Trees and Invisible Pebbles

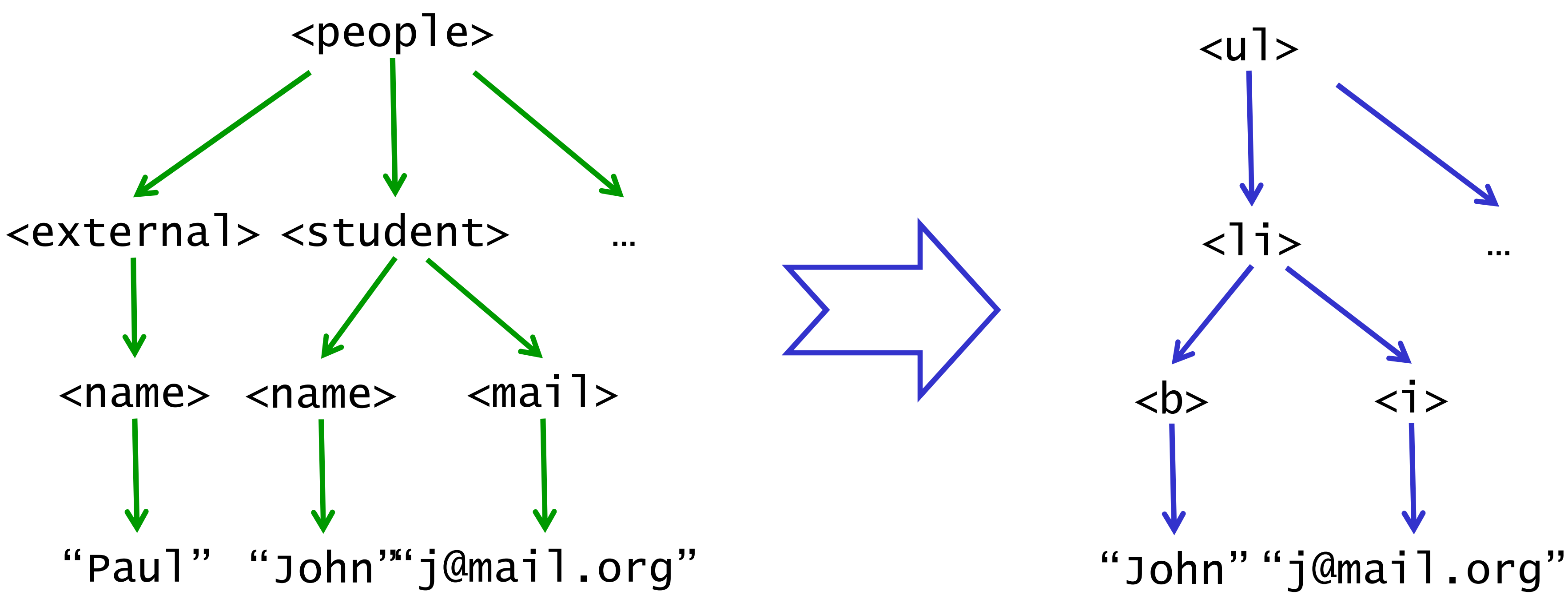
Joost Engelfriet
Hendrik Jan Hoogeboom

Universiteit Leiden

AutoMathA, Liège, June 2009

document transformation

Gauwin Niehren Tison: Earliest query answering …

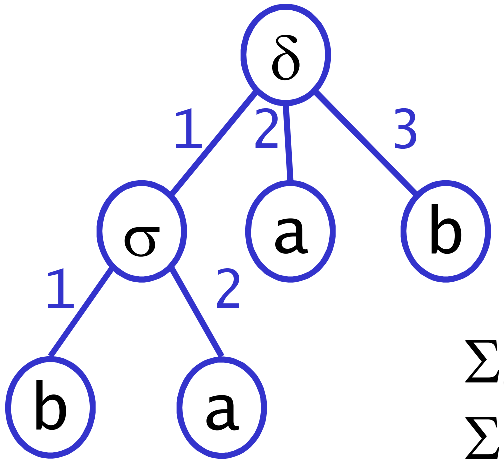


select nodes
& reformat

tree model

ranked trees ~ terms

ranked alphabet
(Σ, rank)
$\text{rank} : \Sigma \to \mathbb{N}$
Σ_k rank k

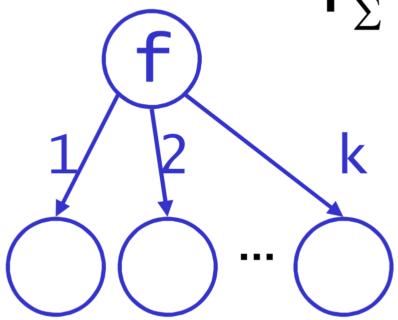


$\delta(\sigma(ba)ab)$
$\delta\sigma baab$

$\Sigma_0 = \{a,b\}$
$\Sigma_2 = \{\delta\}$
$\Sigma_3 = \{\sigma\}$

T_Σ trees over Σ

child number
child-sibling coding



$f \in \Sigma_k$
$f(x_1 x_2 \ldots x_k)$
$f x_1 x_2 \ldots x_k$

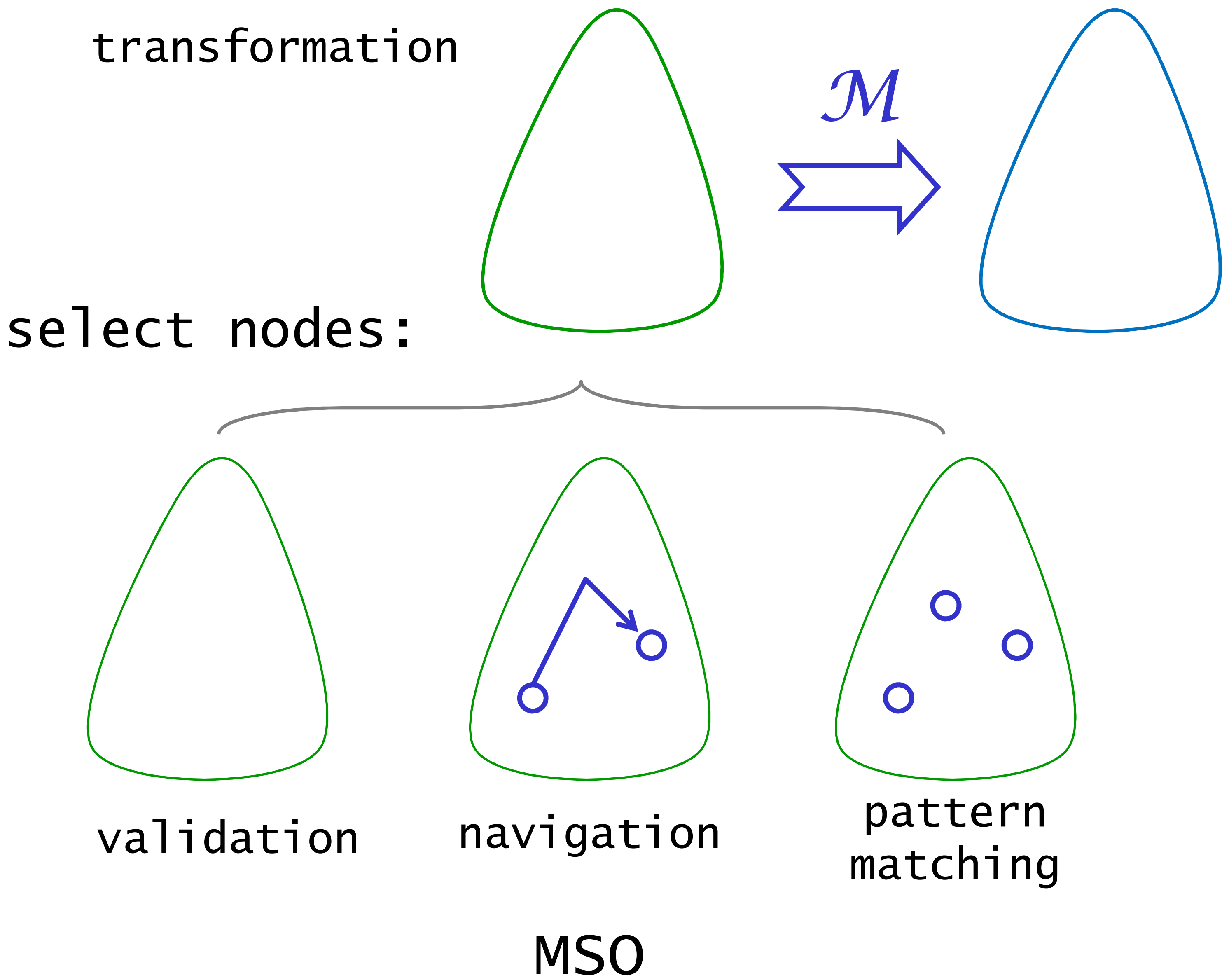
contents

introduction: finding the right model

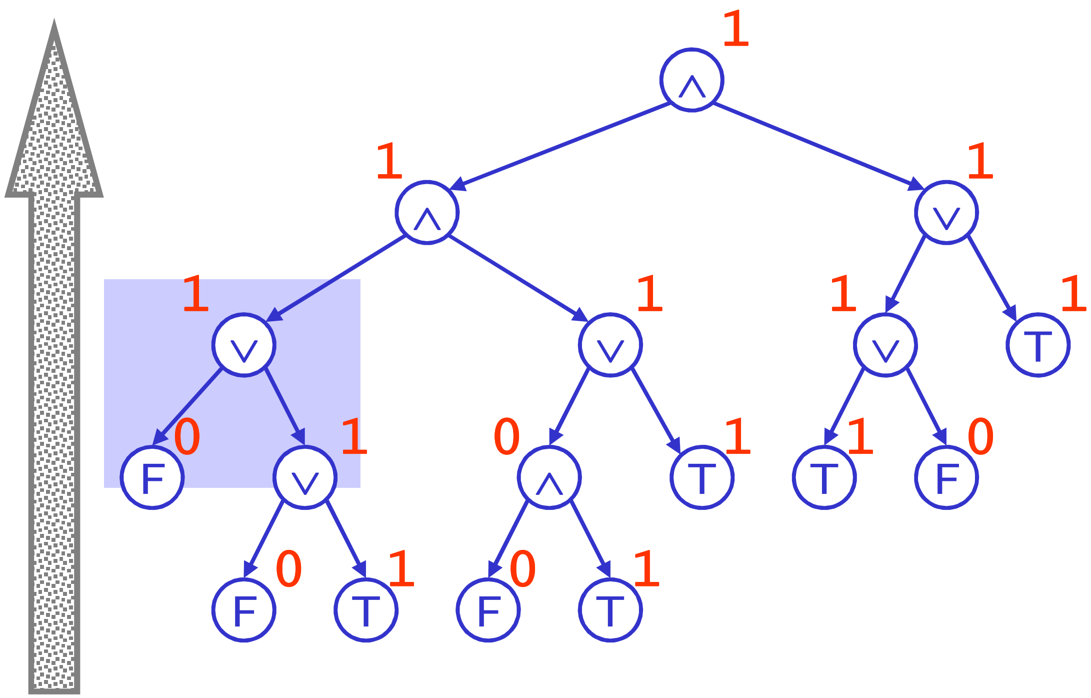
tree walking transducers
with invisible pebbles

