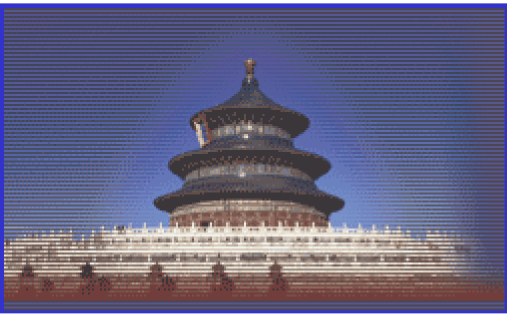
XML Transformation
by
Tree-Walking Transducers
with
Invisible Pebbles

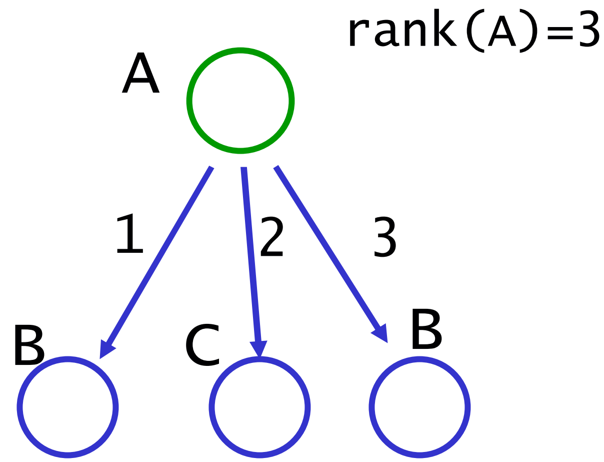
Joost Engelfriet
Hendrik Jan Hoogeboom
Bart Samwel
(Leiden University, NL)



PODS Beijing June 2007

tree model

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



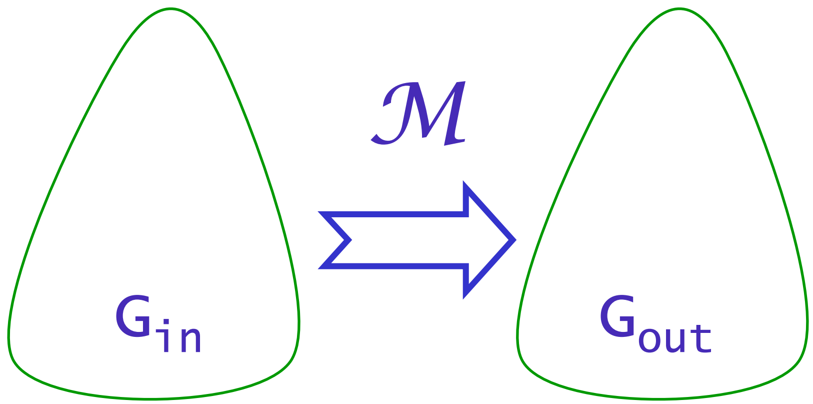
ranked trees
node labels with rank

unbounded number of children
(forests) are to be coded

background

typechecking

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



Milo Suciu Vianu PODS2000

type checking for XML transformers is decidable

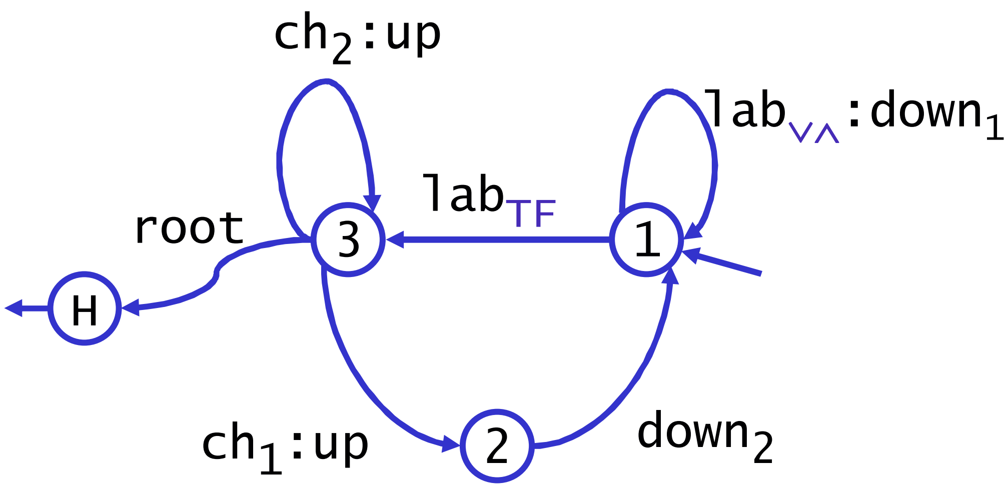
transformers with 'visible' pebbles:
finite number of coloured markers on tree

