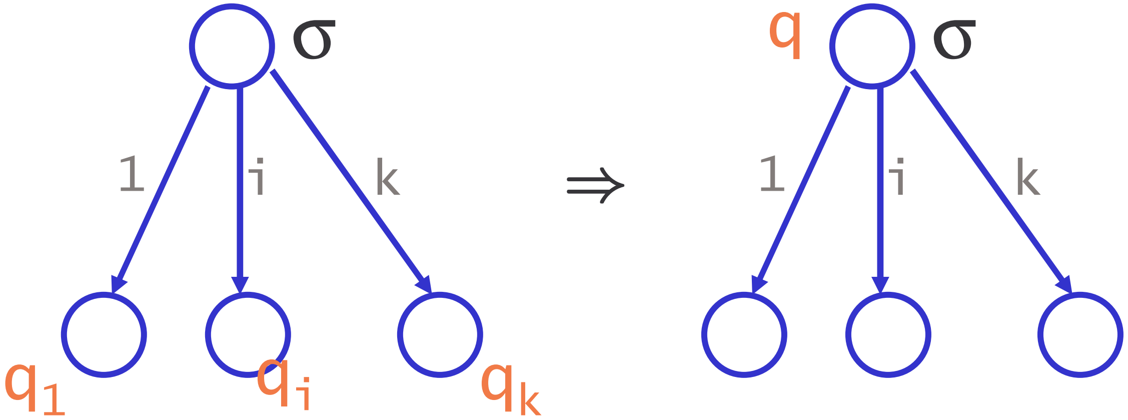


rules:



evaluation

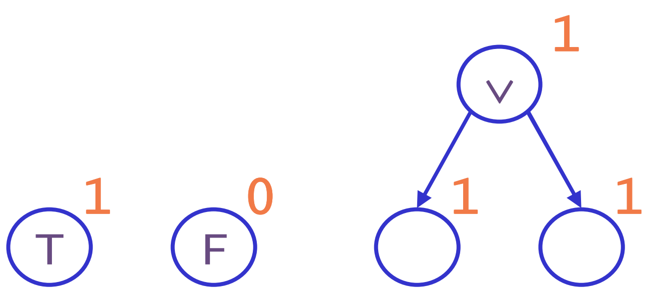
formalization



$$\sigma(q_1 \ldots q_k) \rightarrow q \qquad \text{rank}(\sigma)=k$$

$$\sigma \rightarrow q \qquad \text{rank}(\sigma)=0$$

acceptance by final state (at root)



$$F \rightarrow 0$$

$$T \rightarrow 1$$

$$\vee 11 \rightarrow 1$$

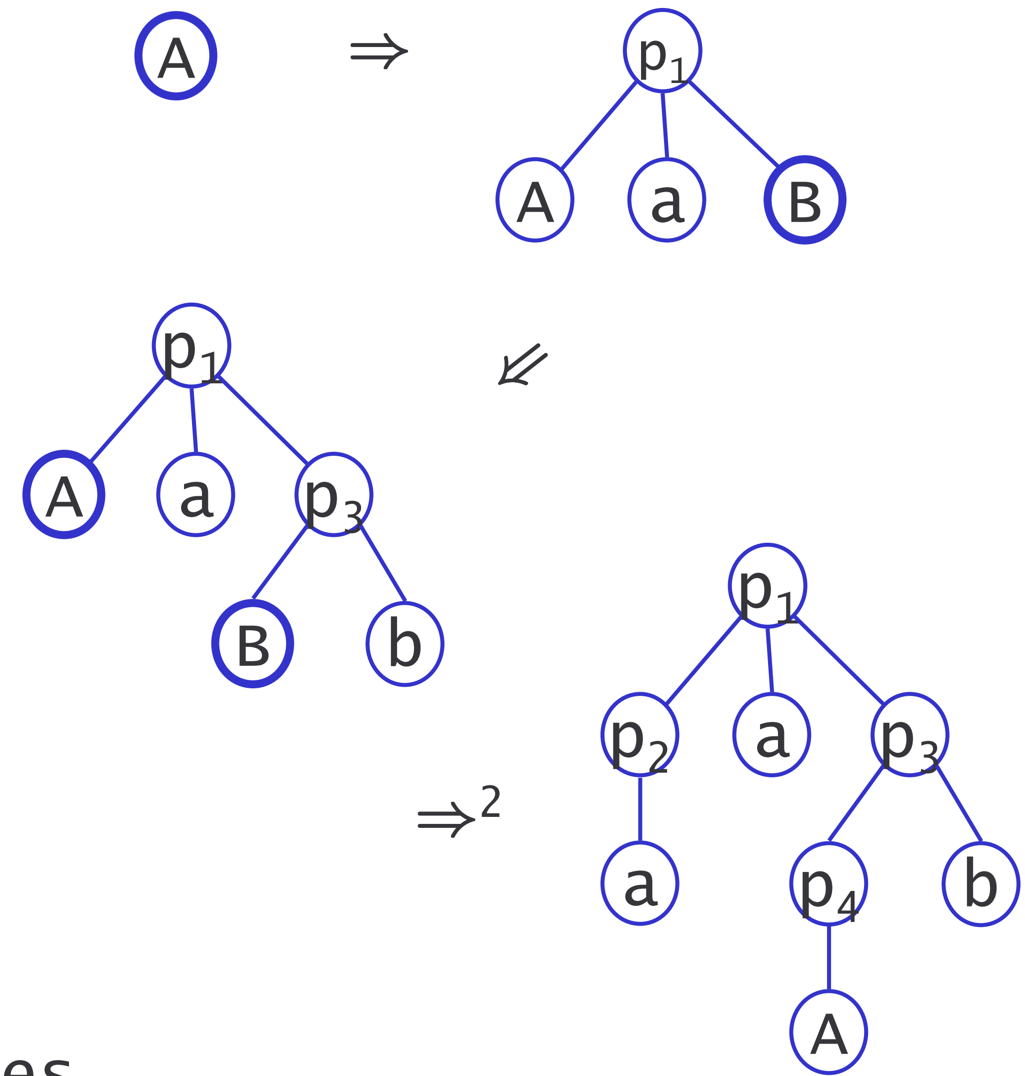
$$F, T \in \Sigma_0$$

$$\vee, \wedge \in \Sigma_2$$

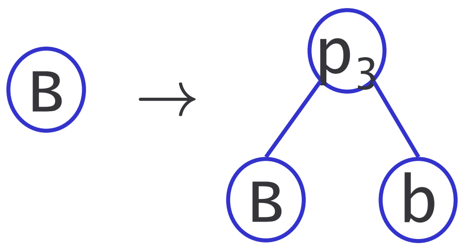
top-down

derivation tree

$p_1 : A \to AaB$
$p_2 : A \to a$
$p_3 : B \to Bb$
$p_4 : B \to A$



regular tree grammar



rewriting at leaves

regular tree languages

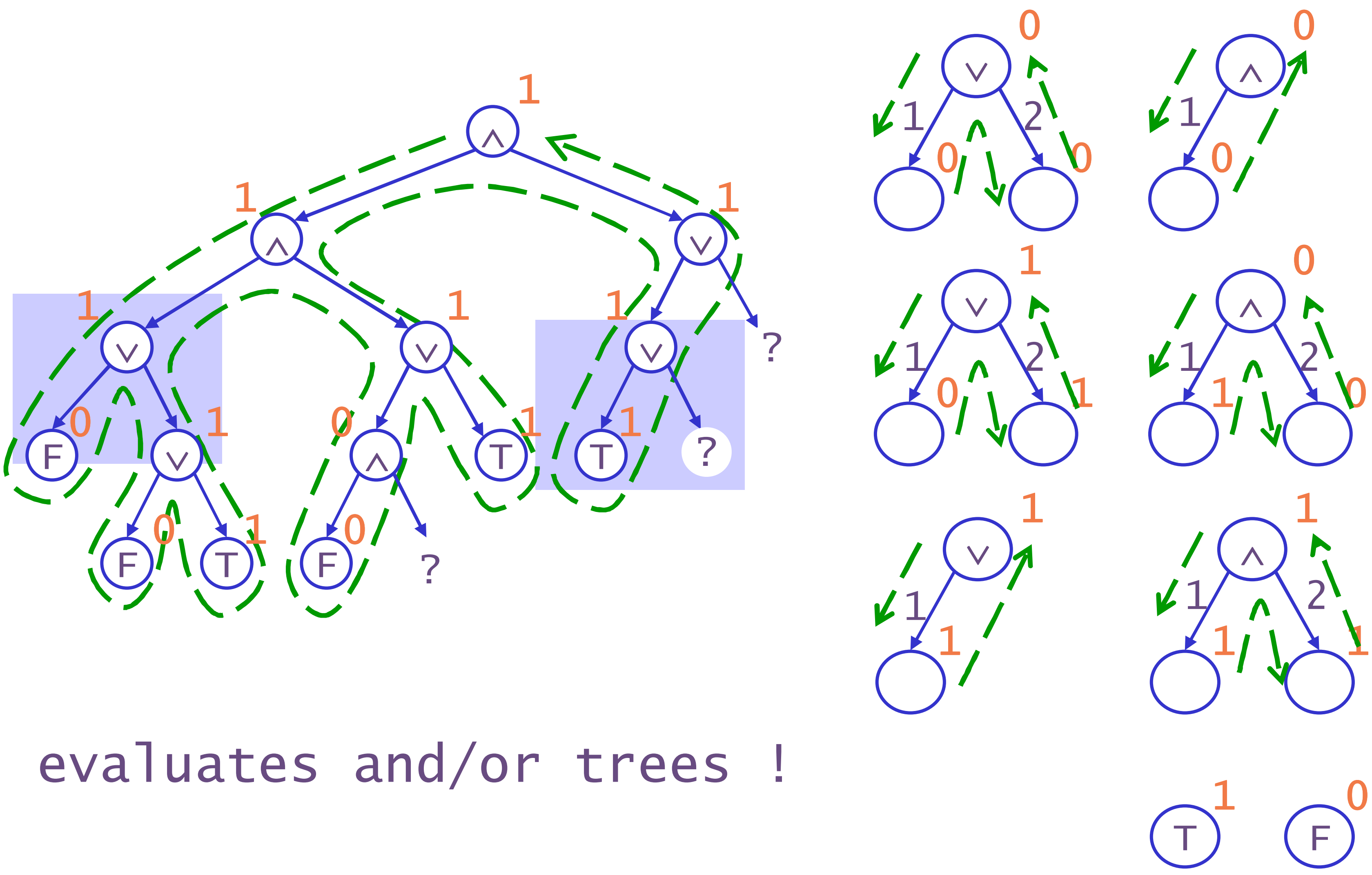
REG natural!

- bottom-up (det/nondet)
  - top-down (nondet)
  - MSO logic
  - regular tree grammars

- closed under intersection, complementation

- decidable emptiness (equivalence)

walking along the tree



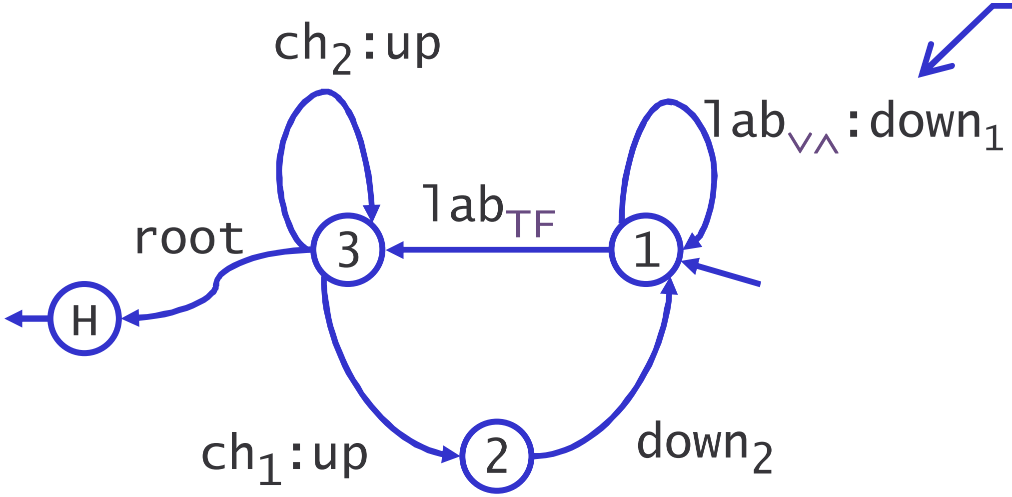
evaluates and/or trees !

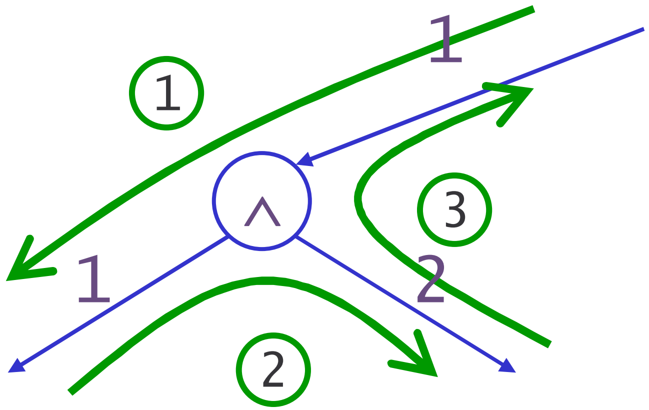
cf. two-way finite state automaton

example: tree traversal

TWA

when label is ∨ or ∧
move to first child

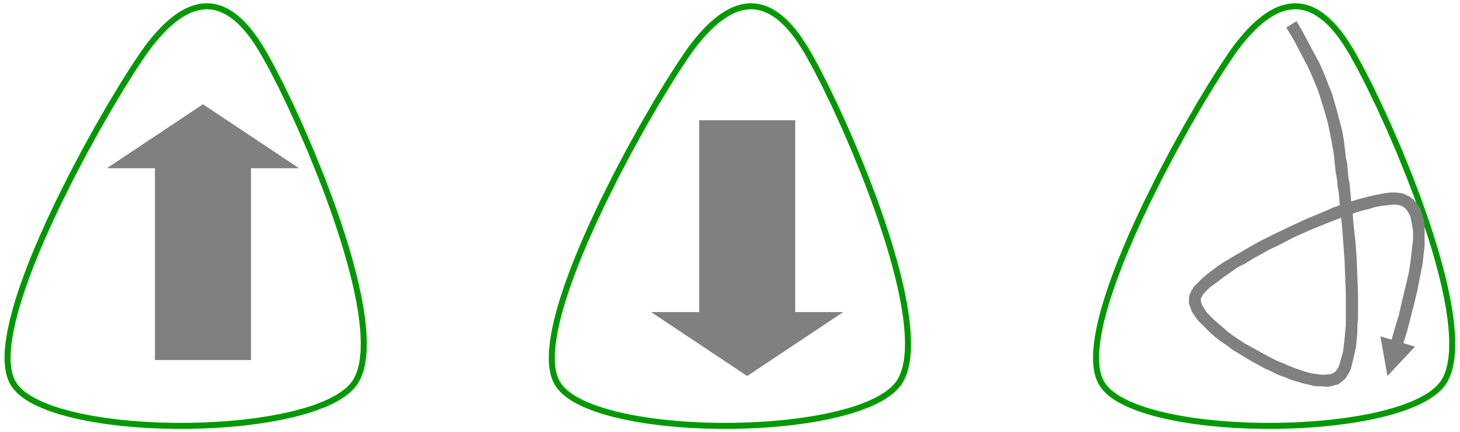




walk along edges, moves based on

- state (2)
- node label lab
- child number ch
  (= incoming edge)