technical results:
decomposition
type checking & regularity
pattern matching

connections to logic
complexity
'real' XML

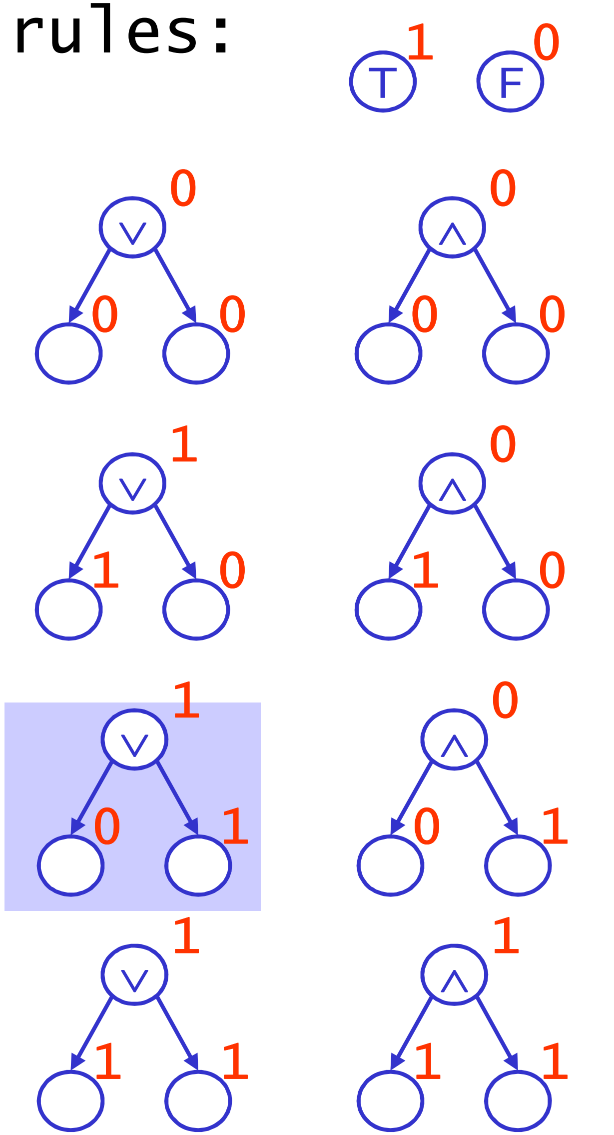

transformation
$\mathcal{M}$
select nodes:
validation
navigation
pattern
matching
MSO

bottom-up tree automaton

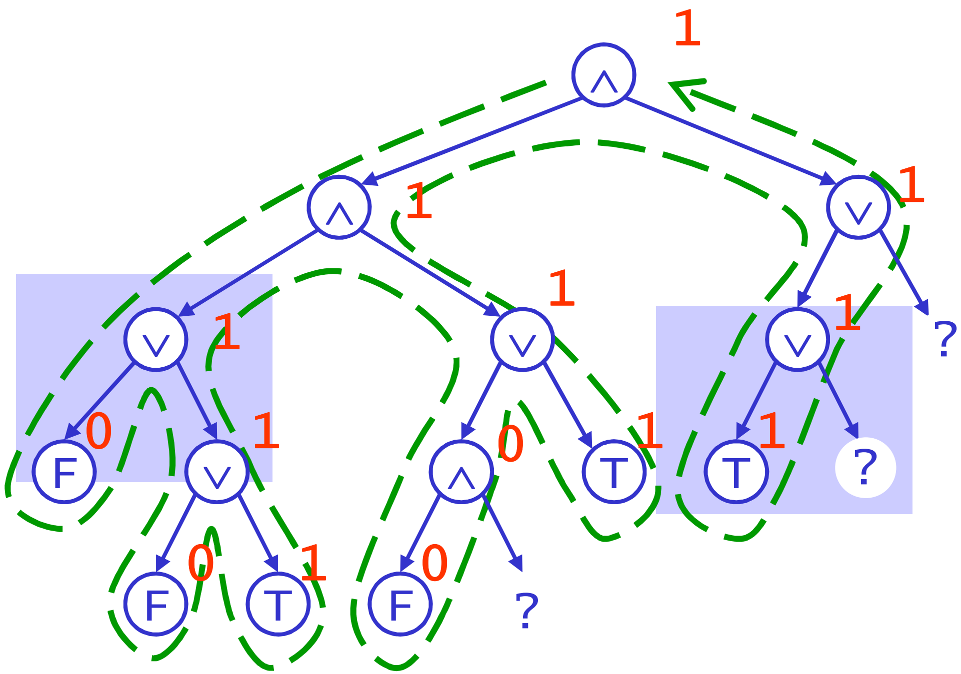


bottom-up evaluation

rules:



tree walking automaton

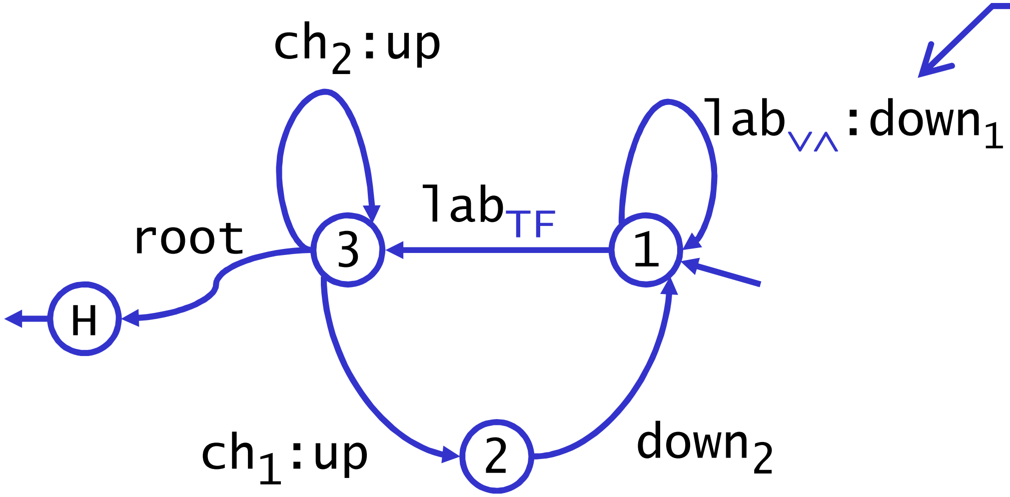


walk along edges

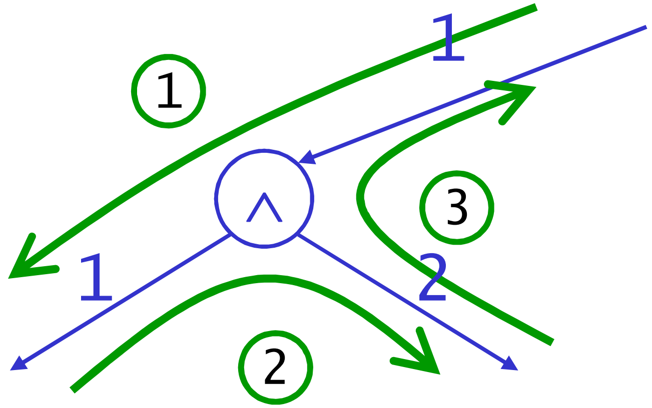
cf. two-way finite state automaton

tree walking automaton

example: pre order tree traversal



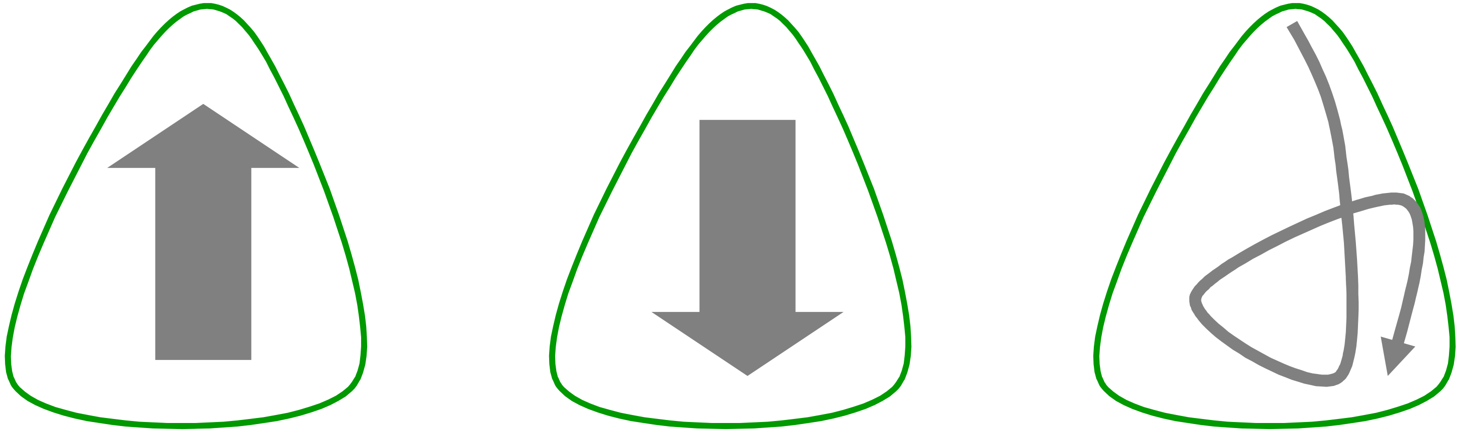
when label is ∨ or ∧
move to first child



walk along edges, moves based on

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

tree automata



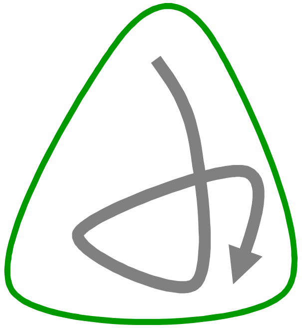
bottom-up	top-down	tree-walking
evaluation	*grammatical*	*navigation*

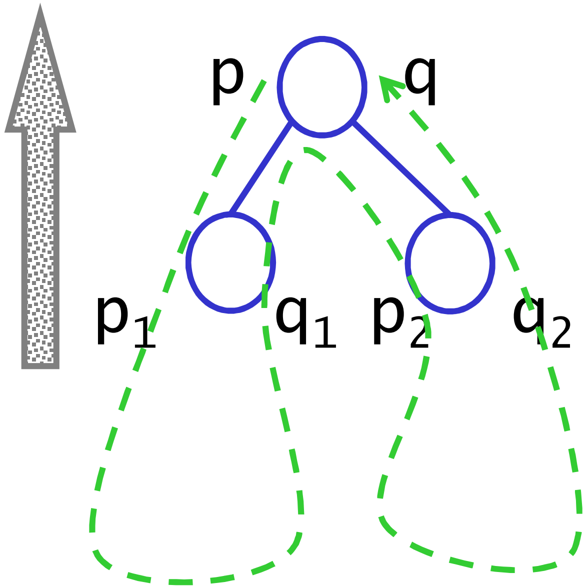
REG ≡ MSO ?

need to:

- verify input tree
- select nodes (both based on a MSO property)

power of tree walking automata





TWA $\subseteq$ REG

state pairs

start/end computation
below node

TWA $\subset$ REG

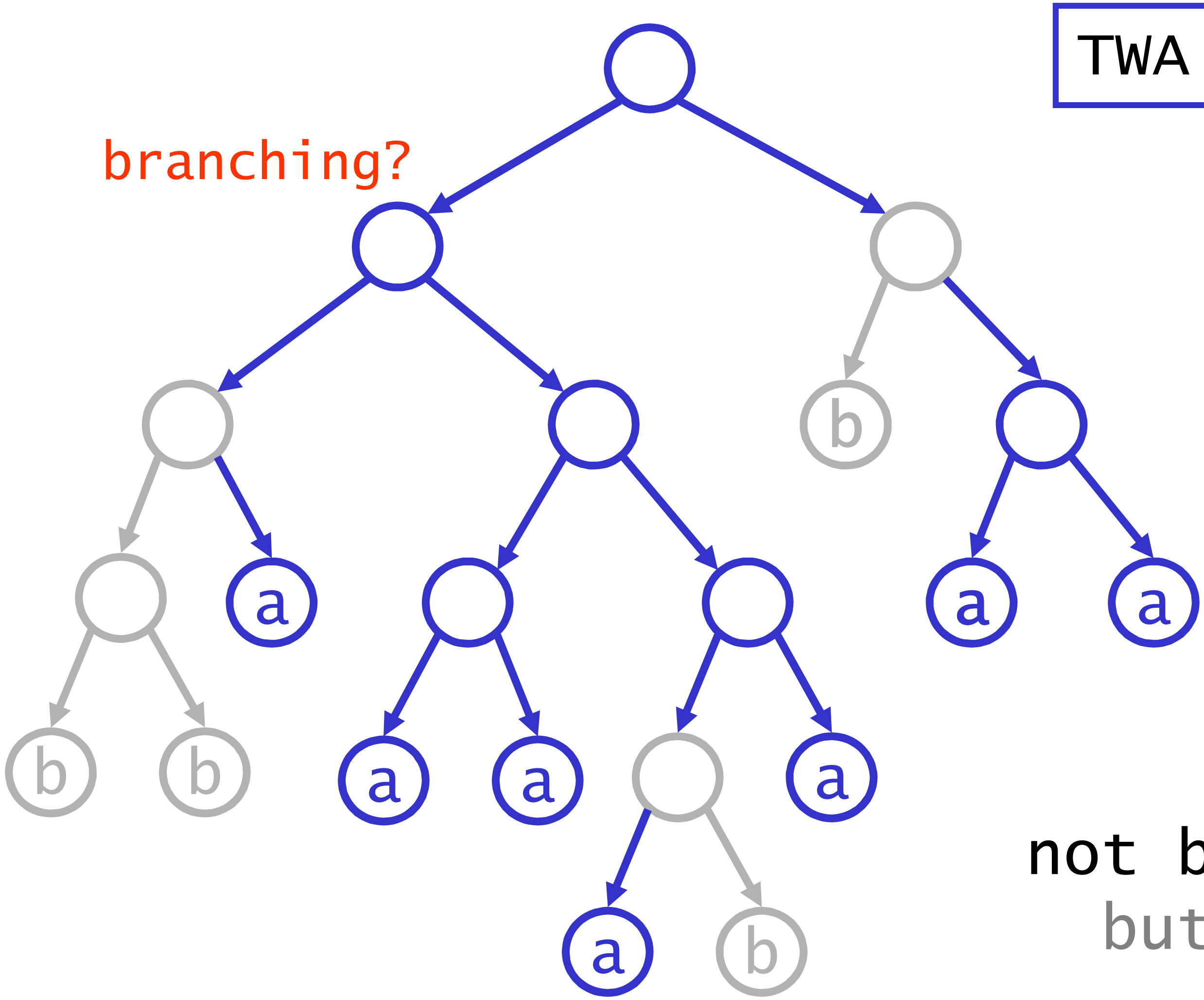
Bojańczyk & Colcombet STOC'05

"tree walking automata easily loose their way"

‘branching structure’ of even length

Bojańczyk & Colcombet

TWA ⊂ REG



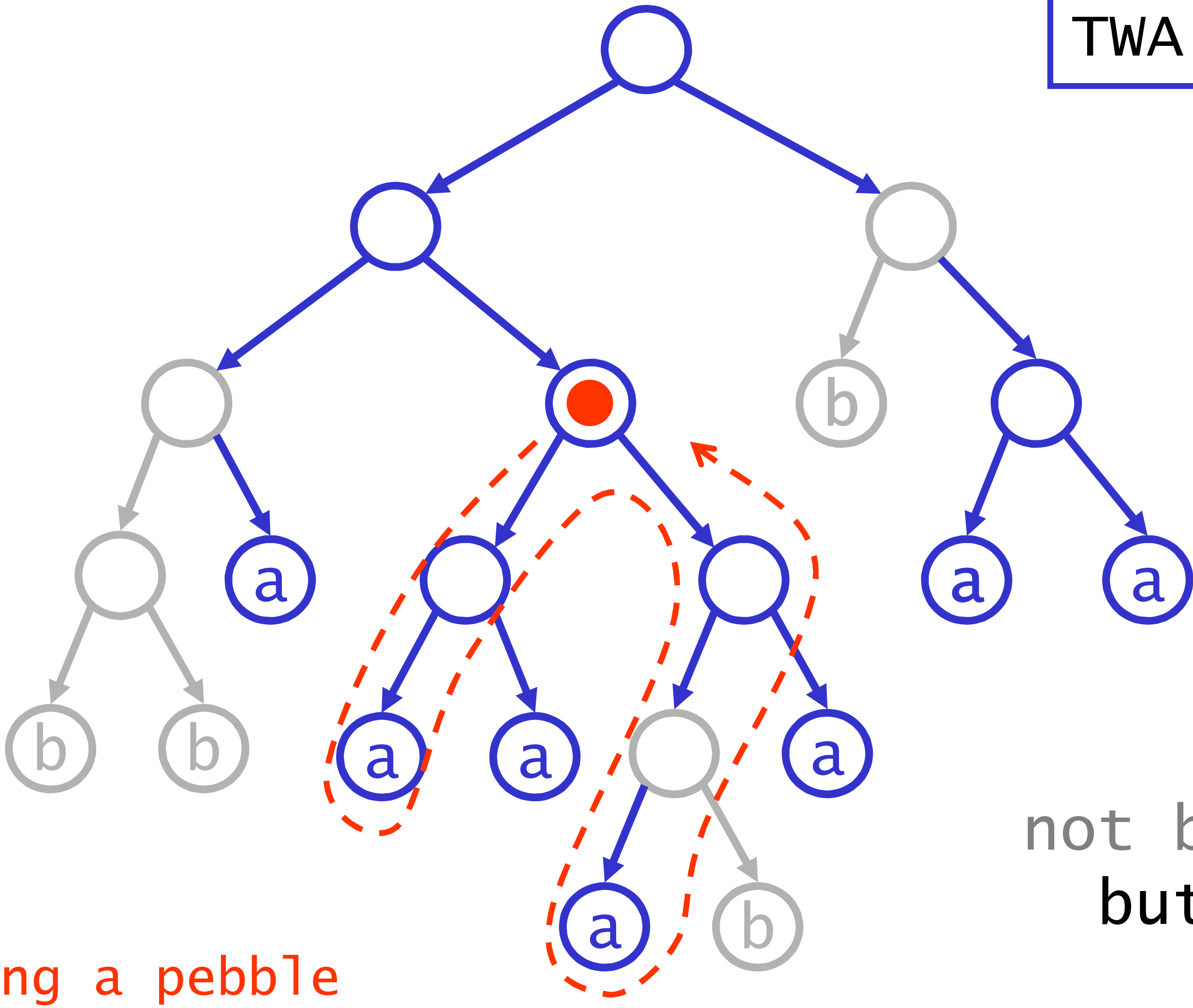
not by TWA

but FO

pebbles to mark nodes

Bojańczyk & Colcombet

TWA ⊂ REG

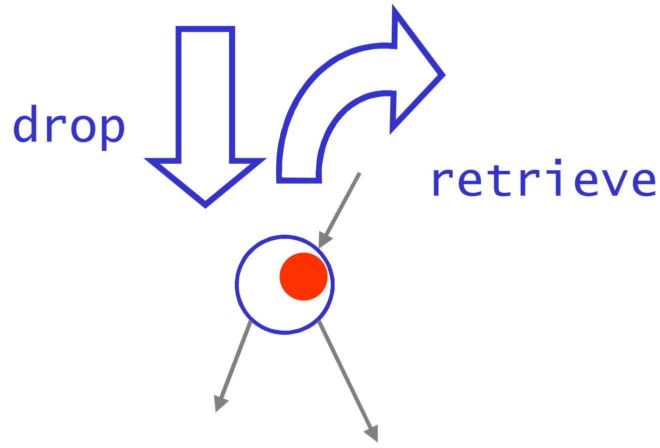


not by TWA
but PTWA

using a pebble
to determine branching

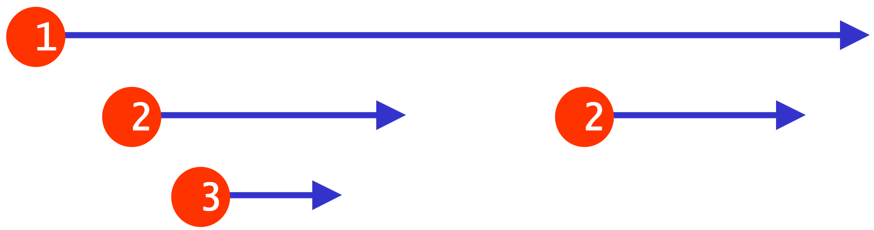
adding nested pebbles to the TWA

pebble: mark a node



- fixed number for automaton
- can be distinguished & reused
- used to determine where to go