tree automata



bottom-up *evaluation*

top-down *grammatical*

tree-walking *navigation*

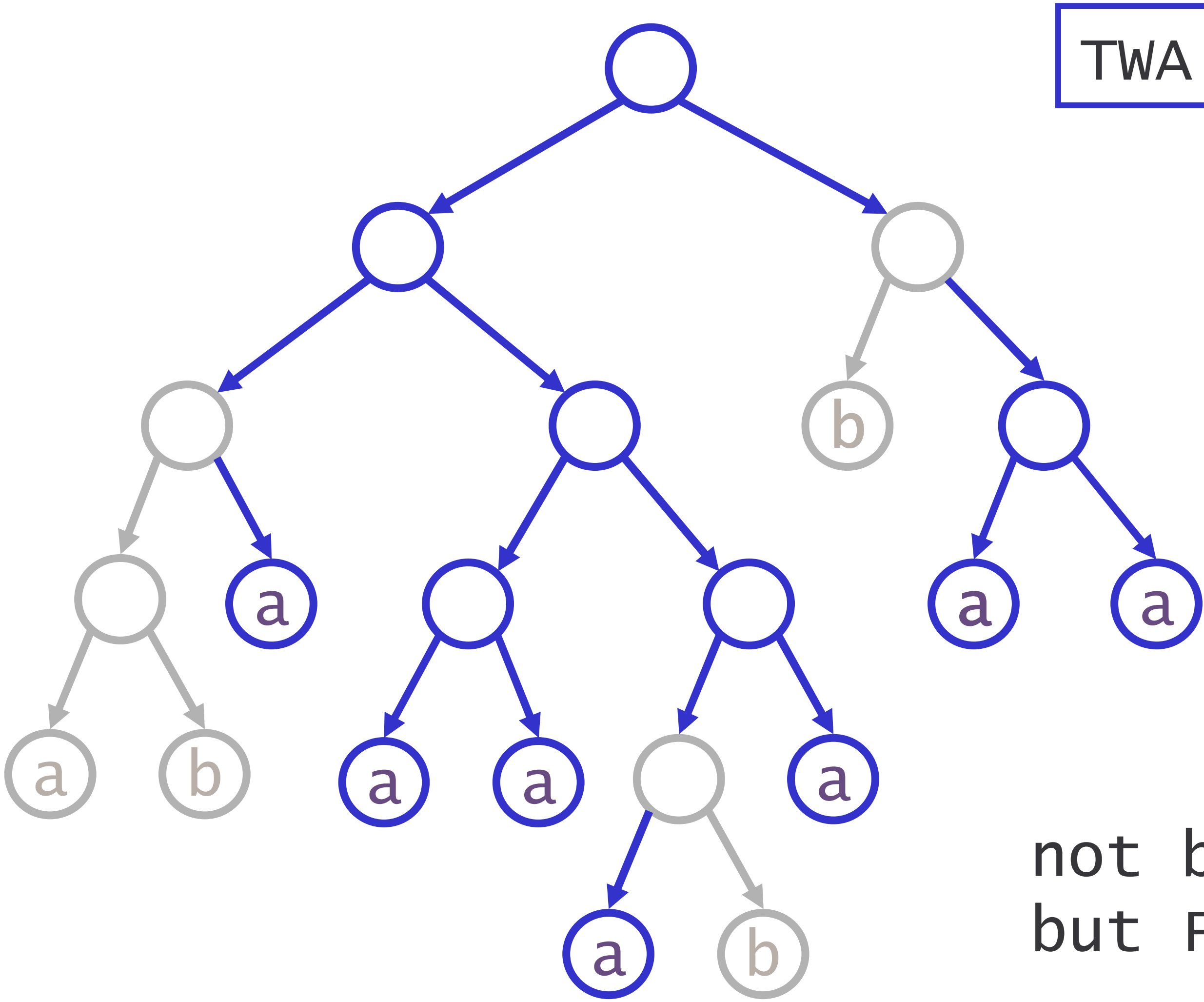
REG $\supseteq$ TWA

“twa easily loose their way”

‘branching structure’ of even length

Bojańczyk & Colcombet

TWA ⊂ REG



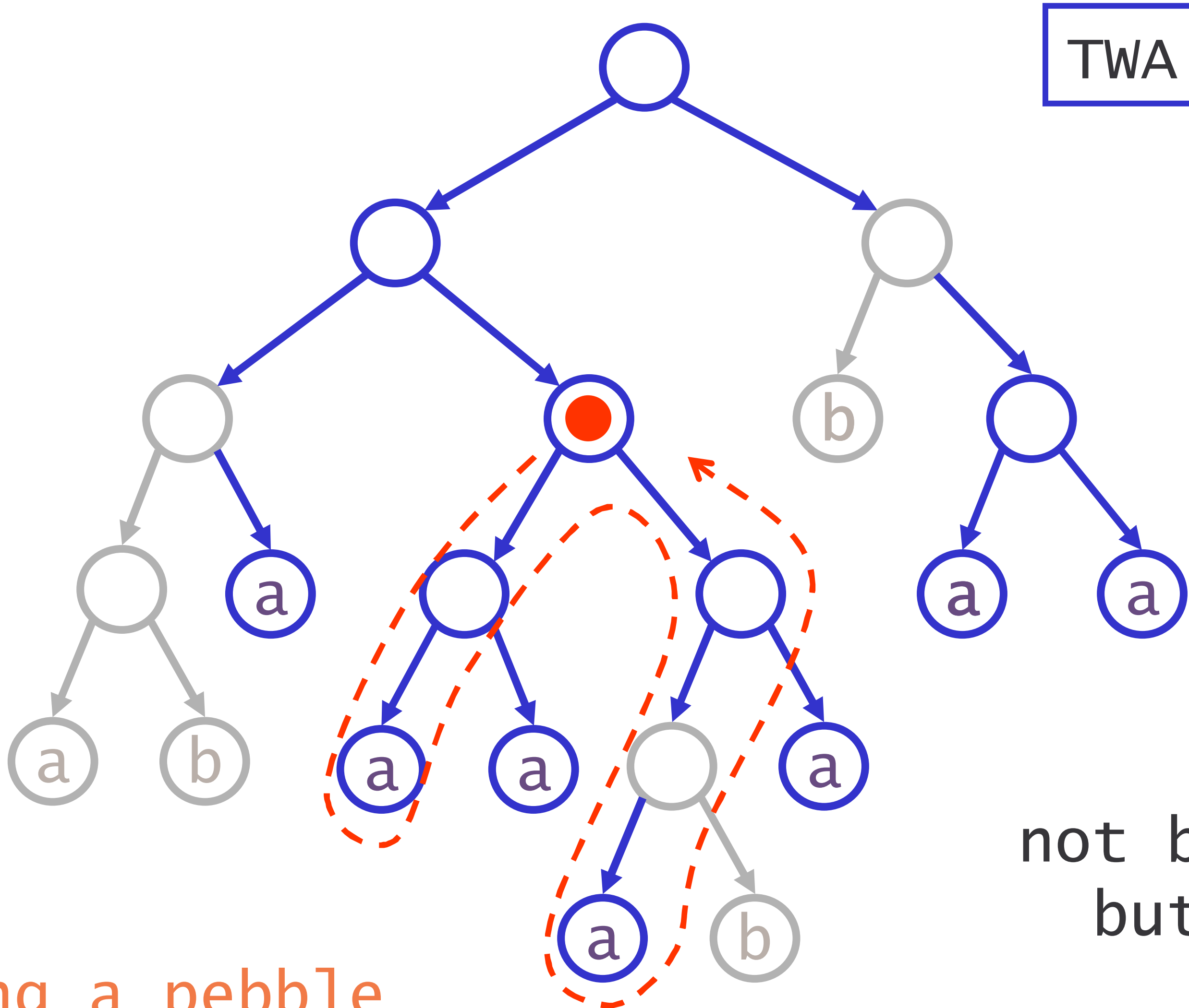
not by TWA
but FO (!)

(aa)*

‘branching structure’ of even length

Bojańczyk & Colcombet

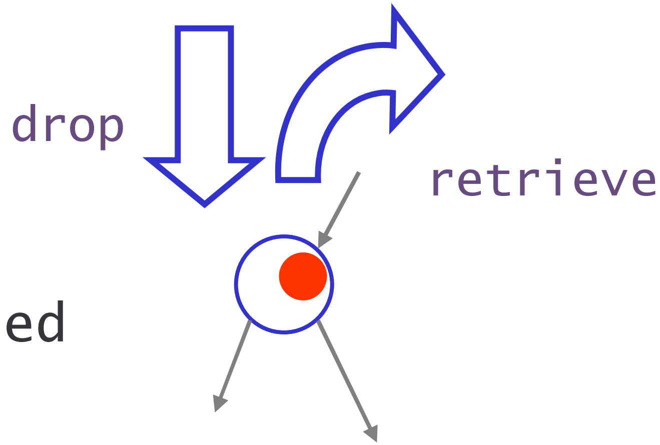
TWA ⊂ REG



not by TWA
but PTWA

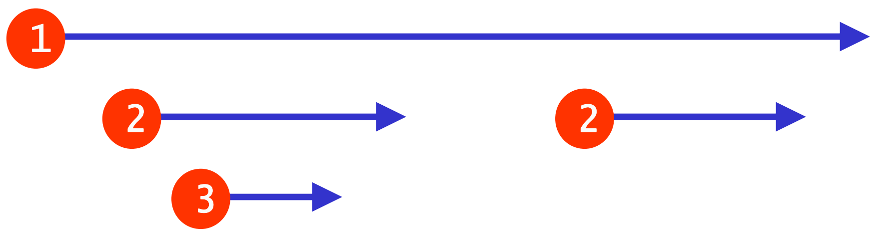
using a pebble

pebble: marks a node



- fixed number for automaton
- can be distinguished & reused

- *nested lifetimes* LIFO



'regular' extension

selected papers on pebble automata

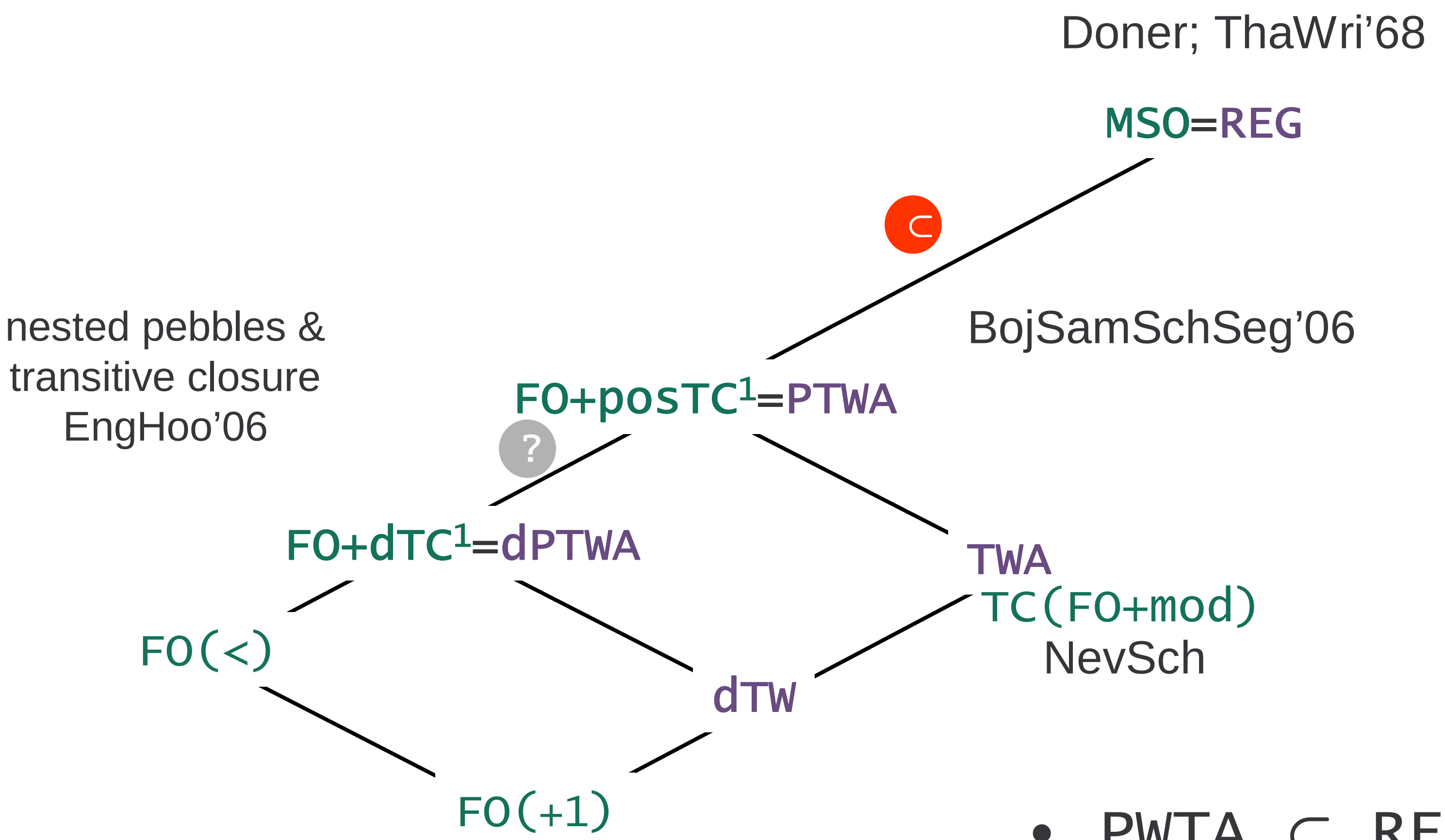
J.Engelfriet, H.J.Hoogeboom. Tree-walking pebble automata, *Jewels are forever*, 1999.

M.Bojańczyk, T.Colcombet. Tree-walking automata do not recognize all regular languages, STOC'05.

J.Engelfriet, H.J.Hoogeboom. Nested pebbles and transitive closure, LMCS, 2007.

M.Bojańczyk, M.Samuelides, T.Schwentick, L.Segoufin. On the expressive power of pebble automata, ICALP'06.

tree automata & logic



- PWTA ⊂ REG strict
- pebble hierarchy
- type of pebbles:
  physical vs. abstract

\section

macro tree

automata on trees

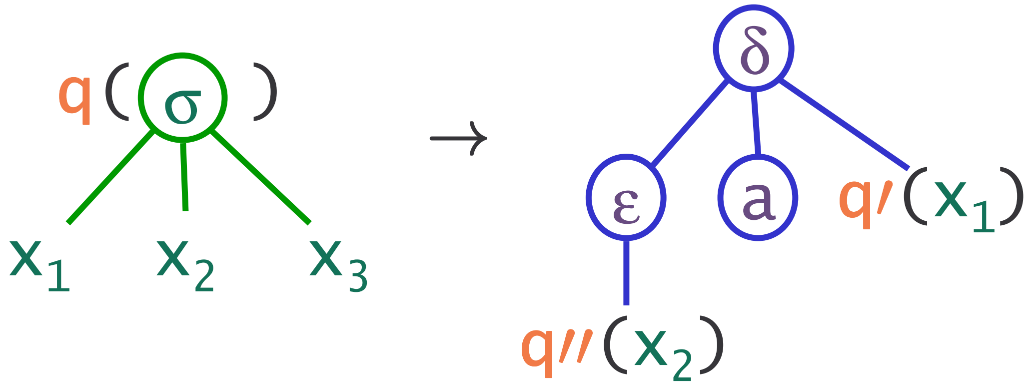
regular models

con pebble tree ree

transducers

grammars

top-down tree transducer



state + subtree ≡ node (input)

$T_\Sigma \to T_\Delta$

$\mathcal{A} = (\Sigma, \Delta, Q, Q_d, R)$

$q(\sigma(x_1 \ldots x_k)) \to t \in T_\Delta[Q(X_k)]$ rank(σ)=k

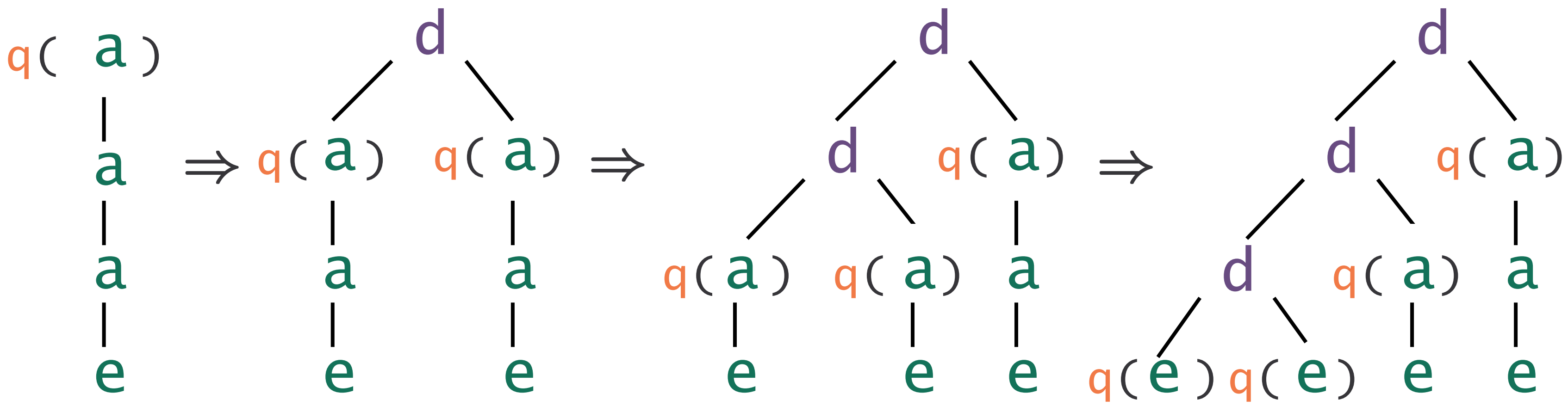
$q(\sigma) \to t \in T_\Delta$ rank(σ)=0