- *nested lifetimes* ‘stack discipline’

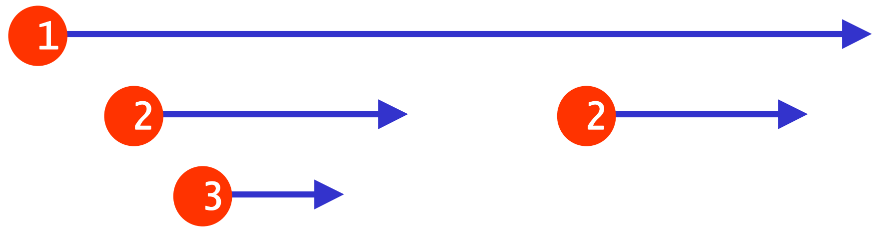


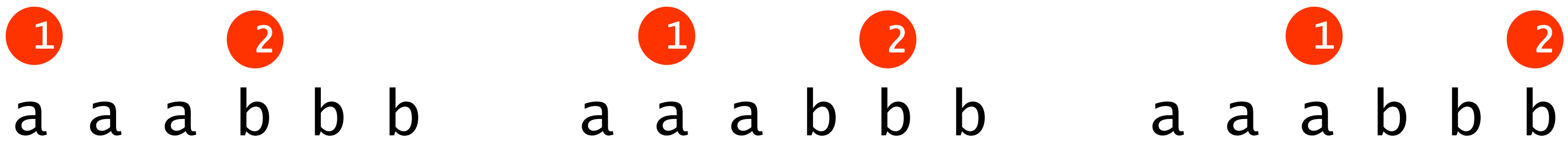
‘regular’ extension

PTWA $\subseteq$ REG

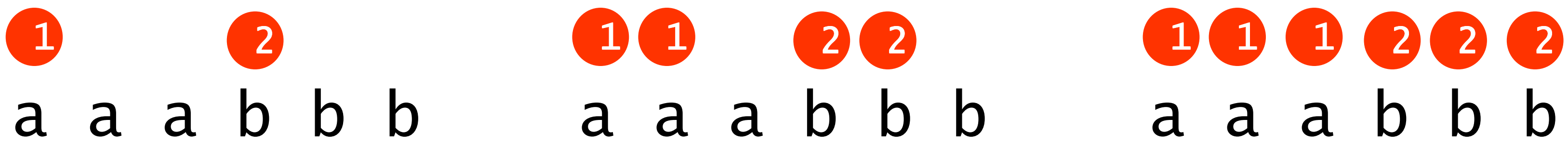
beware of the pebble

avoid counting





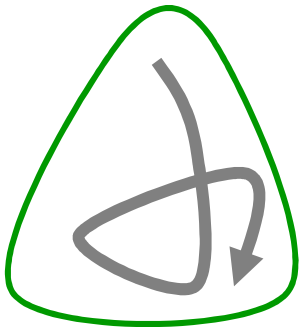
► nest!



► bounded number!

power of tree walking automata

pebble



TWA $\subseteq$ REG

TWA $\subset$ REG

Bojańczyk & Colcombet STOC'05

PTWA $\subseteq$ REG

Engelfriet & H '99

PTWA $\subset$ REG

Bojańczyk, Samuelides,
Schwentick & Segoufin ICALP'06

"tree walking automata easily loose their way"
(even with the help of pebbles)

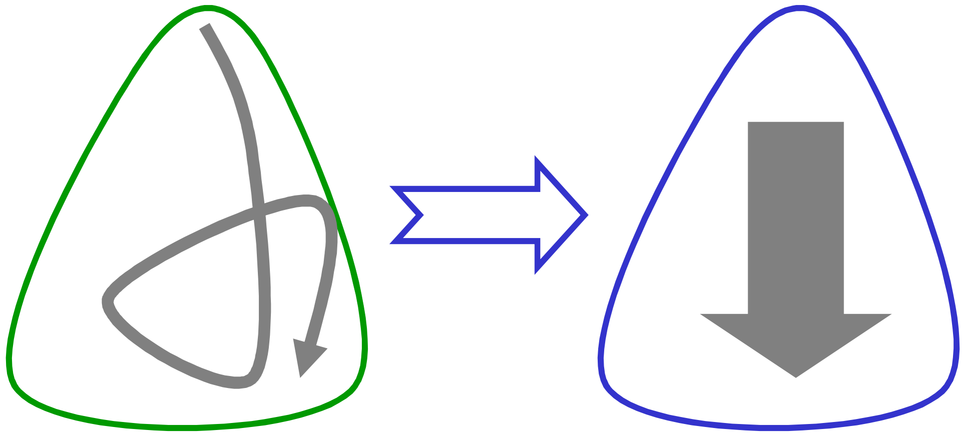
contents

introduction: finding the right model

tree walking transducers
with invisible pebbles

technical results:
- decomposition
- type checking & regularity
- pattern matching

model for tree translations



Aho Ullman 1971
translations on a context-free grammar

TWTT

Milo Suciu Vianu PODS2000
type checking for XML transformers is decidable

+ pebbles

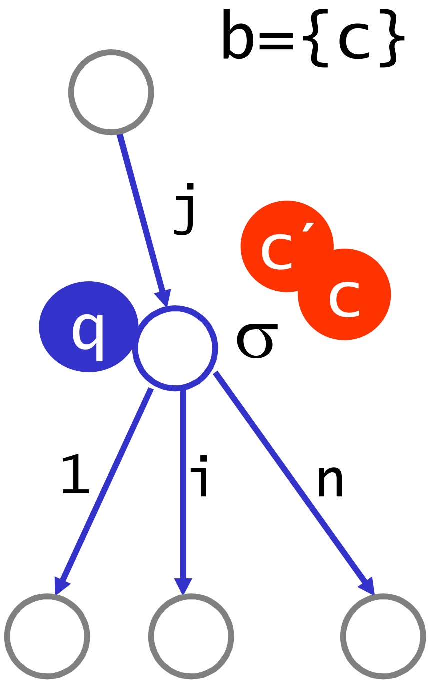
Engelfriet & H & Samwel PODS2007

+ ‘invisible’ pebbles

Slutzki 1985
‘two-way backtracking pushdown tree automata’

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
$(q,\sigma,B,j) \to$
(halt)
(q',stay)
(q',up)
$(q',down_i)$

$(q',drop_c)$
$(q',lift_c)$

- finite set C of pebbles 'colours'
- nested lifetimes: distributed stack
  only topmost can be lifted
- classical: all observable, finite
- set: keep order in finite state

tree-walking pebble automata

C with visible pebbles
'colours' used once
always observable

☹ do not recognize all
regular tree languages
≡ MSO properties

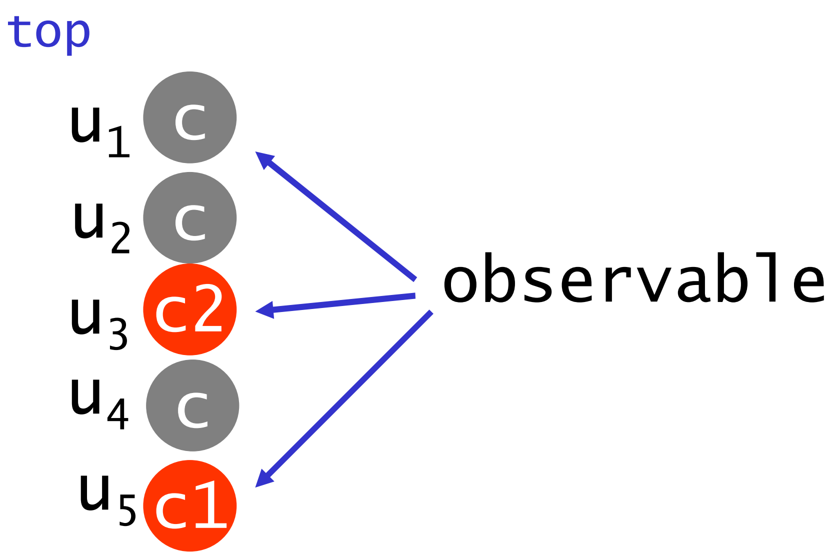
tree-walking pebble automata

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

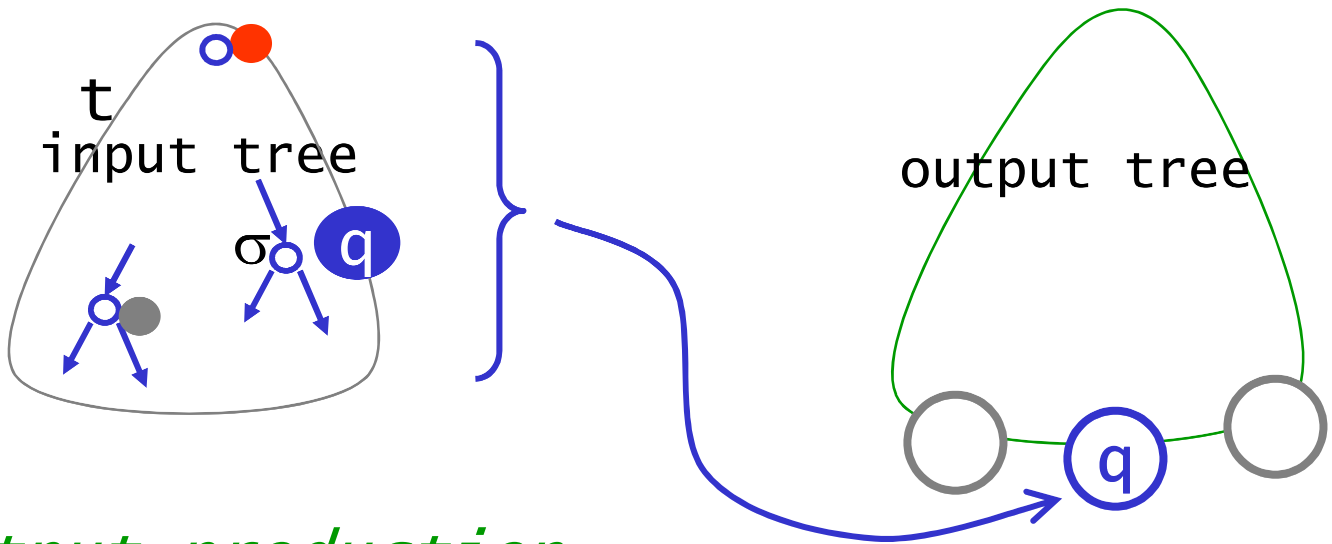


$(q,\sigma,B,j) \rightarrow (q',\text{stay})$

B contains
- all visible pebbles
- invisible when topmost

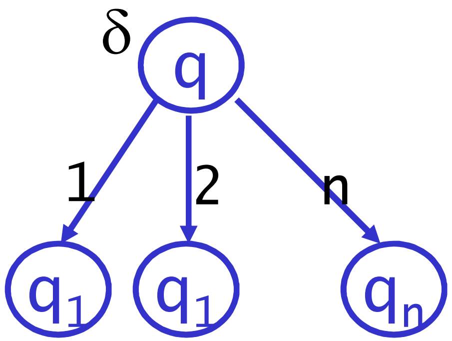
tree-walking pebble tree *transducers*

recursively generate output



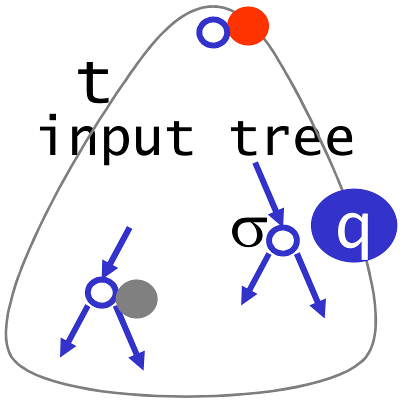
output production

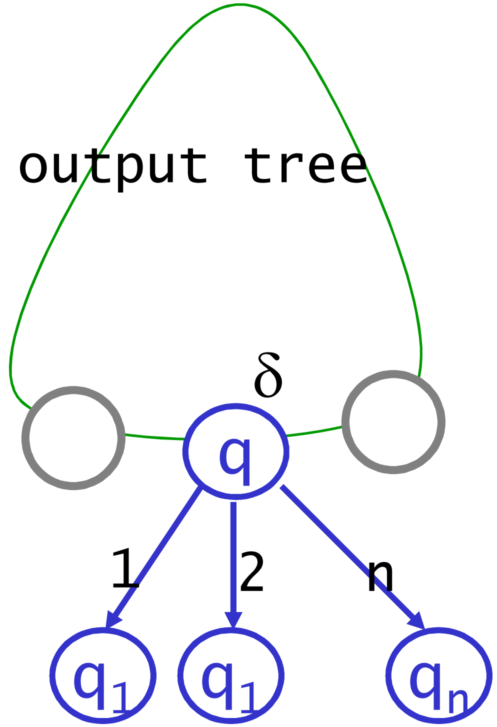
$$(q,\sigma,B,j) \rightarrow \delta(q_1,q_2 \ldots q_n)$$



tree-walking pebble tree transducers

recursively generate output





output production

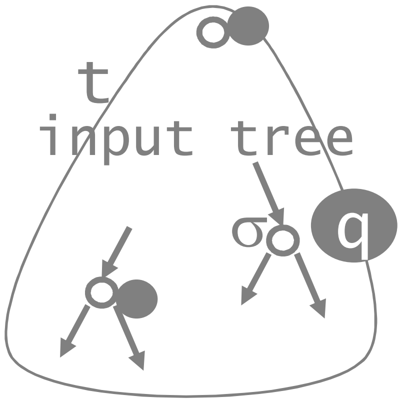
$(q,\sigma,B,j) \rightarrow \delta(q_1,q_2 \dots q_n)$

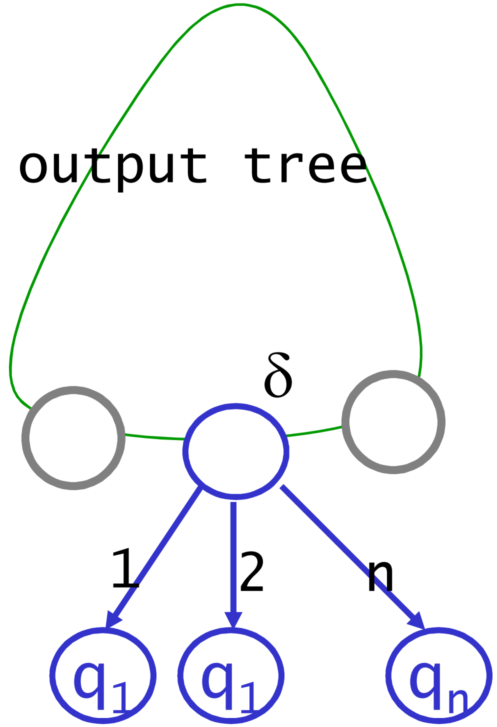
each q works on separate copy input tree

- tdtt - q_i point to children (↓)
- twtt - q_i point to same node
  q's may move up↑ and down↓ in between

tree-walking pebble tree transducers

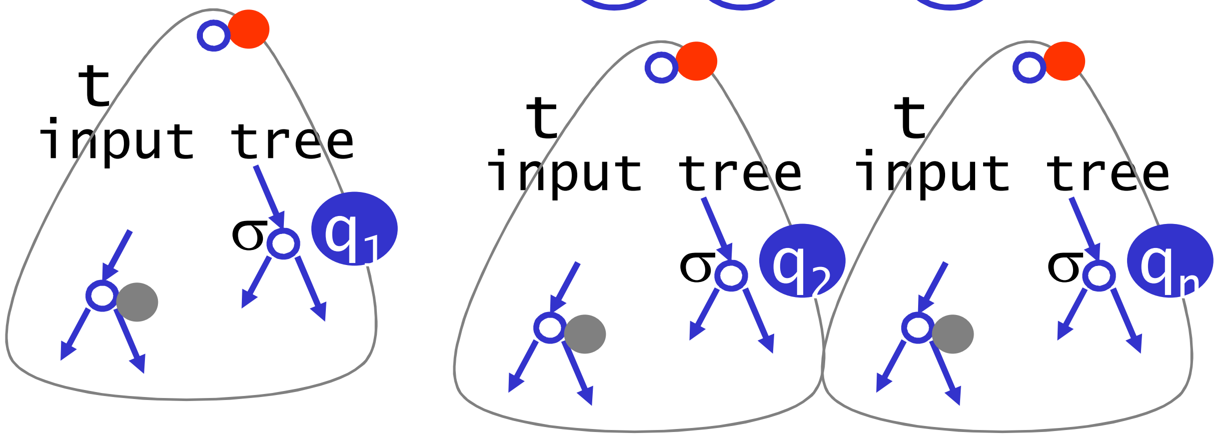
recursively generate output



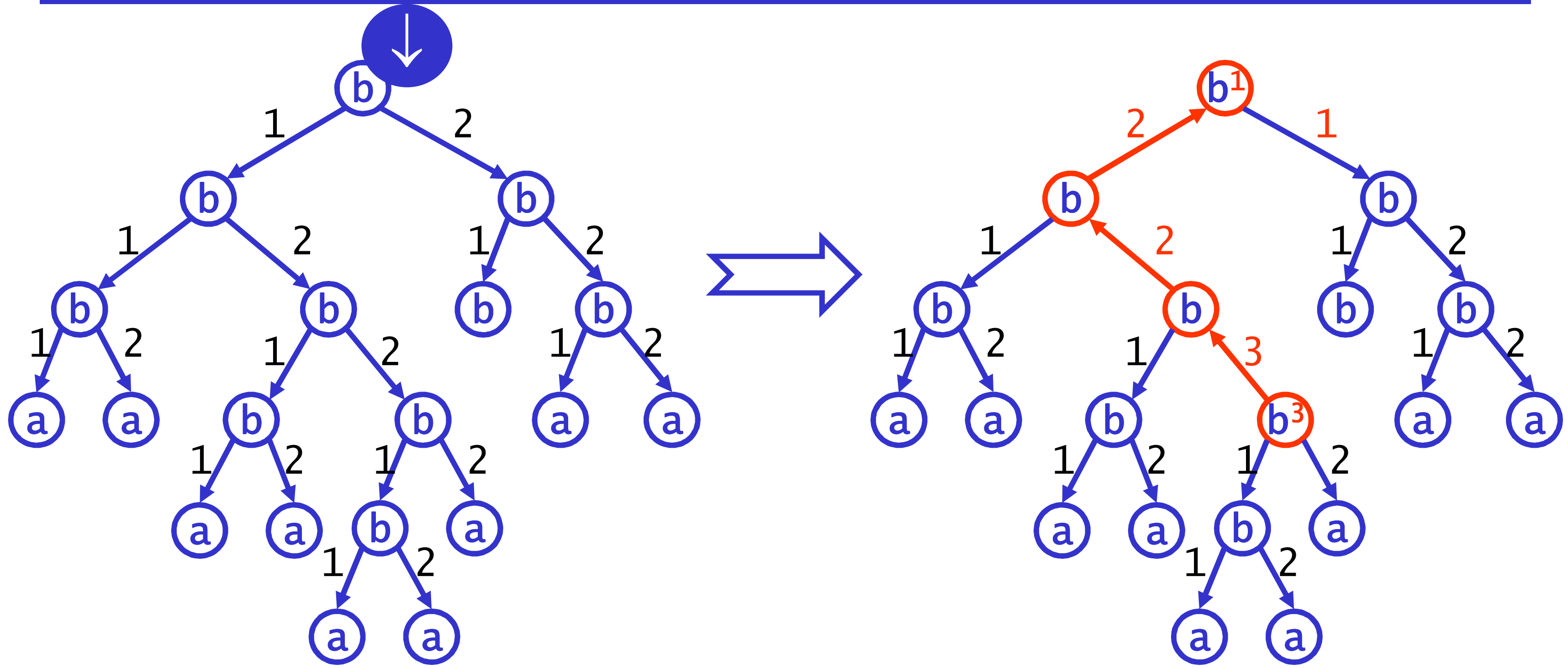


output production

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



example: moving the root



walk down

(↓,b,-,j) → (↓,$down_1$)
(↓,b,-,j) → (↓,$down_2$)

copy up

(↑,b,-,1) → b(↑$_1$,c_2)
(↑,b,-,2) → b(c_1,↑$_2$)
(↑$_i$,b,-,i) → (↑,up)