$\{ (t,s) \in T_\Sigma \times T_\Delta \mid q(t) \Rightarrow^* s,\ q \in Q_d \}$

example top-down

$\Sigma_0 = \{e\},\ \Sigma_1 = \{a\}$
$\Delta_0 = \{e\},\ \Delta_2 = \{d\}$

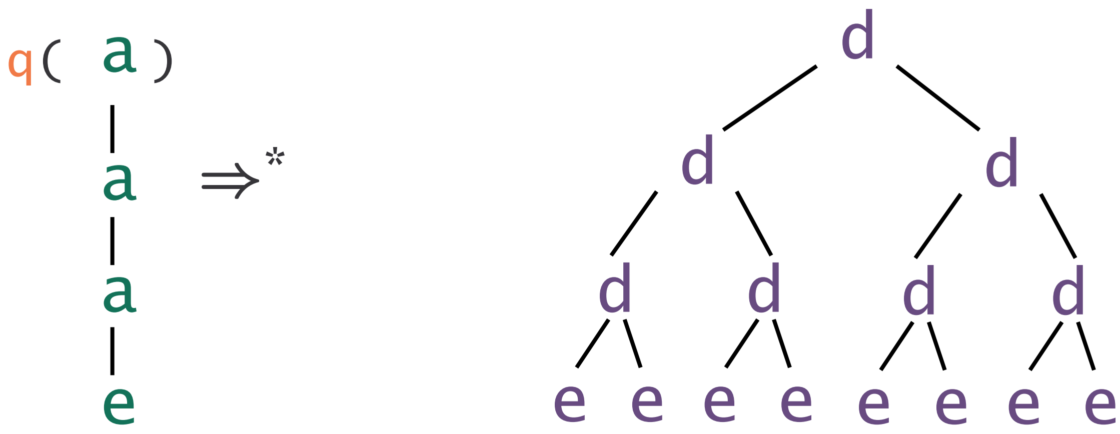
$$q(a(x)) \rightarrow d(q(x), q(x))$$
$$q(e) \rightarrow e$$



example top-down

Σ_0 = {e}, Σ_1 = {a}
Δ_0 = {e}, Δ_2 = {d}

q(a(x)) $\rightarrow$ d(q(x),q(x))
q(e) $\rightarrow$ e



exponential
size increase

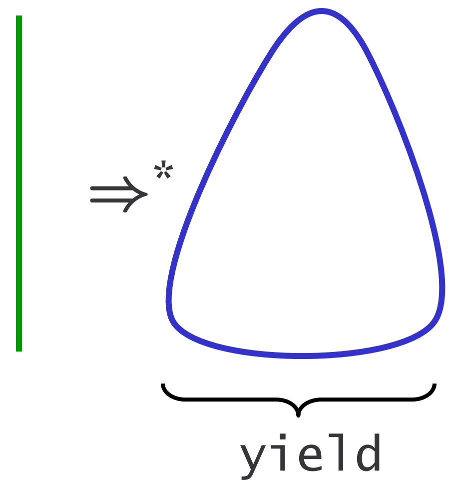
top-down

rules are confluent

$$q(a(x_1)) \rightarrow d(q(x_1)q(x_1)) \quad \textit{copy}$$
$$q(c(x_1x_2x_3)) \rightarrow d(q(x_1)q(x_2)) \quad \textit{delete}$$

linear height increase
exponential size increase

yield linear input $\rightarrow$ ET0L
more Lindenmayer connections



top-down characteristic

‘T’ copying input, processing copies differently

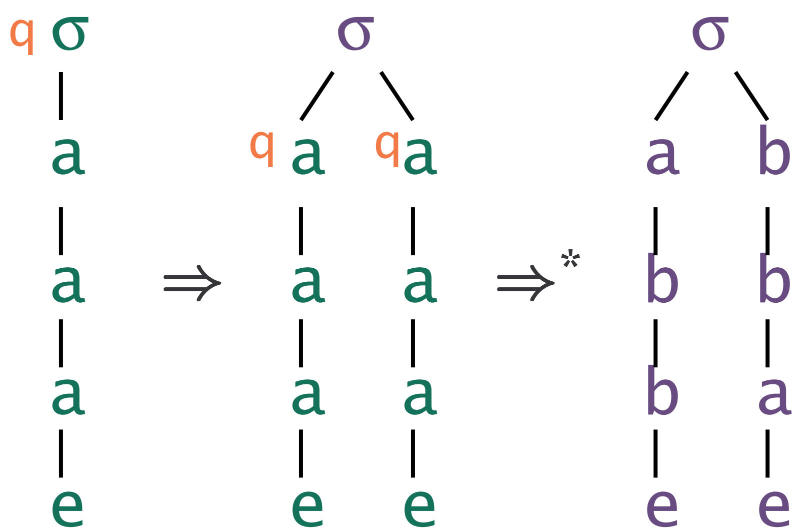
Σ_0 = {e}, Σ_1 = {a,σ}

Δ_0 = {e}, Δ_1 = {a,b}, Δ_2={σ}

q(σ(x)) → σ(q(x),q(x))

q(a(x)) → a(q(x)) | b(q(x))

q(e) → e

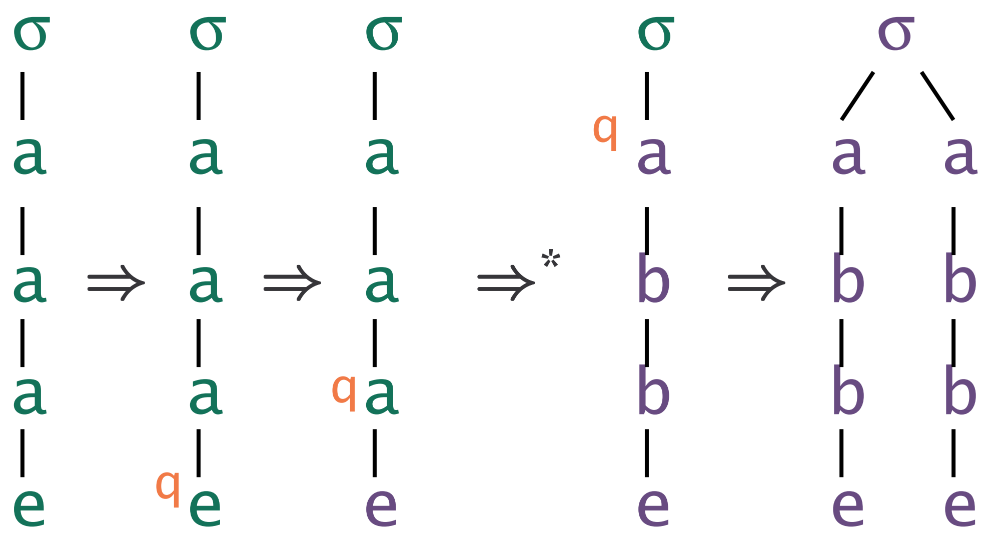


bottom-up characteristic

'B1' copying output after nondet processing input

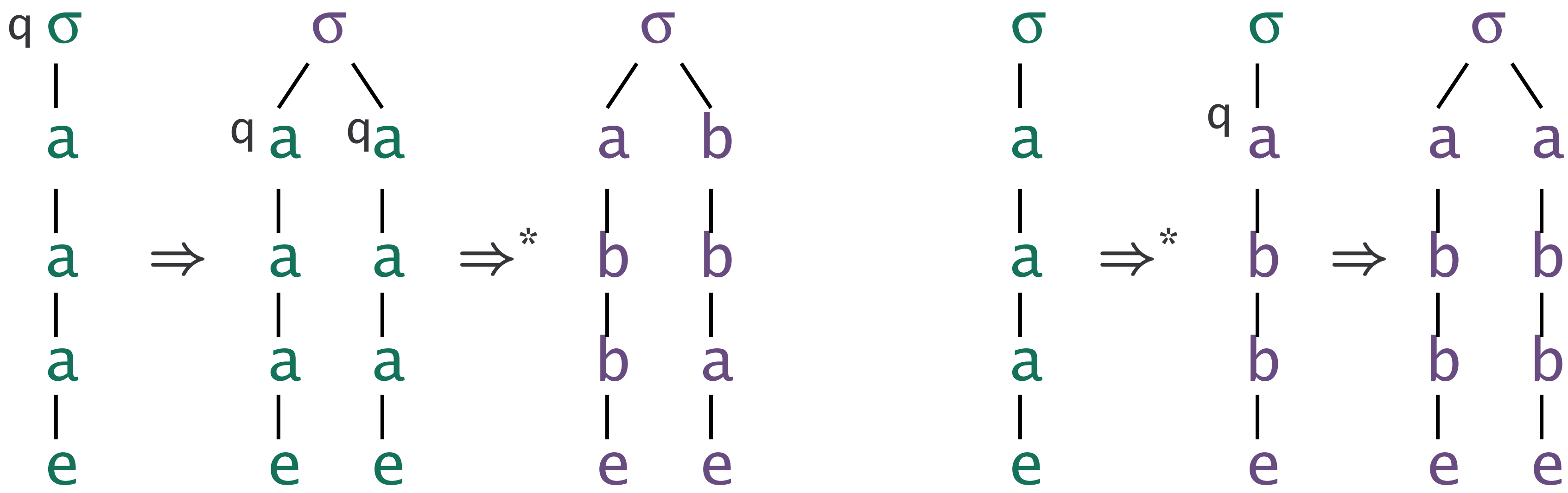
$\Sigma_0 = \{e\},\ \Sigma_1 = \{a,\sigma\}$
$\Delta_0 = \{e\},\ \Delta_1 = \{a,b\},\ \Delta_2 = \{\sigma\}$

$e \rightarrow q(e)$
$a(q(x)) \rightarrow q(a(x)) \mid q(b(x))$
$\sigma(q(x)) \rightarrow q(\sigma(xx))$



top-down vs. bottom-up

top-down   bottom-up



copy   relabel   relabel   copy

bottom-up^2 top-down^2

‘T’ copying input, processing copies differently

‘B1’ copying output after nondet processing input

some properties

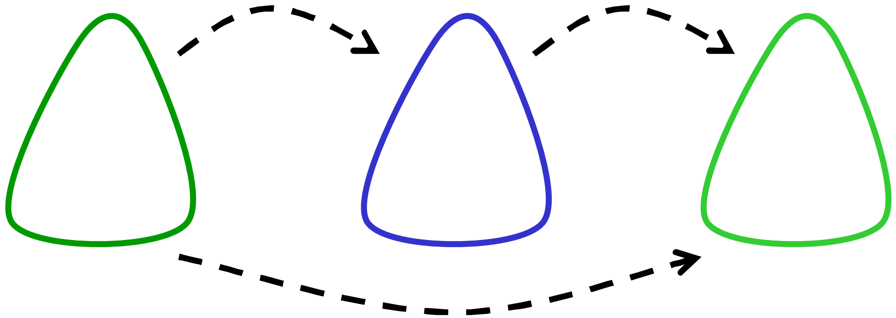
TDT and BUT

… have REG domains

… REG closed under inverse

… are incomparable

… are not closed under composition



linear transducers

linear – no *copy* $q(a(x_1)) \to d(q(x_1)q(x_1))$

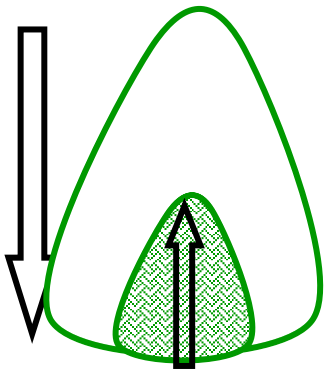
lin-BUT

= lin-TDT*

= lin-TD + *regular look-ahead*

= (FTA ∪ RELAB ∪ LHOM)*

intersection REG — single state, no copy



… is closed under composition

… REG closed under lin-BUT

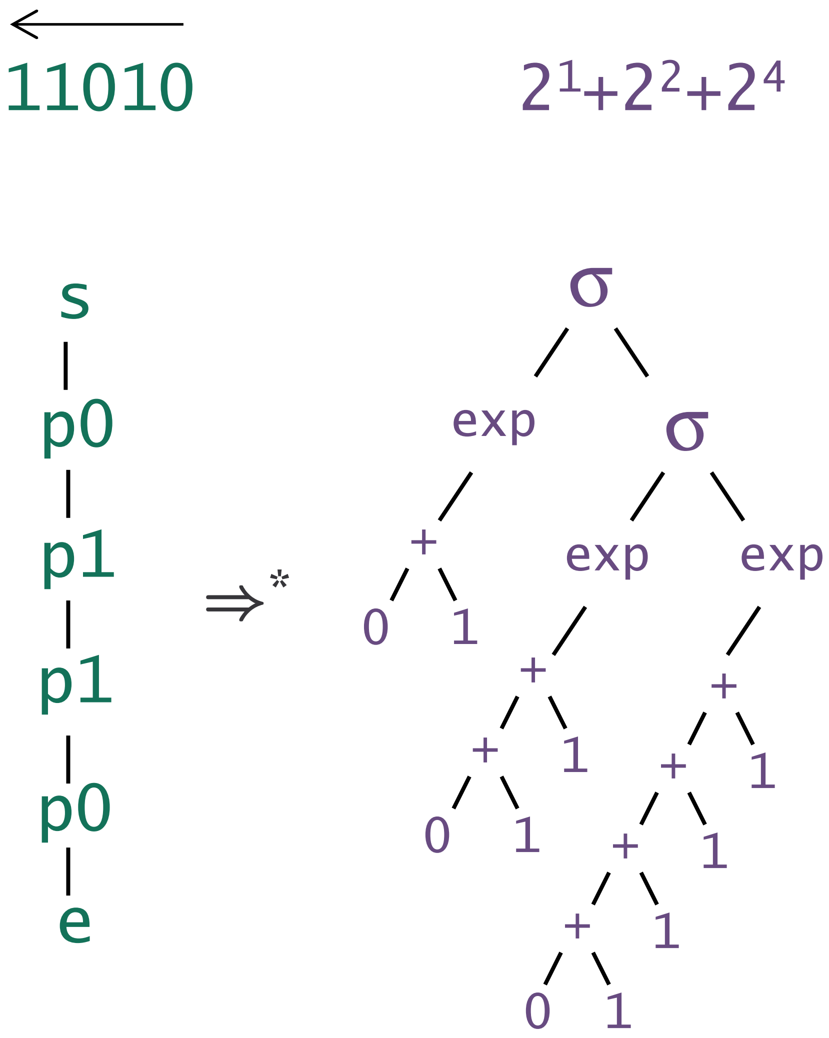
macro tree

automata on trees

regular models

context-free tree grammars

```
 S: S → N
p0: N → N0
p1: N → N1
 e: N → 1
```



handle context!

regular vs. context-free

STRINGS

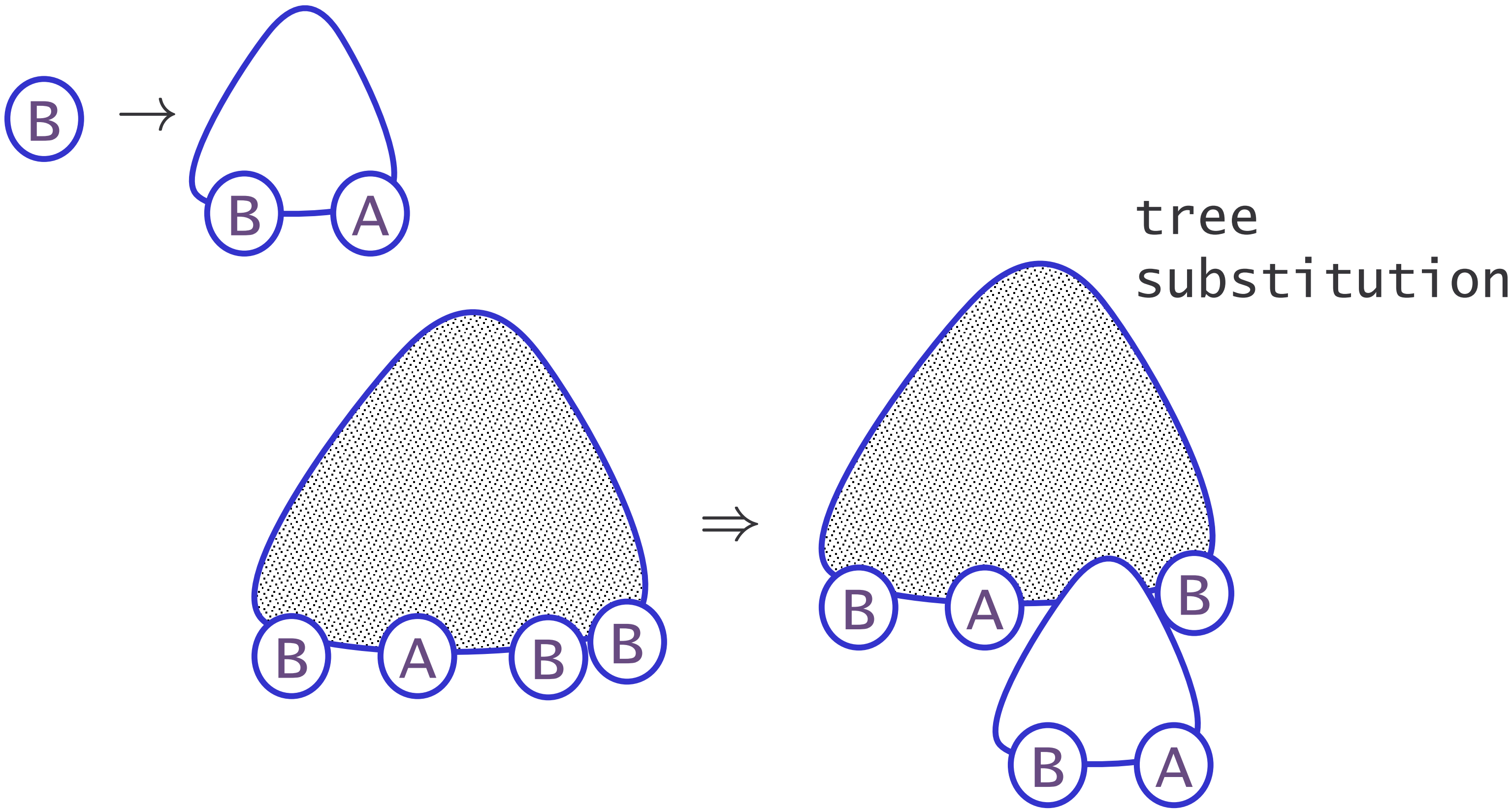
B → bA

aabaB ⇒ aababA

B → BbA

aaBbaA ⇒ aaBbAbaA

TREES



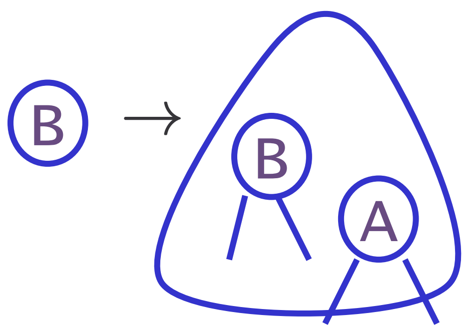
regular vs. context-free

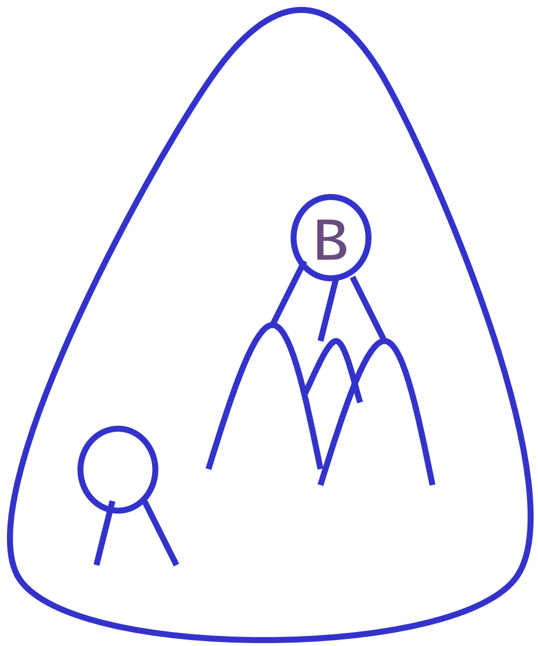
STRINGS

$$B \rightarrow bA \qquad\qquad B \rightarrow BbA$$

$$aaba\underline{B} \Rightarrow aaba\underline{bA} \qquad\qquad aa\underline{B}baA \Rightarrow aa\underline{BbA}baA$$

TREES





how to handle subtrees?

context-free tree grammars

- *regular*

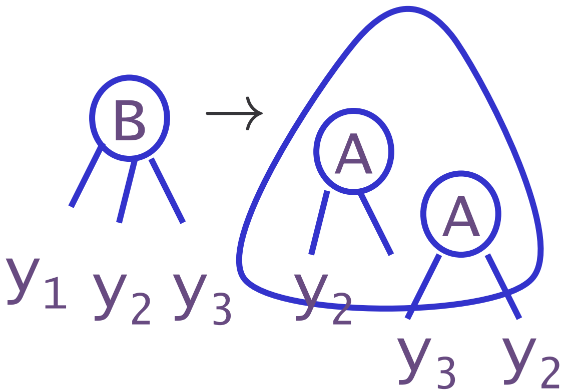
$B \to t \in T_\Sigma[N]$ N nonterminals $B \in N$

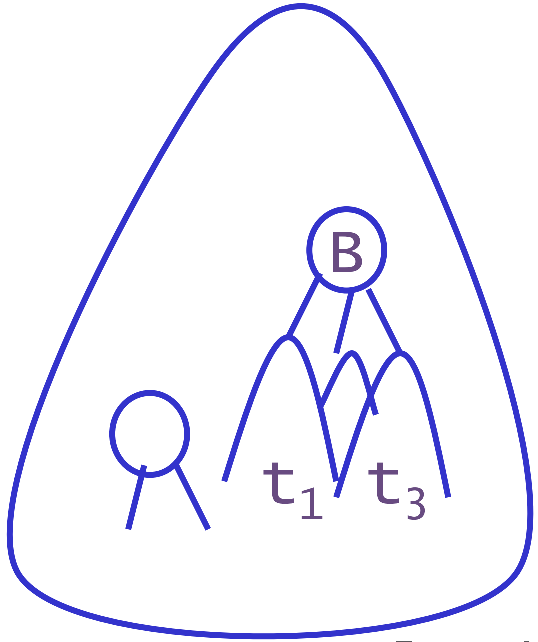
- *context-free*

$B(y_1,\ldots,y_m) \to t \in T_{\Sigma\cup N}[Y_m]$

N ranked nonterminals $B \in N_m$

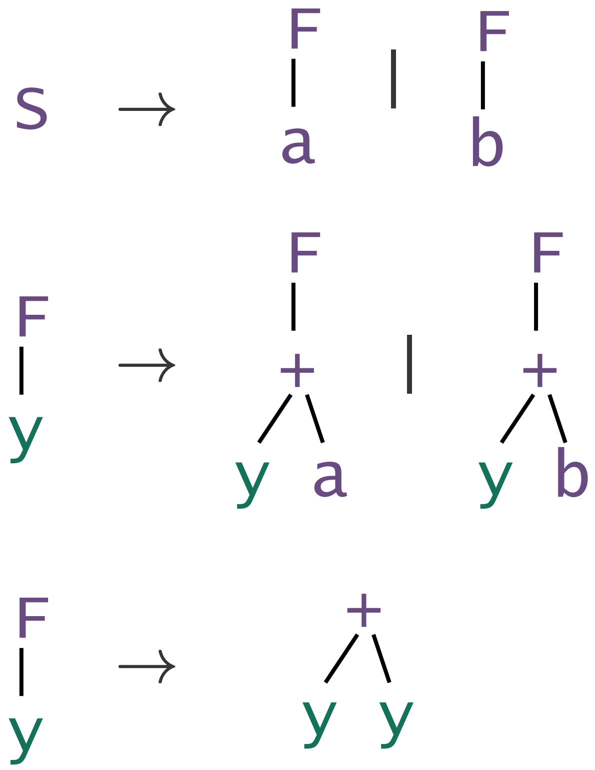
$Y_m = \{y_1,\ldots,y_m\}$ parameters

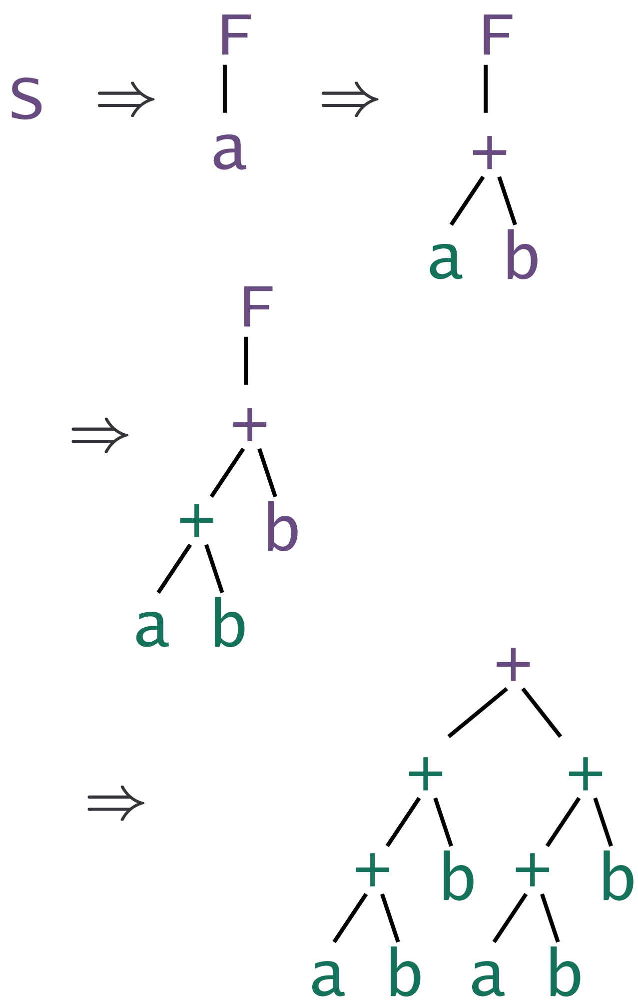




parameters←actual subtrees

cf tree grammar



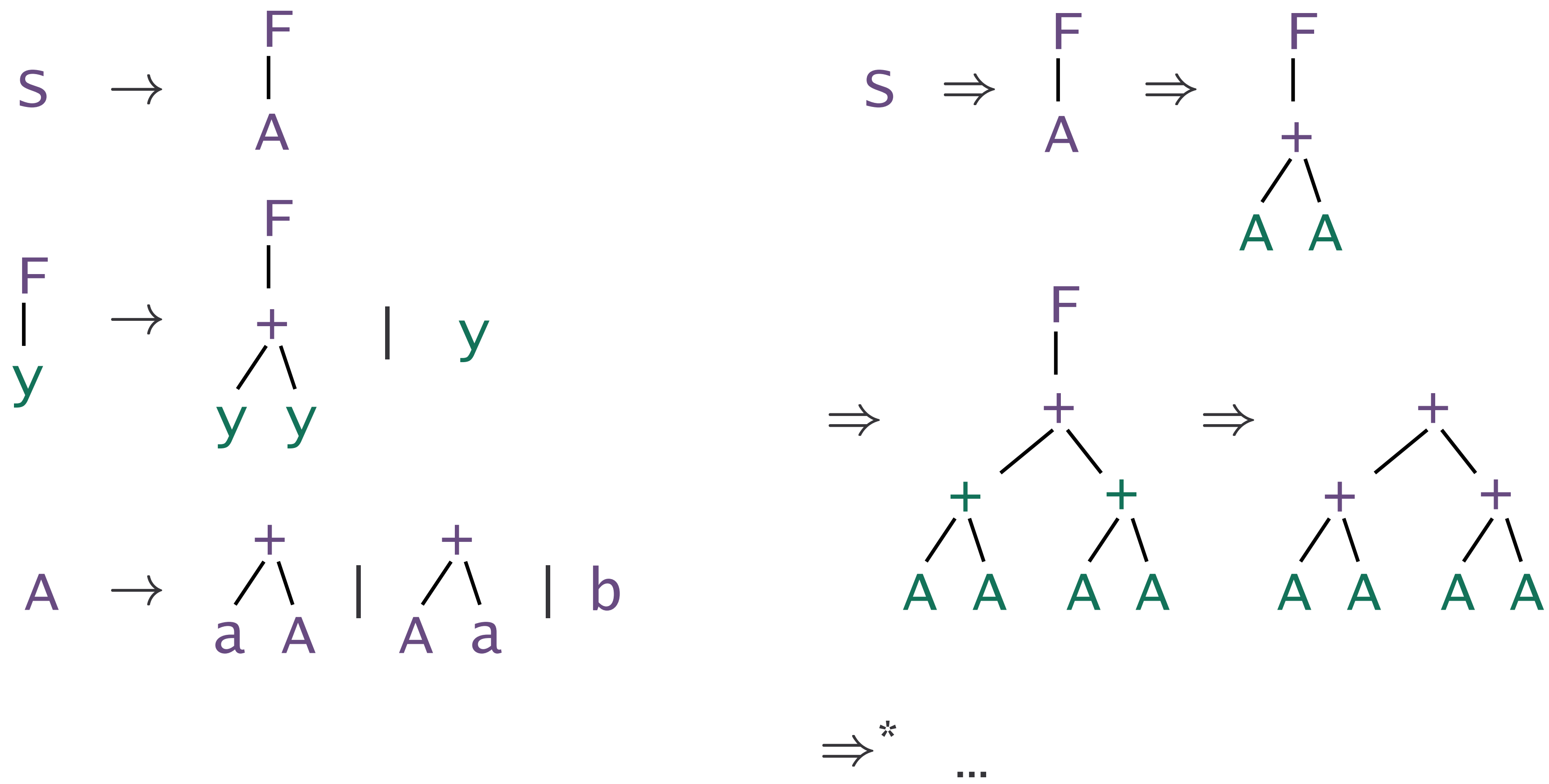


$S \in N_0 \quad F \in N_1$

$a,b \in \Sigma_0 \quad + \in \Sigma_2$

yield $\{ ww \mid w \in \{a,b\}^* \}$
not context-free

(life is simple ...)