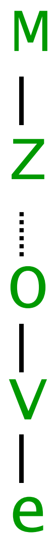
copy down

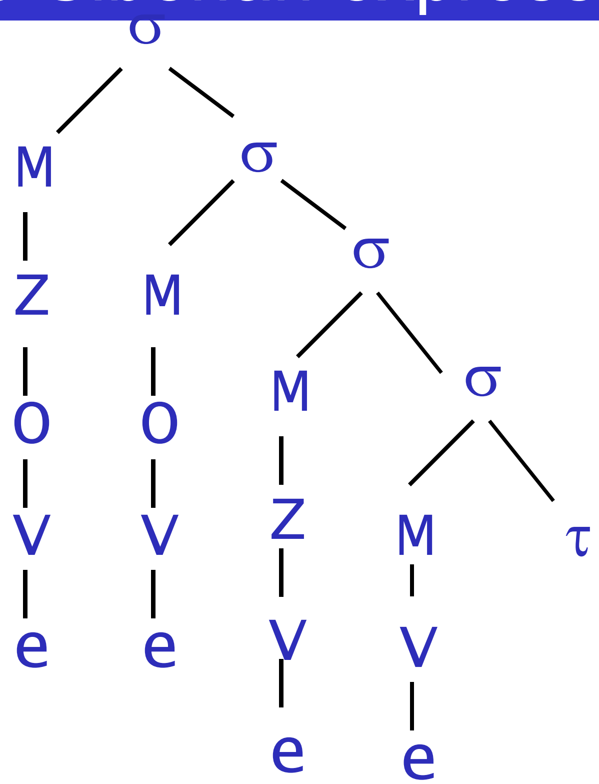
(copy,a,-,j) → a()
(copy,b,-,j) → b(c_1,c_2)
(c_i,b,-,j) → (copy,$down_i$)

j=0,1,2   i=1,2

Trans-Siberian express

Moscow
Zjeleznodorozjny
Vladimir
Bogoljoebovo
Kovrov
Dzerzjinsk
…
Spassk-Dalni
Oessoeriejsk
Vladivostok





1111…
0111
1011
0011

input: list of cities
output: list of itineraries

mark with invisible pebbles & copy
can even make ‘regular’ selections

exponential size output

notation

Pebble Tree Transducers

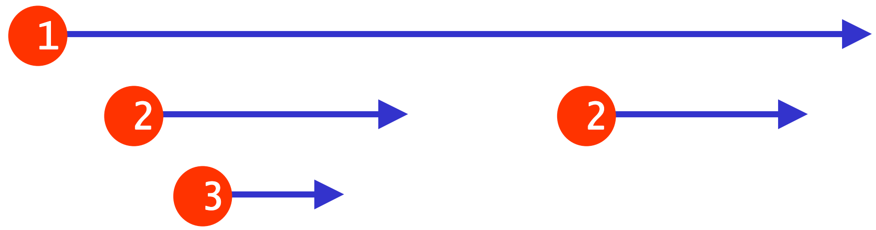
$V_k I$-PTT visible + invisible
V_k-PTT k visible pebbles Milo etal.
I-PTT invisible only
TT tree-walking (no pebbles)

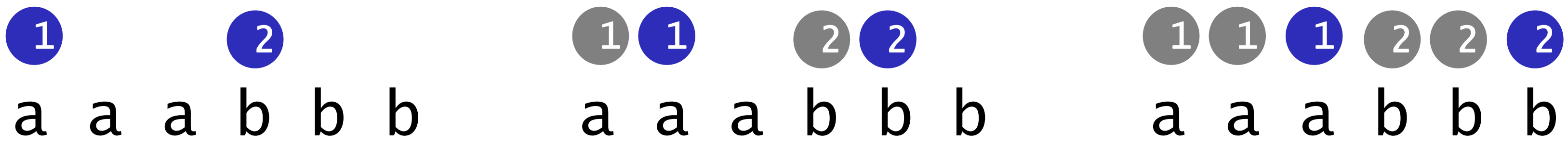
Pebble Tree Automata

$V_k I$-PTA
V_k-PTA
I-PTA

beware of the pebble (again)

avoid counting





► only topmost observable

contents

introduction: finding the right model

tree walking transducers
with invisible pebbles

technical results:
decomposition
type checking & regularity
pattern matching

► ‘classic’ pebbles

macro TT ~ topdown TT + cf tree grammar

comparison pebble TT vs. macro TT:

- V_n-PTT $\subseteq$ dTT^{n+1} $\subseteq$ $dMTT^{n+1}$
- dMTT $\subseteq$ dTT^3