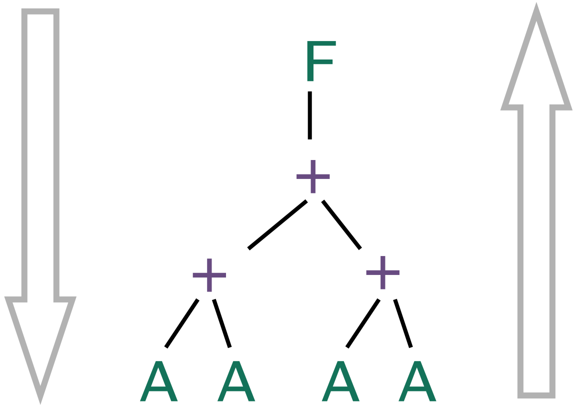
$S, A \in N_0 \quad F \in N_1$

$a, b \in \Sigma_0 \quad + \in \Sigma_2$

yield
$\{\ w \in \{a,b\}^* \mid w \text{ has } 2^n \text{ b's}\ \}$

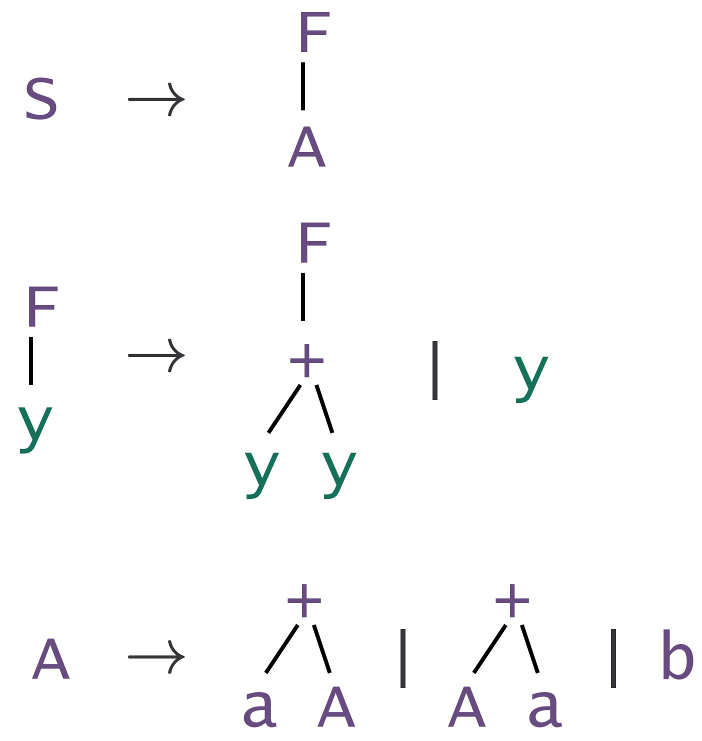
(… no it isn’t)

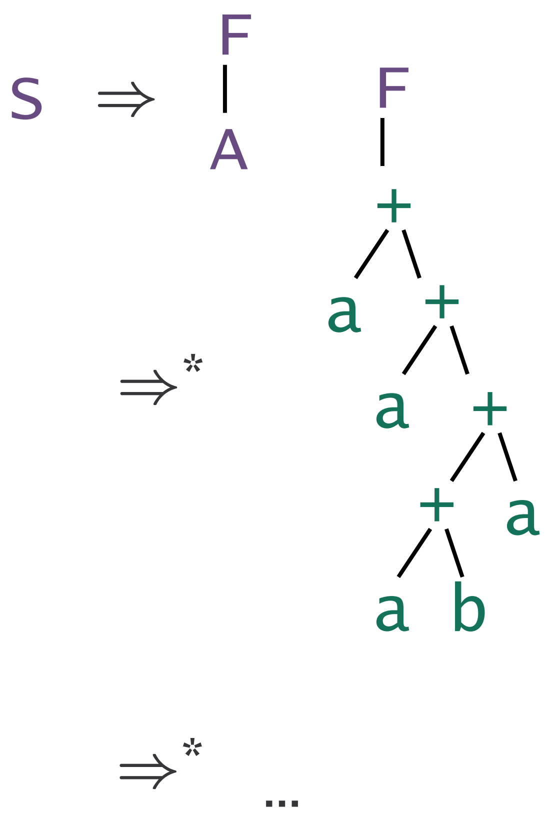
cfg: leftmost vs. unrestricted derivations



OI	outside-in	top-down	‘lazy’
		unrestriced	
IO	inside-out	bottom-up	‘eager’

IO mode





$S, A \in N_0 \quad F \in N_1$

$a, b \in \Sigma_0 \quad + \in \Sigma_2$

yield
{ w^2ⁿ | w ∈ a^*ba^* }

IO-CFT and OI-cft incomparable

IO generates more equal copies
OI is lazy: unsuccessful subtrees

OI ≡ unrestricted

postpone ‘inner’ steps
context-free property

yield REG = CFL
yield OI-CFT = Indexed

macro tree transducers

automa…

regular models

con… ree transducers

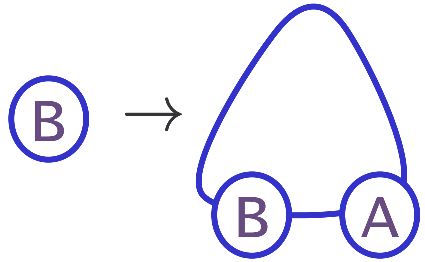
selected papers on macro tree transducers

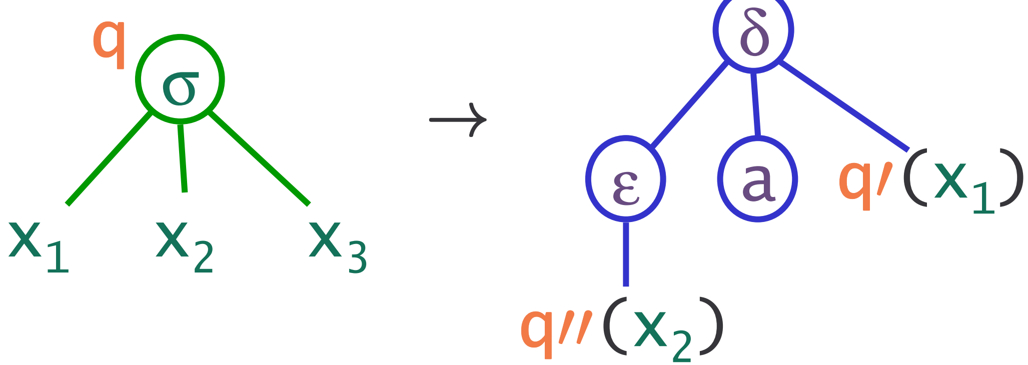
J.Engelfriet, H.Vogler. Macro tree transducers, JCSS, 1985.

macro tree transducers

top-down tree transducers (input) &
context-free tree grammars (output)

regular





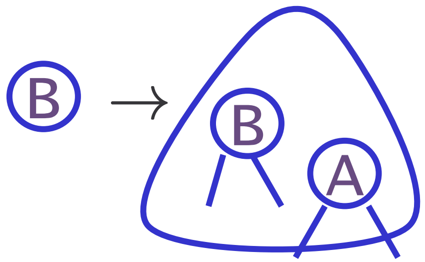
state +
subtree ≡ node (input)

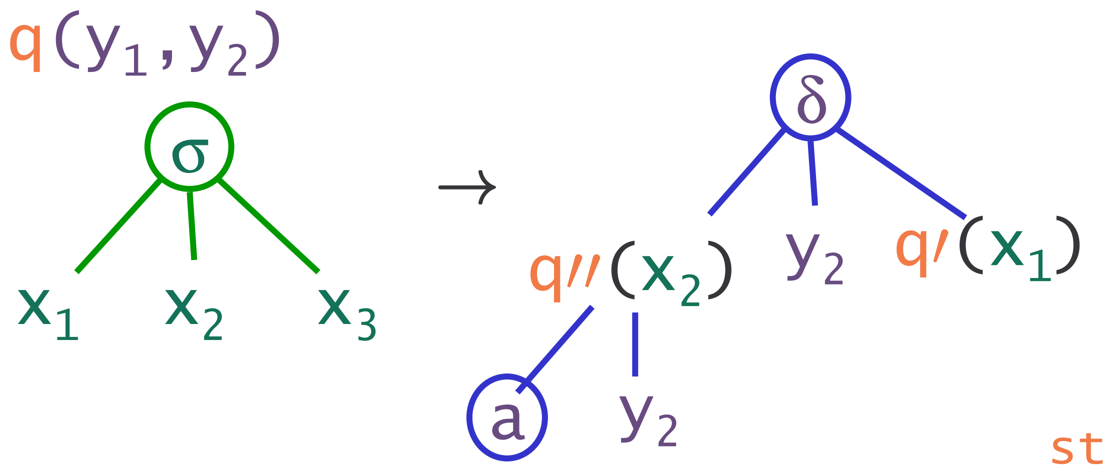
$q(\sigma(x_1 \ldots x_k)) \to t \in T_\Delta[Q(X_k)]$ rank(σ)=k

macro tree transducers

top-down tree transducers (input) &
context-free tree grammars (output)

context-free



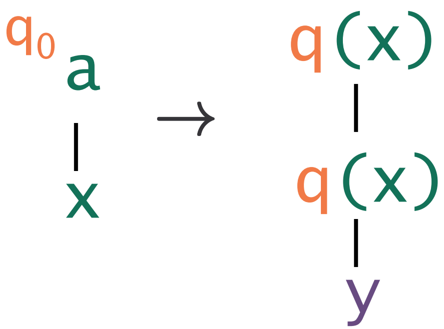


state +
subtree ≡ node (input) +
parameters (output)

$q(\sigma(x_1 \dots x_k), y_1 \dots y_m) \to$
$t \in T_{\Delta \cup Q(Xk)}[Y_m]$ rank(σ)=k, rank(q) =m

mtt for linear trees

$$q_0\langle a(x_1)\rangle \to q(x_1)(q(x_1)\ e)$$
$$q\langle a(x_1), y_1\rangle \to q(x_1)(q(x_1)\ (y_1))$$
$$q\langle e, y_1\rangle \to a(y_1)$$



$q_0(aae) \Rightarrow$

$q(ae)q(ae)e \Rightarrow$

$q(e)q(e)q(ae)e \Rightarrow$

$aq(e)q(ae)e \Rightarrow$

$aaq(ae)e \Rightarrow$

$aaq(e)q(e)e \Rightarrow$

$aaaq(e)e \Rightarrow$

$aaaae$

exponential size-to-height

double exponentional size-to-size

MTT properties

- unrestricted $\equiv$ OI
- OI-MTT and IO-MTT incomparable

- MTT has *regular look-ahead*

  bottom-up inspection

- REG closed under inverse MTT

  $T^{-1}(R) \in REG$

macro tree

automata on trees

els

con ree

pebble tree transducers

selected papers on pebble tree transducers

T.Milo, D.Suciu, V.Vianu. Typechecking for XML transformers, JCSS, 2003.

J.Engelfriet, S.Maneth. A comparison of pebble tree transducers with macro tree transducers, Acta Inf, 2003.

pebble tree transducers

tree-walking automata Aho& Ullman 71

Milo etal. 2000:
'all XML query languages can be modeled by k-pebble tree transducers'

great for navigation, but:
k-PTT cannot test all regular domains

comparison pebbles vs. macro:

- n-dPTT $\subseteq$ 0-dPTT^{n+1} $\subseteq$ dMTT^{n+1}
- dMTT $\subseteq$ 0-dPTT^{3}

$\Rightarrow$ same composition closure

(1) logic to nested pebbles

$\varphi \rightarrow \mathcal{A}$

$lab_a(x)$
$edg_i(x,y)$

$x \leq y$
$x = y$

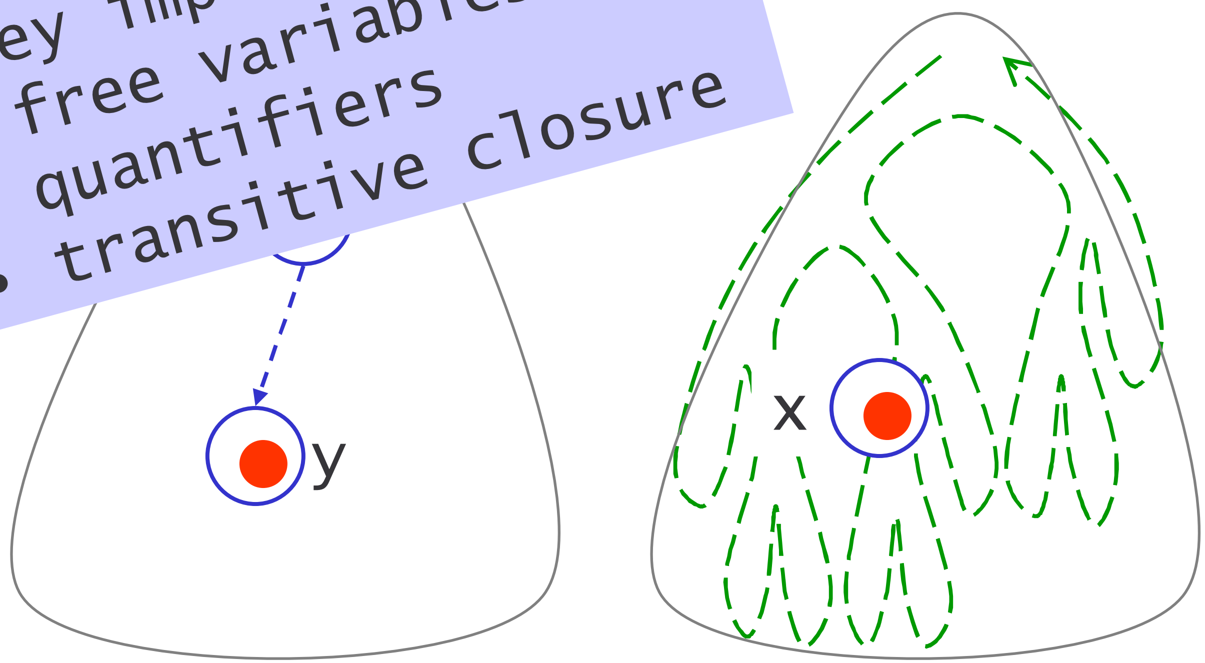
$\neg \; \wedge \; \vee$
$\forall x \; \exists x$

$\varphi^*(x,y)$

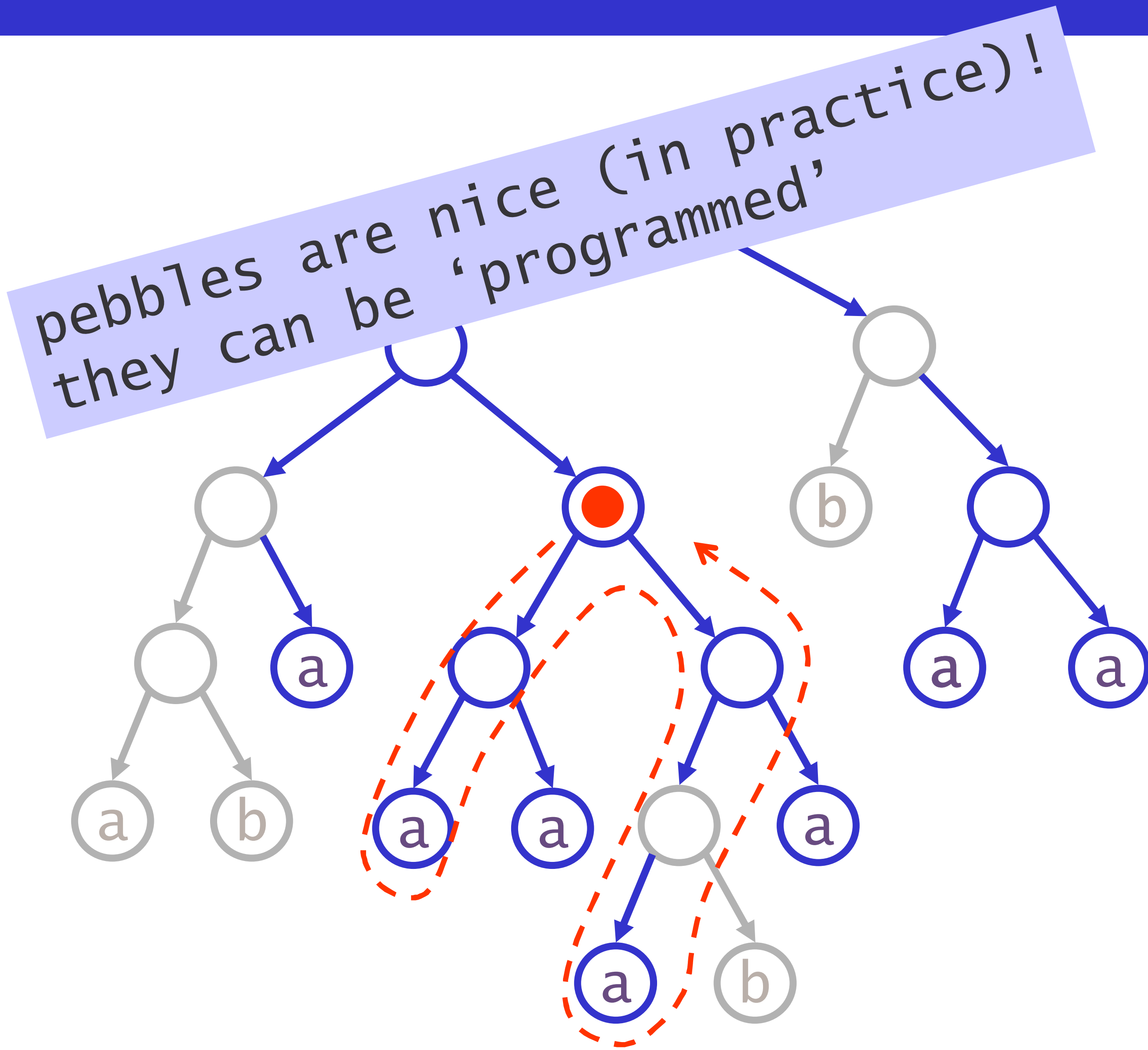
pebbles are nice
(in theory)!

they implement
• free variables
• quantifiers
• transitive closure

...ays halting
...e variables ~
fixed pebbles



$x \leq y$ $\quad\quad$ $\forall x \; \varphi(x) \quad \mathcal{A}_\varphi$


pebbles are nice (in practice)!
they can be 'programmed'
a
a
b
a
a
a
a
a
a
b
b
a
a

▶ 'classic' pebbles

comparison pebbles vs. macro:

- n-dPTT $\subseteq$ 0-dPTT^{n+1} $\subseteq$ dMTT^{n+1}
- dMTT $\subseteq$ 0-dPTT^{3}

▶ introducing invisible pebbles

issues - decomposition
       - complexity per pebble

we move to another conference

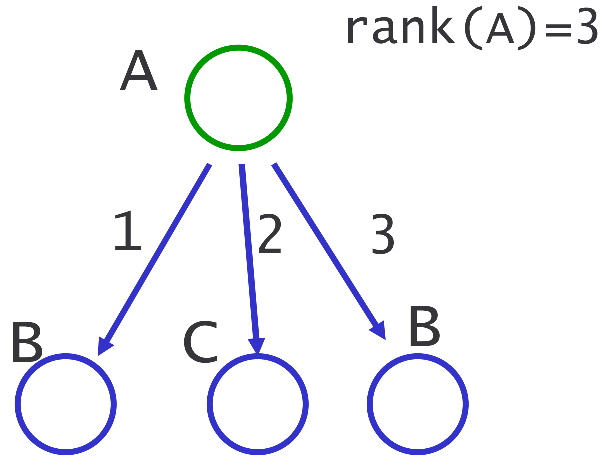
XML Transformation
by
Tree-Walking Transducers
with
Invisible Pebbles

Joost Engelfriet
Hendrik Jan Hoogeboom
Bart Samwel
(Leiden University, NL)

PODS Beijing June 2007

tree model

<A>
<B> …
</B>
<C> …
</C>
<B> …
</B>
</A>

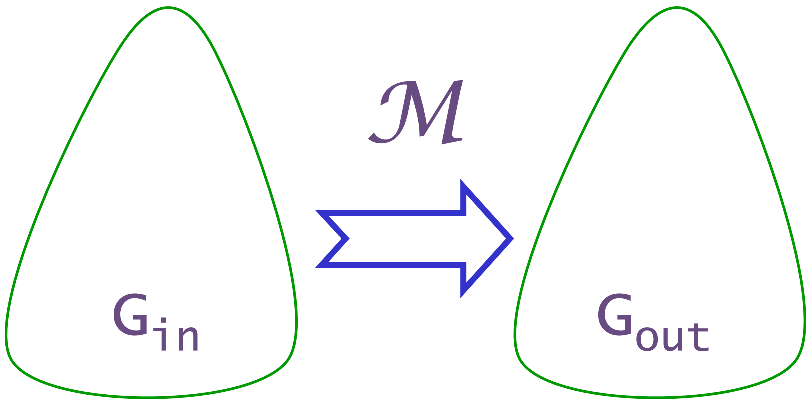


ranked trees
node labels with rank

unbounded number of children
(forests) are to be coded
[*usually*] this is no problem

typechecking

decide whether tree (document) generated by transformation $\mathcal{M}$ satisfies description



Milo Suciu Vianu PODS2000

type checking for XML transformers is decidable

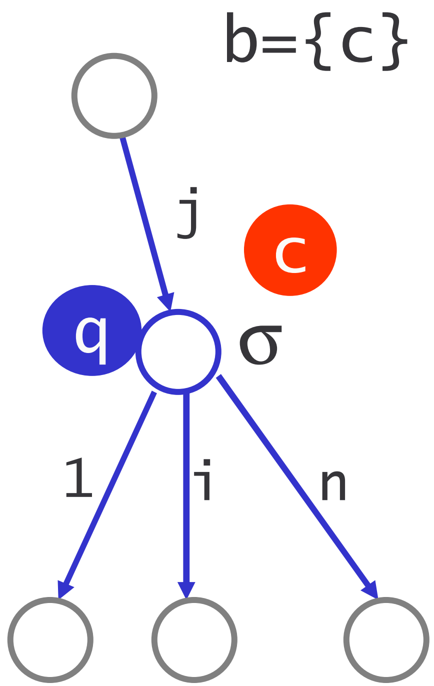
transformers with ‘visible’ pebbles:
finite number of coloured markers on tree

contents

1. automata with pebbles
2. decomposition
3. typechecking
4. regular trees
5. document navigation
6. pattern matching
7. conclusion

tree-walking automata

with pebbles



local configuration

q state

σ node label

j child number

j=0 root

b pebble colours

$b \subseteq C$

instructions

$(q,\sigma,b,j) \rightarrow$

(halt)

(q',stay)

(q',up)

(q',$down_i$)

(q',$drop_c$)

(q',$lift_c$)

- finite set C of pebbles
- nested lifetimes
  stack behaviour
  only topmost can be lifted
- all observable

tree-walking pebble automata

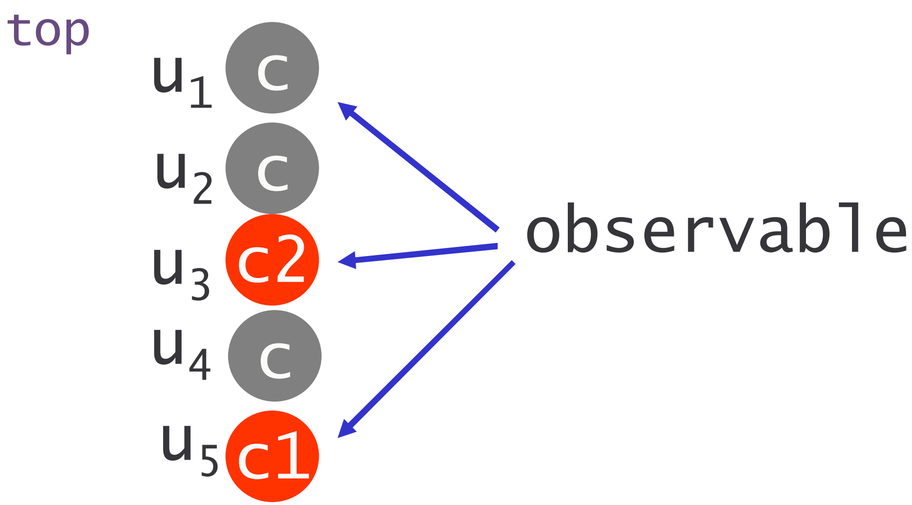
with visible pebbles
'colours' used once
always observable

☹ do not recognize all
regular tree languages
≡ MSO properties

we add invisible pebbles
colours used many times
only topmost is observable

☺ recognize regular
& decidable type checking
& better complexity

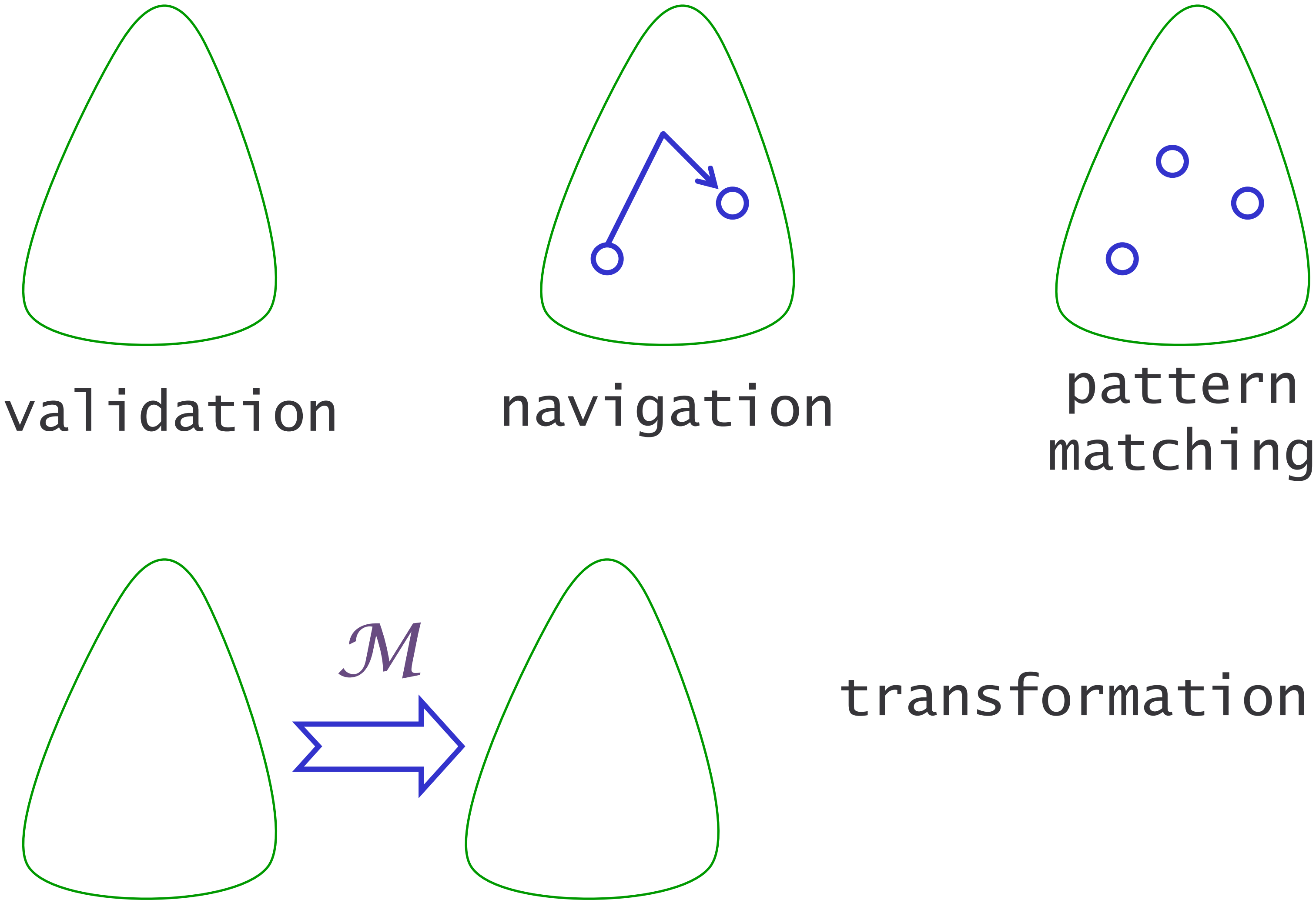
stack behaviour of pebbles!
(avoid 'counting')



$(q,\sigma,b,j) \to (q',stay)$

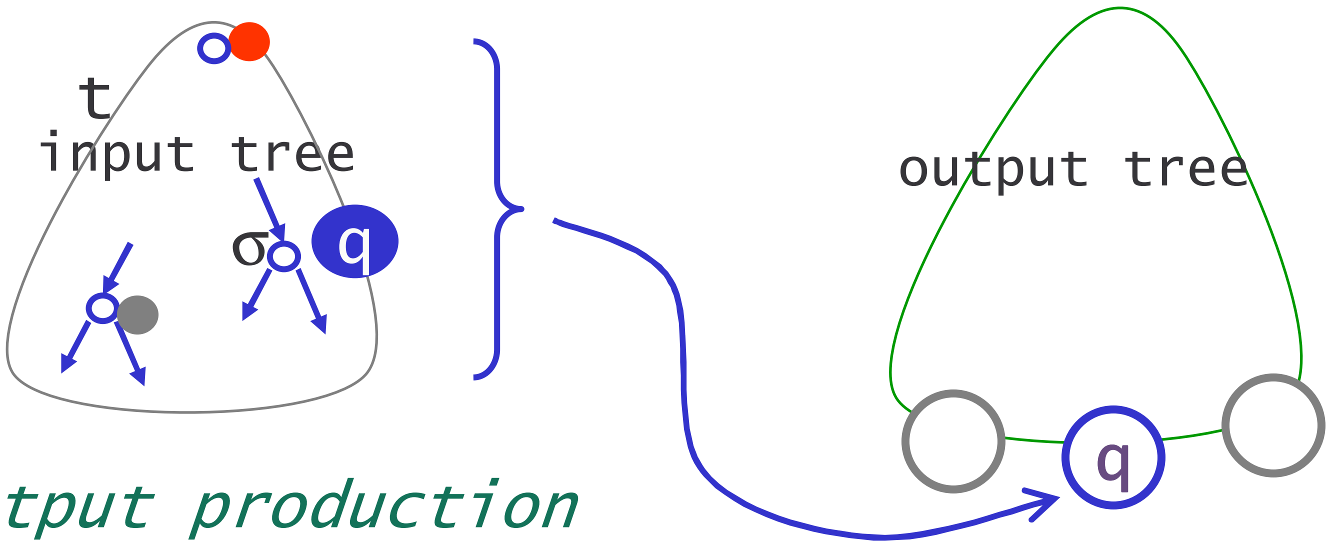
b contains
- all visible pebbles
- invisible when topmost

automaton defines ...



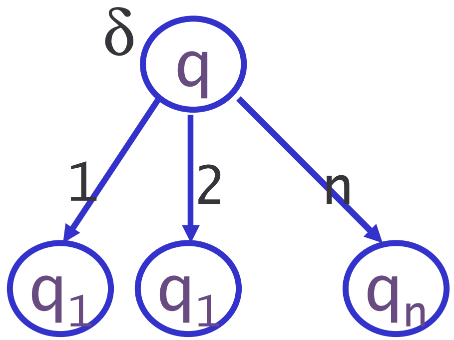
tree-walking pebble tree *transducers*

recursively generate output



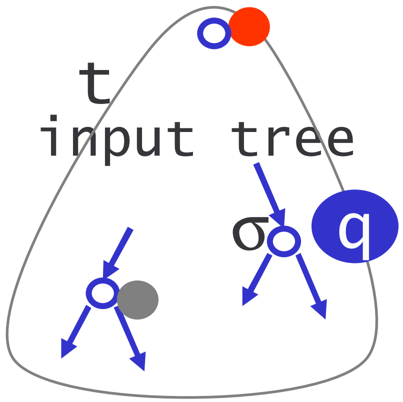
output production

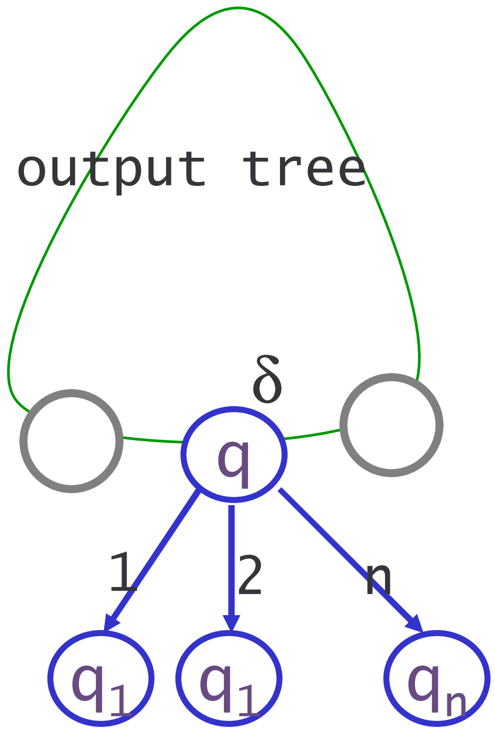
$(q,\sigma,b,j) \rightarrow \delta(q_1,q_2 \dots q_n)$



tree-walking pebble tree transducers

recursively generate output





output production

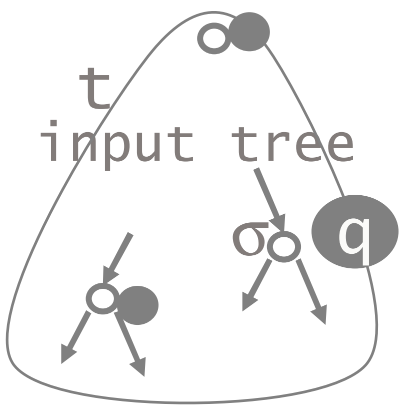
$$(q,\sigma,b,j) \rightarrow \delta(q_1,q_2 \ldots q_n)$$