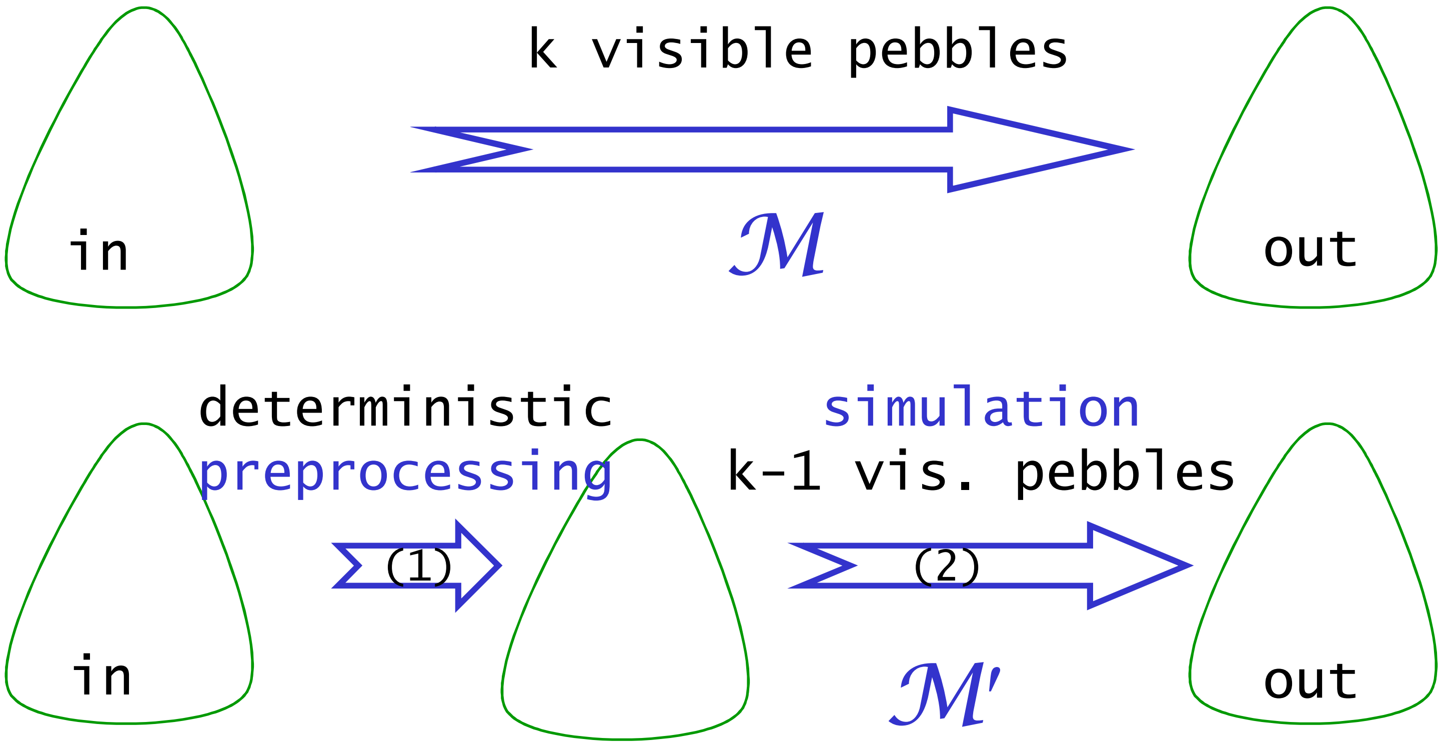
Engelfriet Maneth

► add invisible pebbles

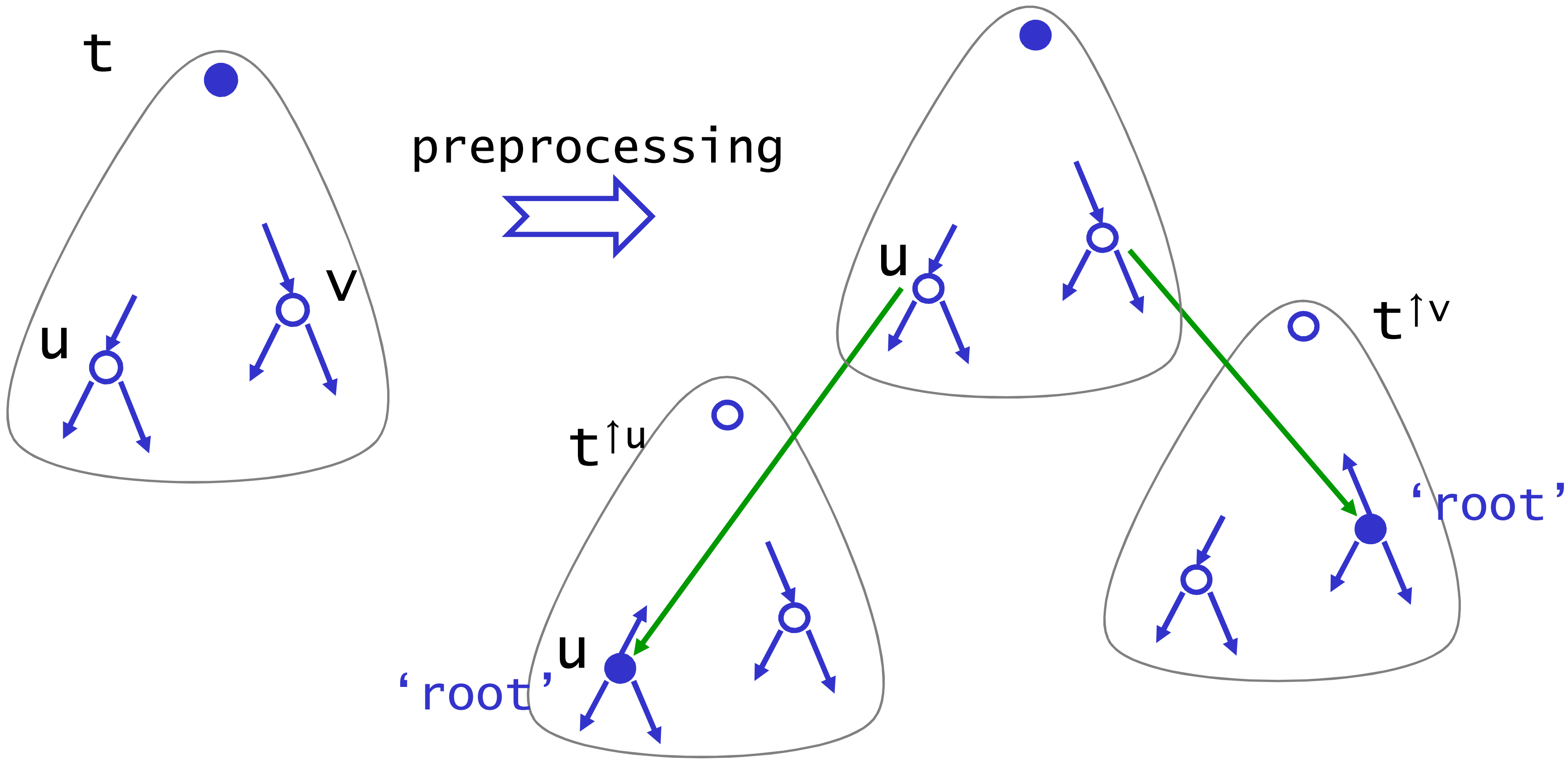
complexity per pebble

decomposition visible pebbles

$$V_k\text{I-PTT} \subseteq \text{dTT} \bullet V_{k-1}\text{I-PTT}$$

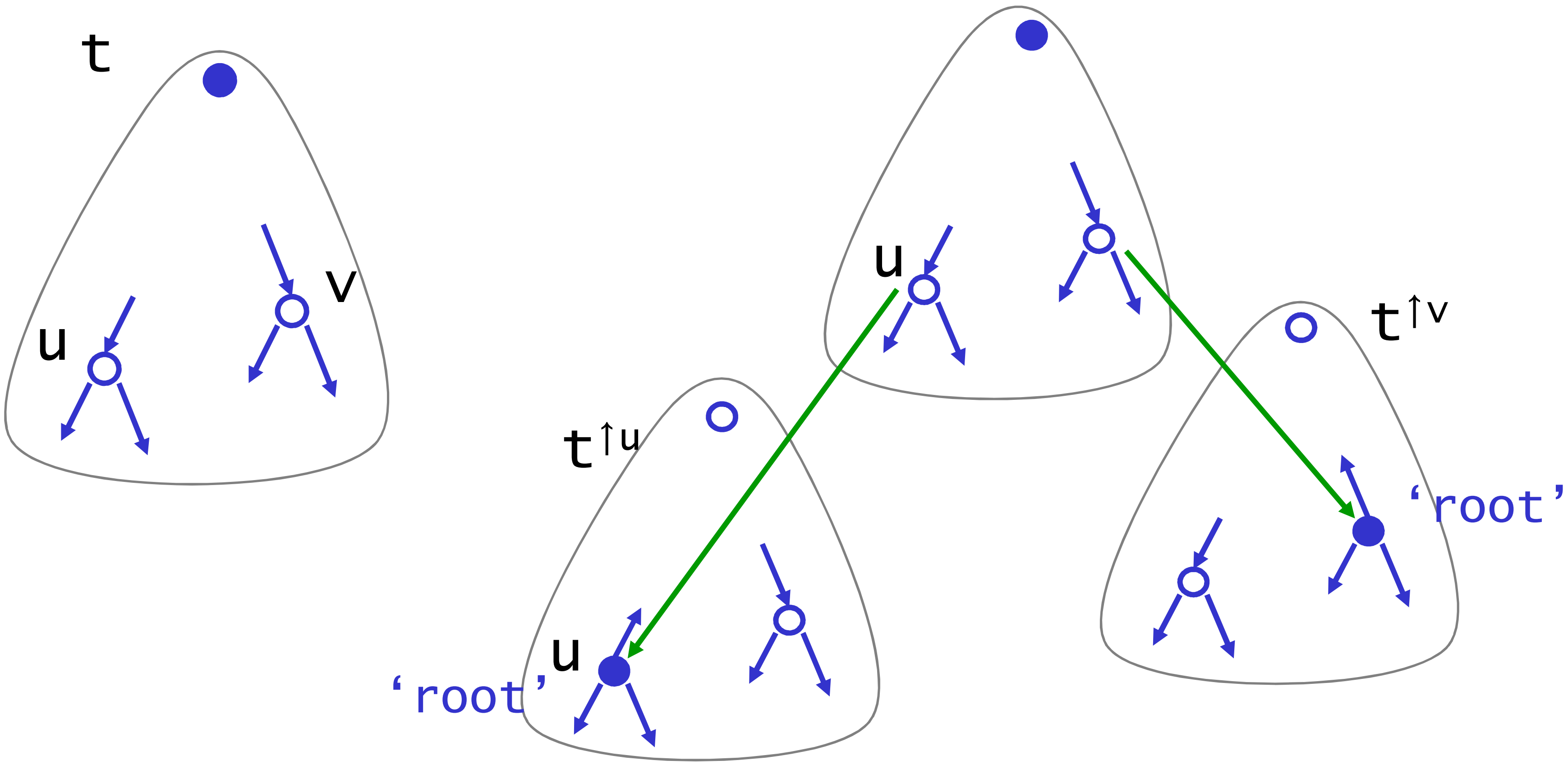


decomposition (1) *preprocessing*



copying can be done without pebbles

decomposition (2) *simulation*



$\mathcal{M}$

drop / lift
first visible pebble

$\mathcal{M}'$

move up /down
into subtree

decomposition

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

$I\text{-}dPTT \subseteq TT \bullet dTT$

(deterministic)

nondeterministic
guess number of pebbles

THEOREM

$V_k I\text{-}PTT \subseteq TT^{k+2}$

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

contents

introduction: finding the right model

tree walking transducers
with invisible pebbles

technical results:

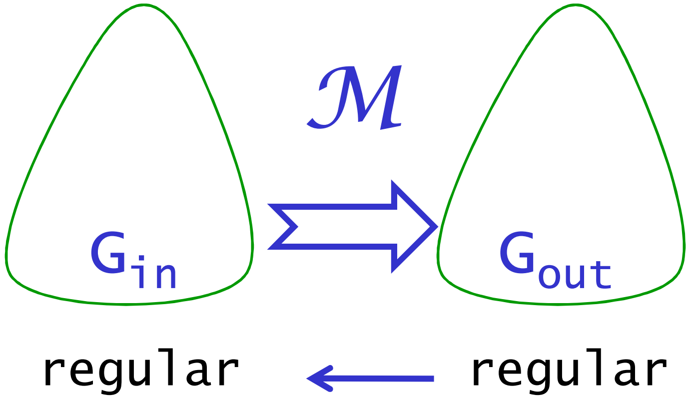
decomposition

type checking & regularity

Milo etal. pattern matching

inverse type inference

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

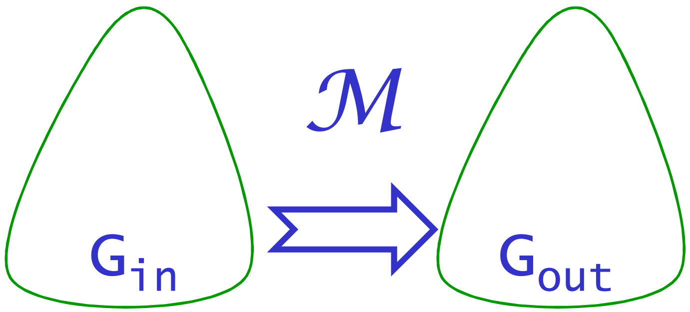


Bartha 1982
regular tree grammar G for the domain
of TT $\mathcal{M}$ can be constructed
in *exponential* time

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

type checking

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 $A \cap M^{-1}(B^c) = \emptyset$

'typechecking' 'inverse type inference'

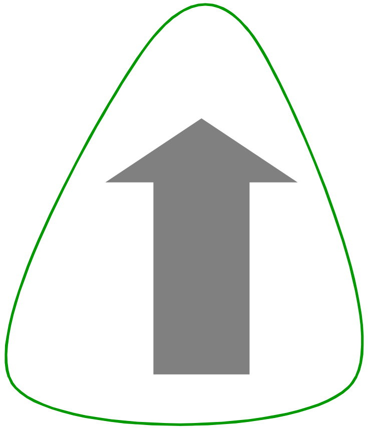
$V_k\text{-PTT} \subseteq TT^{k+1}$

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

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

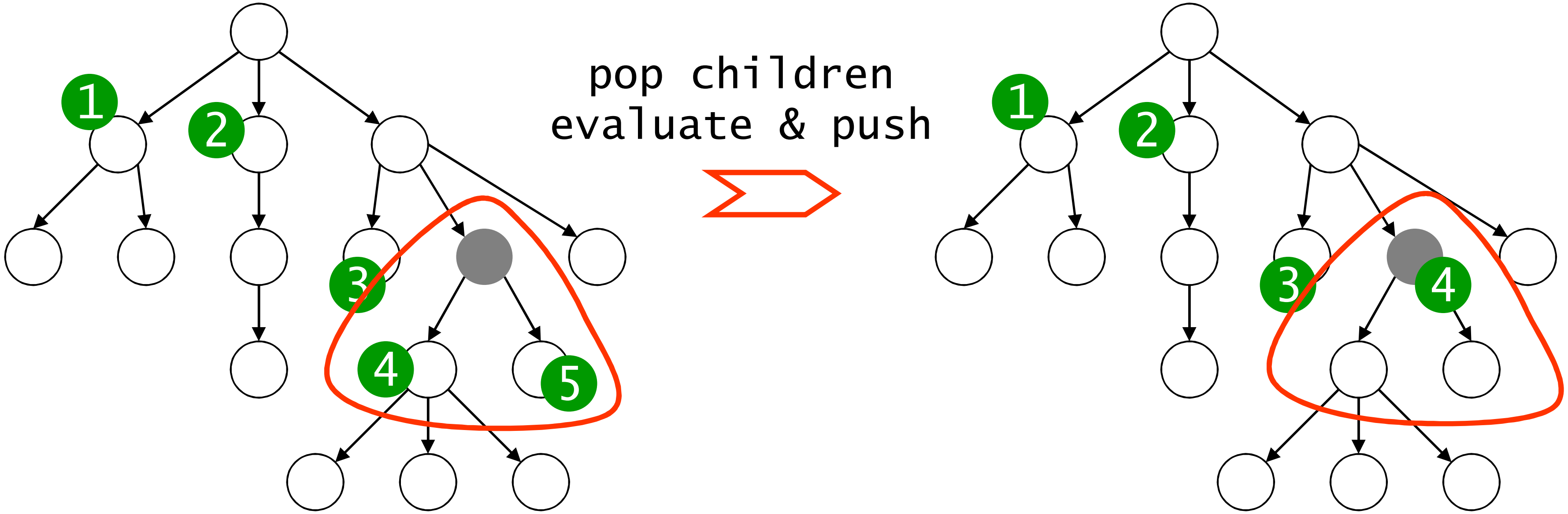
invisible pebbles are almost for free!