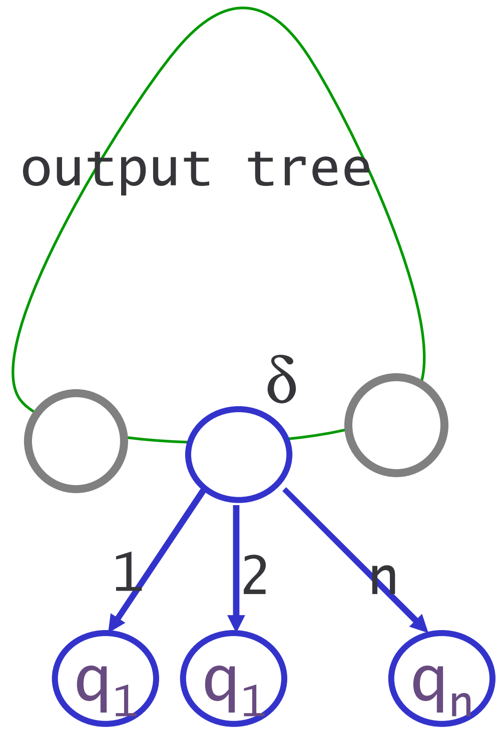
NOTE

each q works on separate copy input tree
- tdt – q_i point to children (↓)
- twt – q_i point to same node
  q's may move up↑ and down↓ in between

tree-walking pebble tree transducers

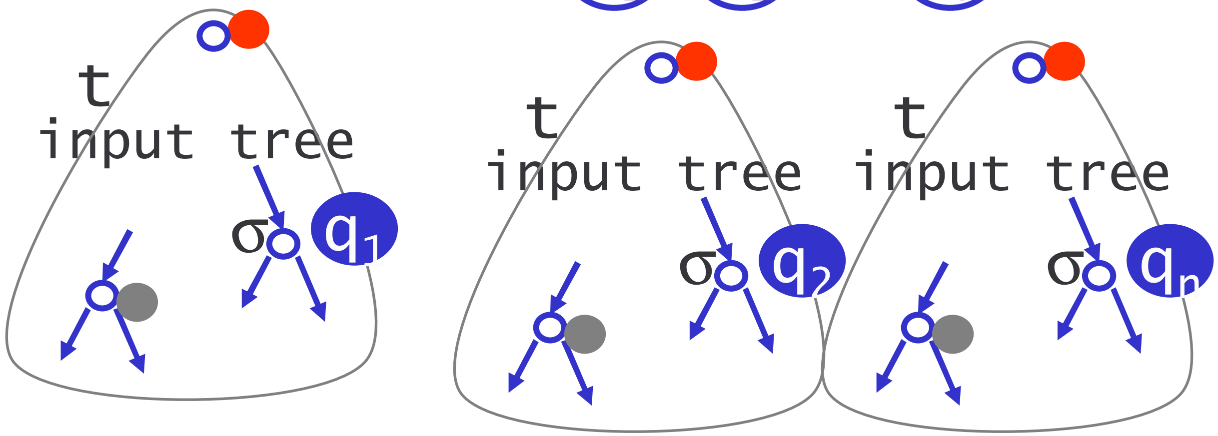
recursively generate output



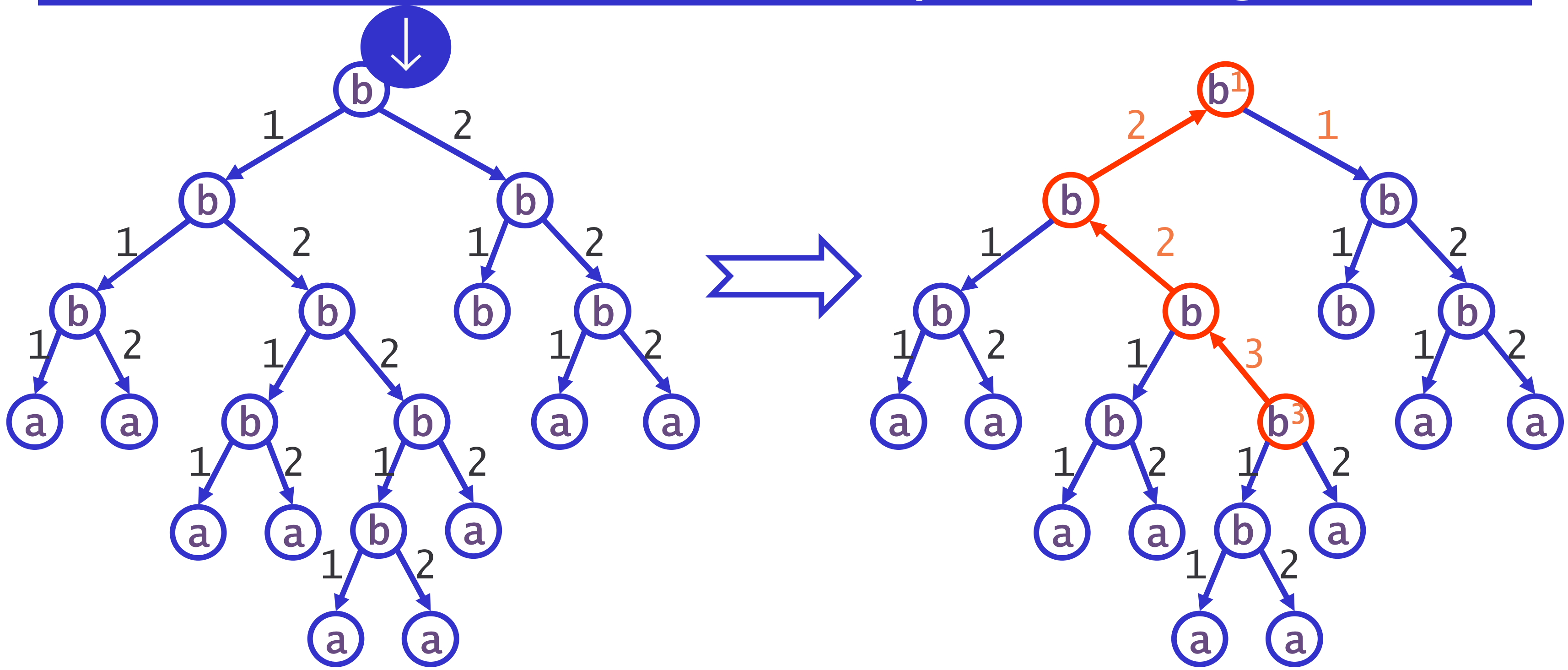


output production

$$(q,\sigma,b,j) \rightarrow \delta(q_1,q_2 \ \dots \ q_n)$$



example: moving the root



walk down

$(\downarrow, b, -, j) \rightarrow (\downarrow, down_1)$

$(\downarrow, b, -, j) \rightarrow (\downarrow, down_2)$

copy up

$(\uparrow, b, -, 1) \rightarrow b(\uparrow_1, c_2)$

$(\uparrow, b, -, 2) \rightarrow b(c_1, \uparrow_2)$

$(\uparrow_i, b, -, i) \rightarrow (\uparrow, up)$

copy down

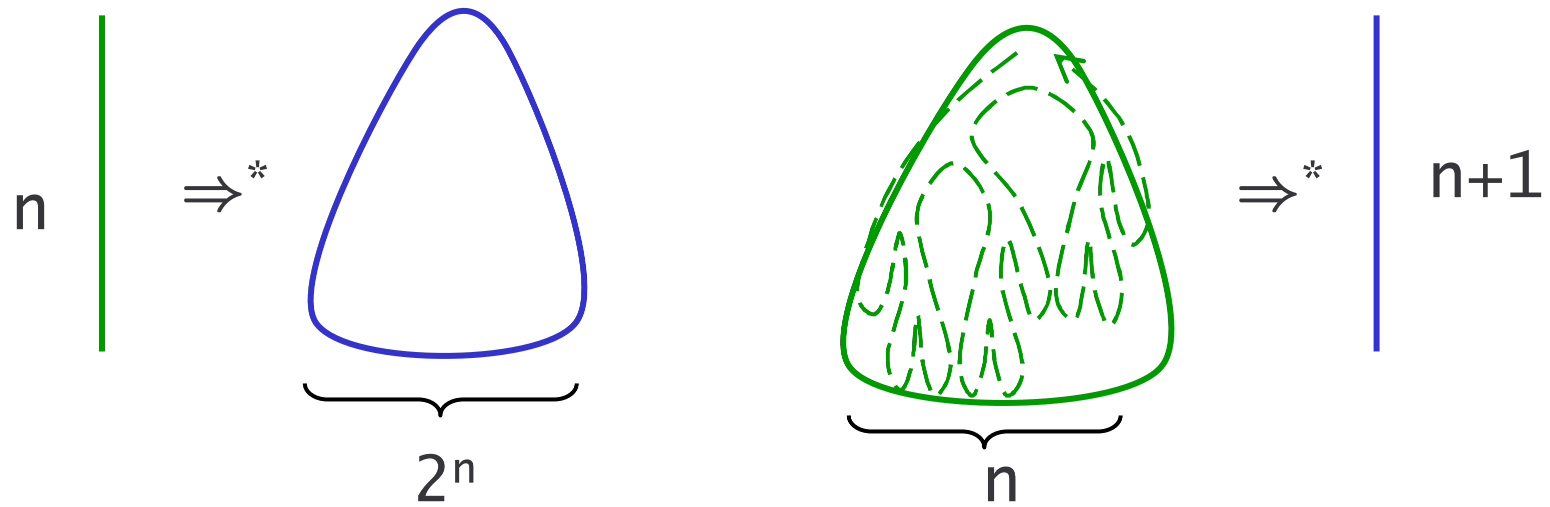
$(copy, a, -, j) \rightarrow a()$

$(copy, b, -, j) \rightarrow b(c_1, c_2)$

$(c_i, b, -, j) \rightarrow (copy, down_i)$

$j=0,1,2 \quad i=1,2$

the power of composition



together: exponential size-to-height

n-PTT: polynomial size increase

notation

Pebble Tree Transducers

V_kI-PTT	visible + invisible	
V_k-PTT	k visible pebbles	Milo etal.
I-PTT	invisible only	
TT	tree-walking (no pebbles)	

Pebble Tree Automata

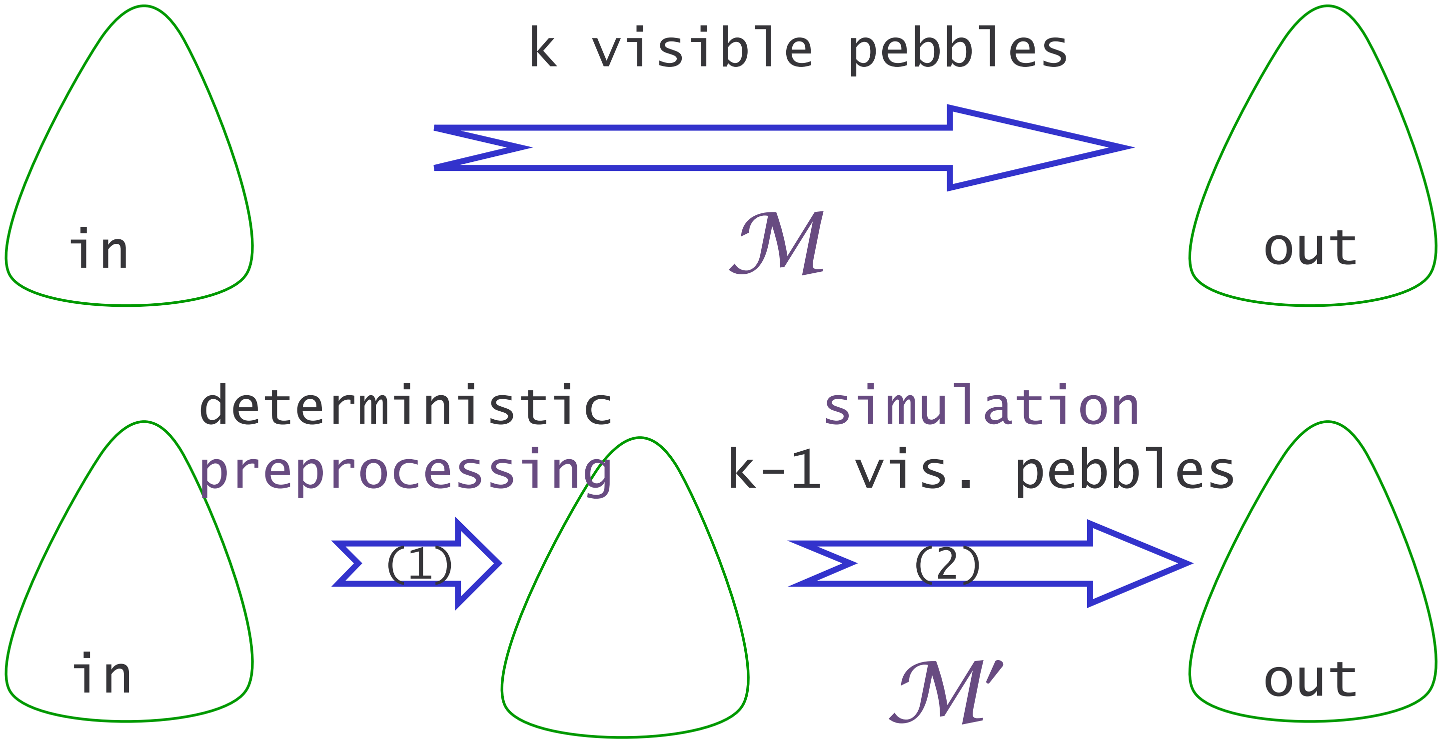
V_kI-PTA
V_k-PTA
I-PTA

contents

1. automata with pebbles
2. decomposition
3. typechecking
4. regular trees
5. document navigation
6. pattern matching
7. conclusion

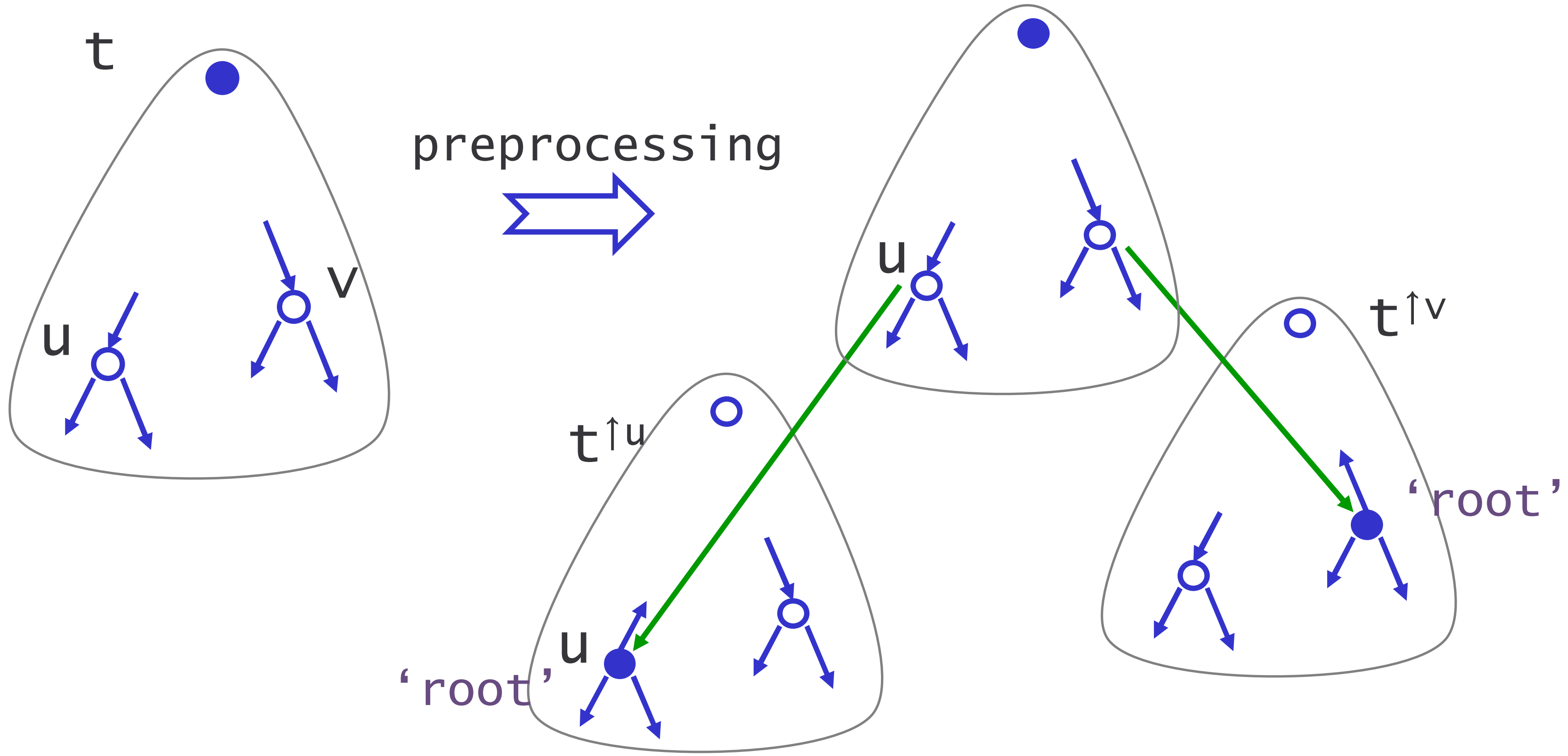
decomposition visible pebbles

$$V_k\text{I-dPTT} \subseteq \text{dTT} \circ V_{k-1}\text{I-dPTT}$$



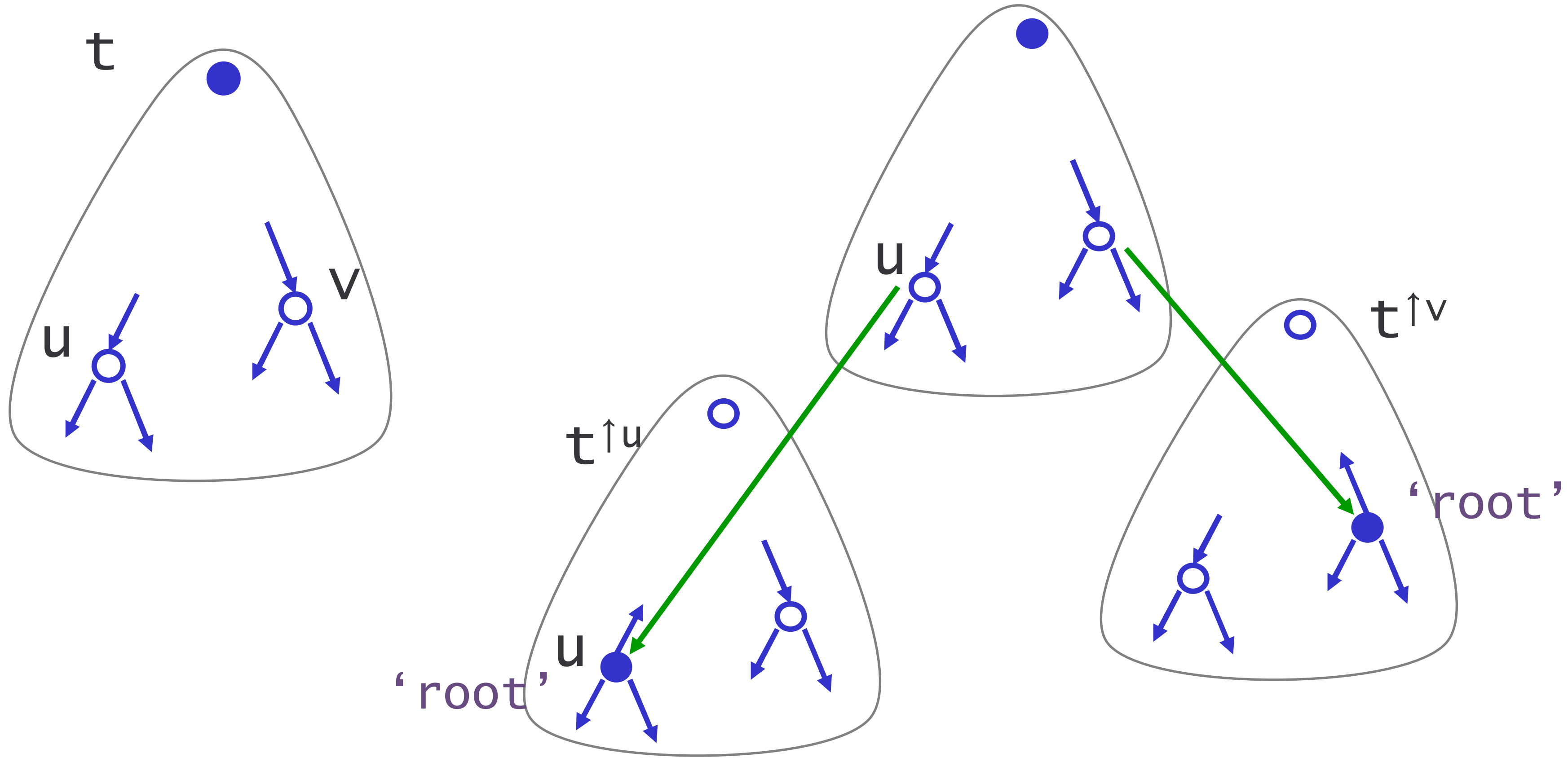
iterate $V_k\text{I-dPTT} \subseteq \text{dTT}^k \circ \text{I-dPTT}$

decomposition (1) *preprocessing*



copying can be done without pebbles

decomposition (2) *simulation*



$\mathcal{M}$

drop / lift
first visible pebble

$\mathcal{M}'$

move up /down
into subtree

decomposition

V_kI-dPTT ⊆ dTT ∘ V_{k-1}I-dPTT

I-dPTT ⊆ TT ∘ dTT

(deterministic)

THEOREM

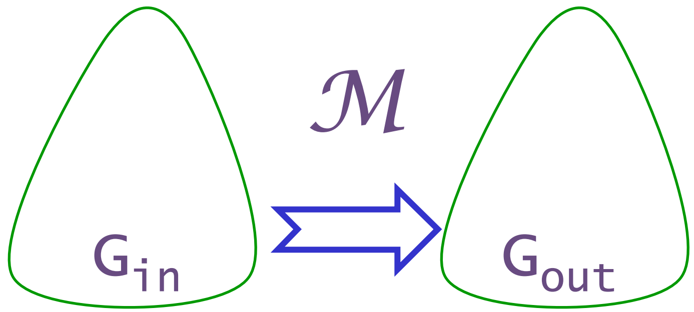
V_k-PTT ⊆ TT^{k+1}

V_kI-PTT ⊆ TT^{k+2}

contents

inverse type inference

given transducer $\mathcal{M}$ and regular G_{out},
construct regular G_{in} such that
$L(G_{in}) = \mathcal{M}^{-1}\ L(G_{out})$



Bartha 1982

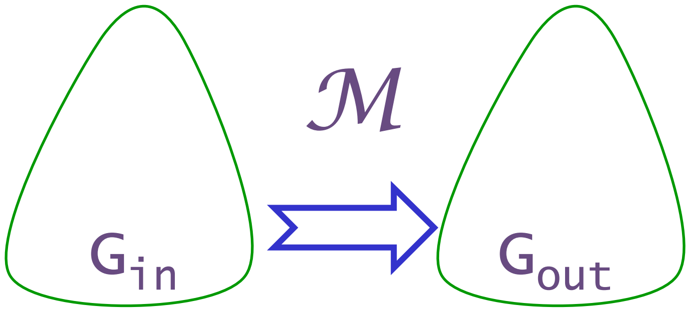
regular tree grammar G for the domain
of tree transducer $\mathcal{M}$ can be constructed
in *exponential* time

inverse type inference is solvable
⇒ for TT in exponential time
⇒ for TT^k in k-fold exponential time

type checking complexity

type checking

given transducer $\mathcal{M}$ and regular G_{in}, G_{out},
decide whether $\mathcal{M}(L(G_{in})) \subseteq L(G_{out})$



$M(A) \subseteq B$ iff $A \cap M^{-1}(B^c) = \varnothing$

'typechecking' 'inverse type inference'

V_k-PTT $\subseteq$ TT^{k+1}

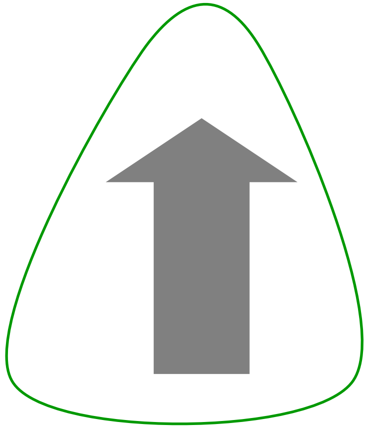
V_kI-PTT $\subseteq$ TT^{k+2}

we can typecheck
- ⇒ TT^k in $(k+1)$-fold exponential time
- ⇒ V_k-PTT in $(k+2)$-fold exponential time
- ⇒ V_kI-PTT in $(k+3)$-fold exponential time

invisible pebbles are almost for free!

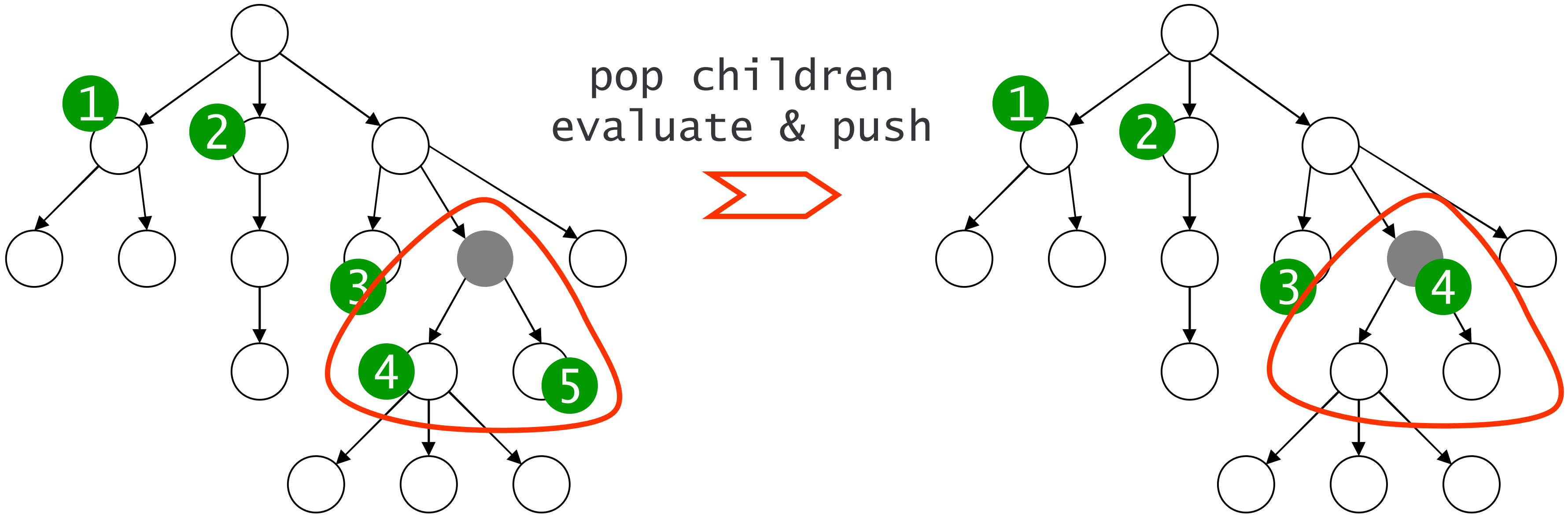
contents

regular trees



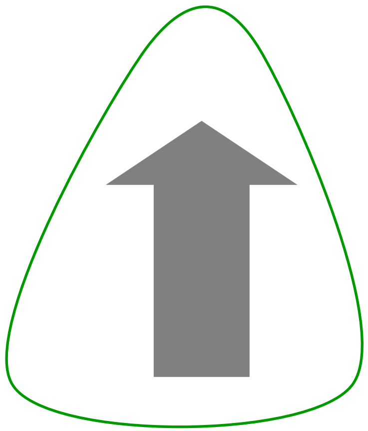
regular tree language
≡ bottom-up tree evaluation
≡ post-order evalation with stack

REGT ⊆ I-PTA



postorder evaluation

regular trees



regular tree language
≡ bottom-up tree evaluation
≡ post-order evalation with stack

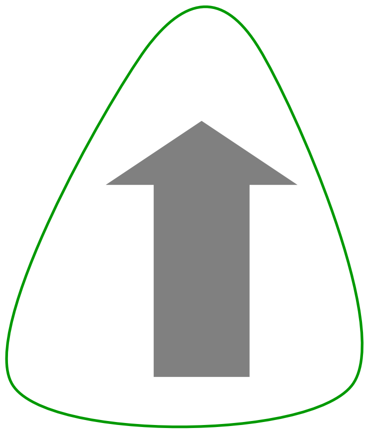
REGT $\subseteq$ I-PTA

REGT $\not\subseteq$ V_k-PTA   Bojańczyk etal.

V_kI-PTT $\subseteq$ TT^{k+2}

V_kI-PTA $\subseteq$ REGT

regular trees



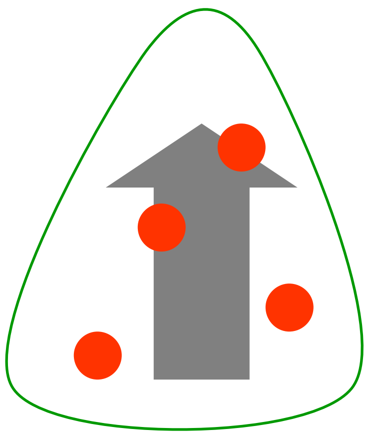
regular tree language
≡ bottom-up tree evaluation
≡ post-order evalation with stack

$\mathrm{REGT} \subseteq \mathrm{I\text{-}PTA}$

$\mathrm{REGT} \not\subseteq \mathrm{V_k\text{-}PTA}$

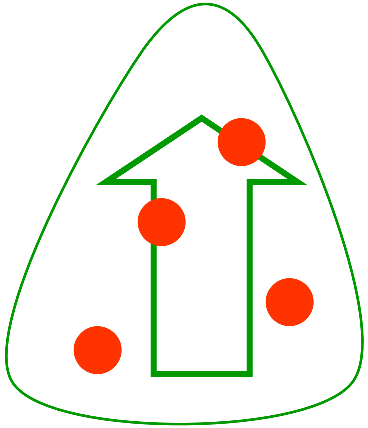
$\mathrm{V_kI\text{-}PTT} \subseteq \mathrm{TT}^{k+2}$

$\mathrm{V_kI\text{-}PTA} \subseteq \mathrm{REGT}$

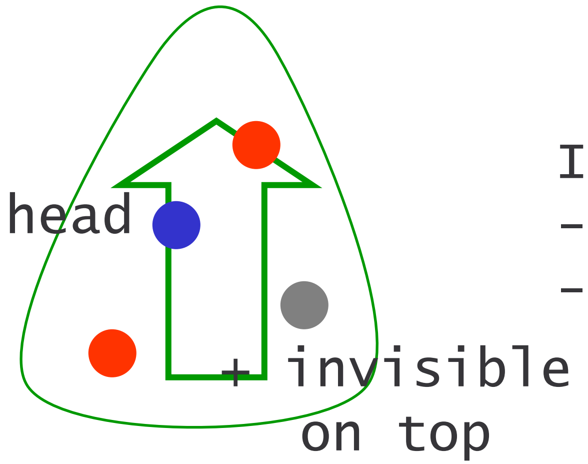


I-PTA can
- evaluate *marked* trees
- test their visible configuration

contents



I-PTA can
- evaluate *marked* trees
- test their visible configuration



I-PTA can
- evaluate *marked* trees
- test their ~~visible~~ observable configuration

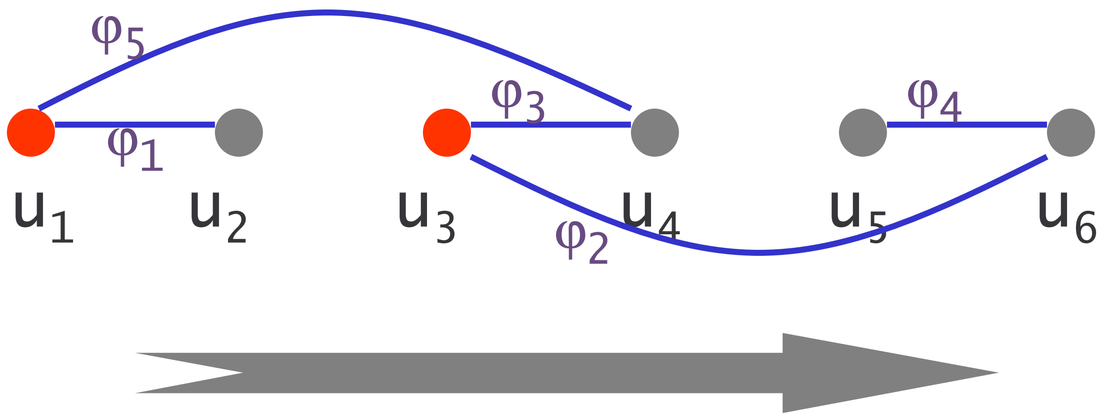
VI-PTA can test $\varphi(x_1,\dots,x_n)$ with $n-2$ visible pebbles (using head) general test $\varphi(x_1,\ldots,x_n)$

XQuery **for** $x_1,\ldots,x_n$ **with** $\varphi_1\wedge\ldots\wedge\varphi_n$ **return** t

φ_i binary

example

$\varphi_1(x_1,x_2)\wedge\varphi_2(x_3,x_6)\wedge\varphi_3(x_4,x_3)\wedge\varphi_4(x_5,x_6)\wedge\varphi_5(x_1,x_4)$



only 2 visible pebbles!

contents

conclusion

- extends known models

V-PTT Milo,Suciu,Vianu
I-PTT = TL Maneth etal. PODS'05
DTL document transformation language

- MSO complete

- invisible pebbles are cheap

\end{document}