regular trees

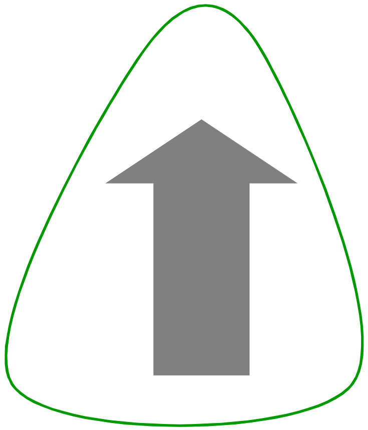


regular tree language
≡ bottom-up tree evaluation
≡ post-order evalation *with stack*

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



postorder evaluation



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

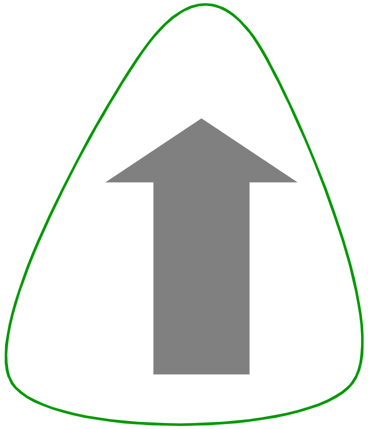
pebble++ automata recognize
regular tree languages

contents

introduction: finding the right model

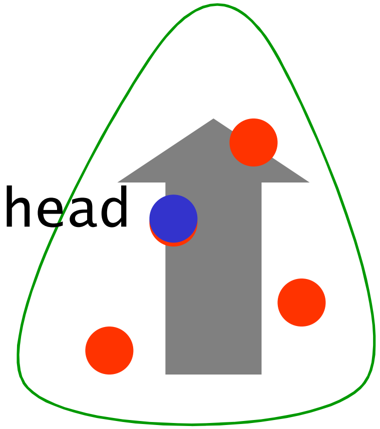
tree walking transducers
with invisible pebbles

technical results:
decomposition
type checking & regularity
pattern matching



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

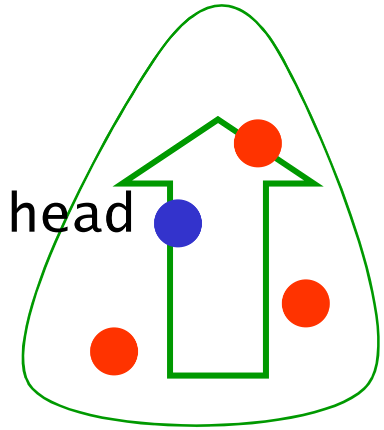
REGT ⊆ I-PTA



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

drop pebble, evaluate, return

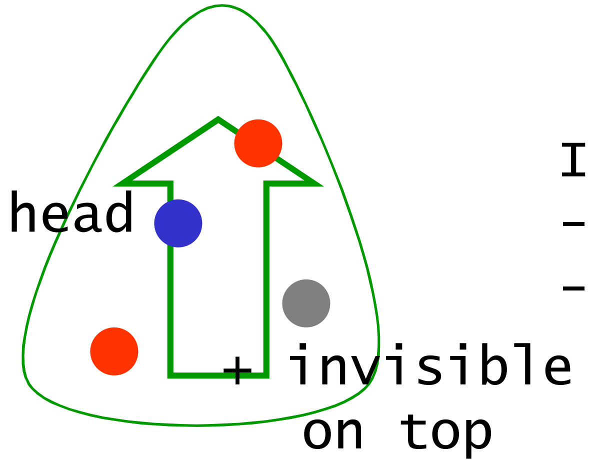
pattern matching



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

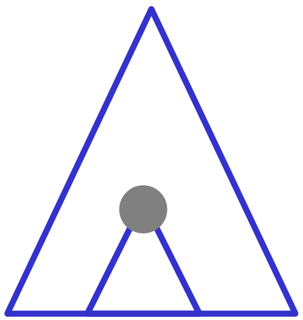
drop pebble, evaluate, return

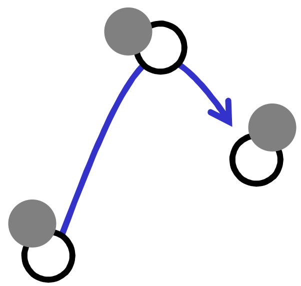
pattern matching



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

~~drop pebble, evaluate, return~~





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

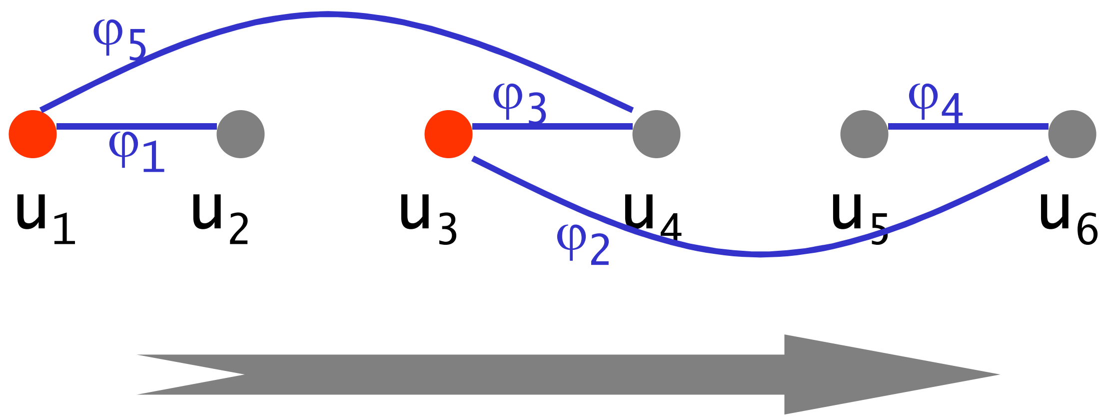
general test $\varphi(x_1,\dots,x_n)$

XQuery **for** $x_1,\dots,x_n$ **with** $\varphi_1 \wedge \dots \wedge \varphi_n$ **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

introduction: finding the right model

tree walking transducers
with invisible pebbles

technical results:
decomposition
type checking & regularity
pattern matching

conclusion

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

Trees and Invisible Pebbles

Joost Engelfriet
Hendrik Jan Hoogeboom

Universiteit Leiden

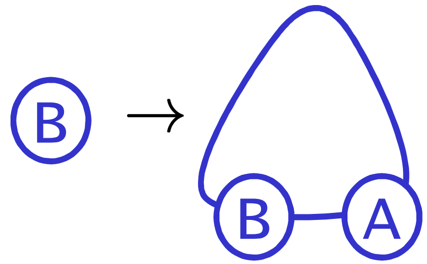
THANK YOU

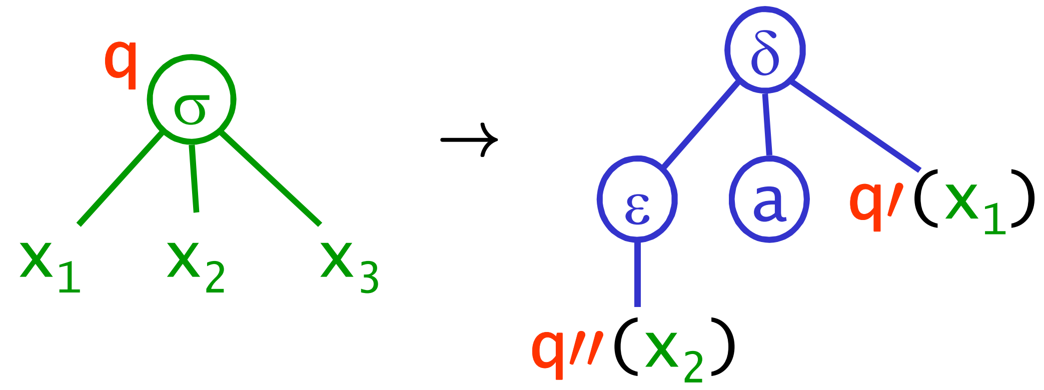
AutoMathA, Liège, June 2009

macro tree transducers

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

regular



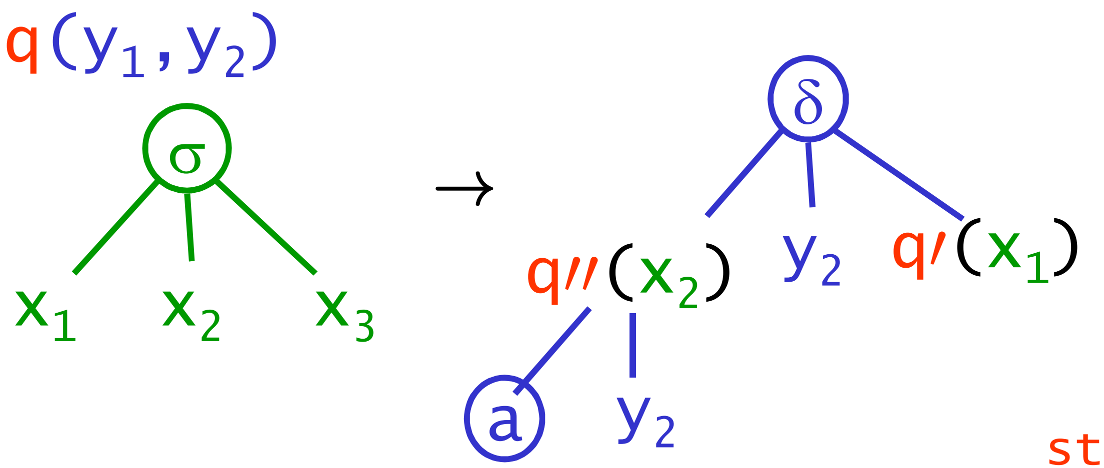


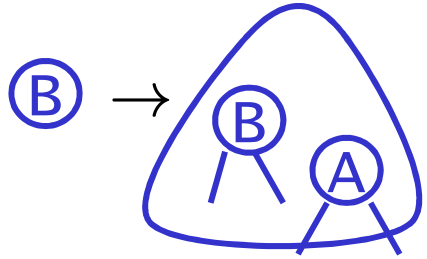
state +
subtree ´ node (input)

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

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

context-free





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

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