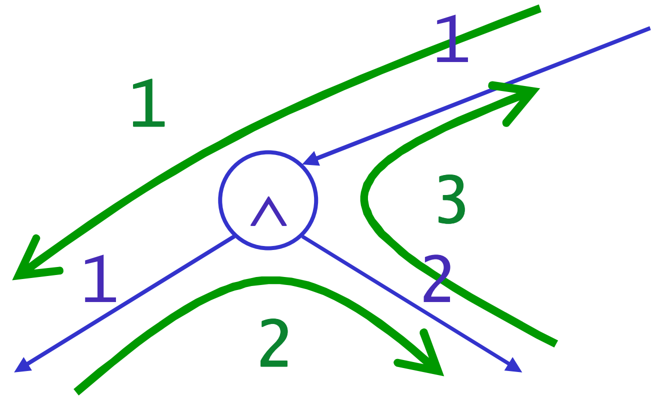
contents

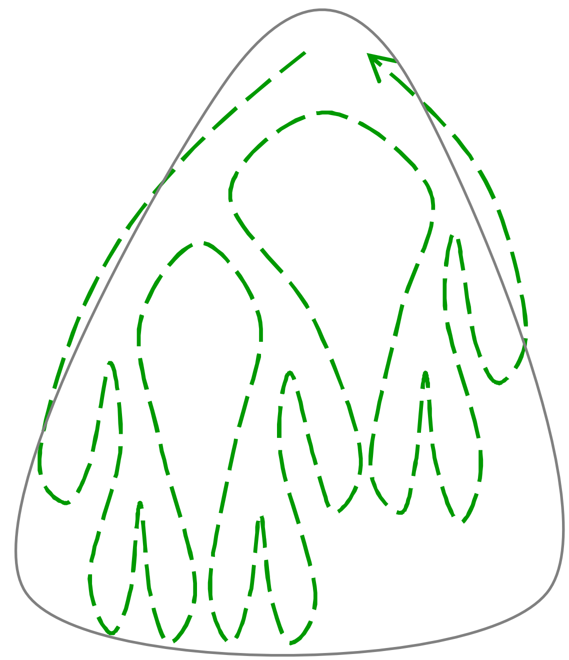
1. automata with pebbles
2. decomposition
3. typechecking
4. regular trees
5. document navigation
6. pattern matching
7. conclusion

without pebbles

example: preorder tree traversal





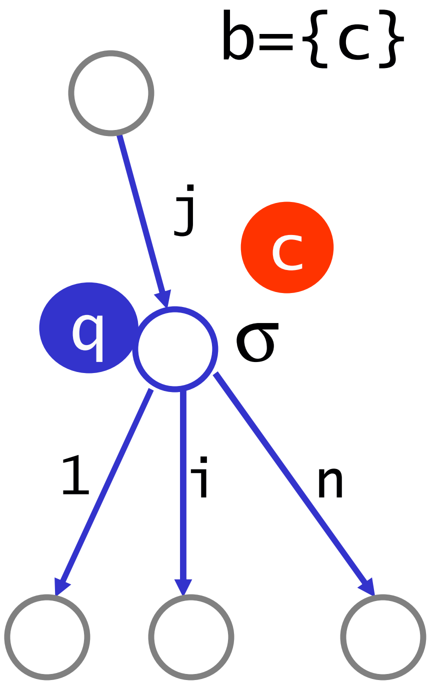


walk along edges, moves based on

- state
- node label
- child number
  (= incoming edge)

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)

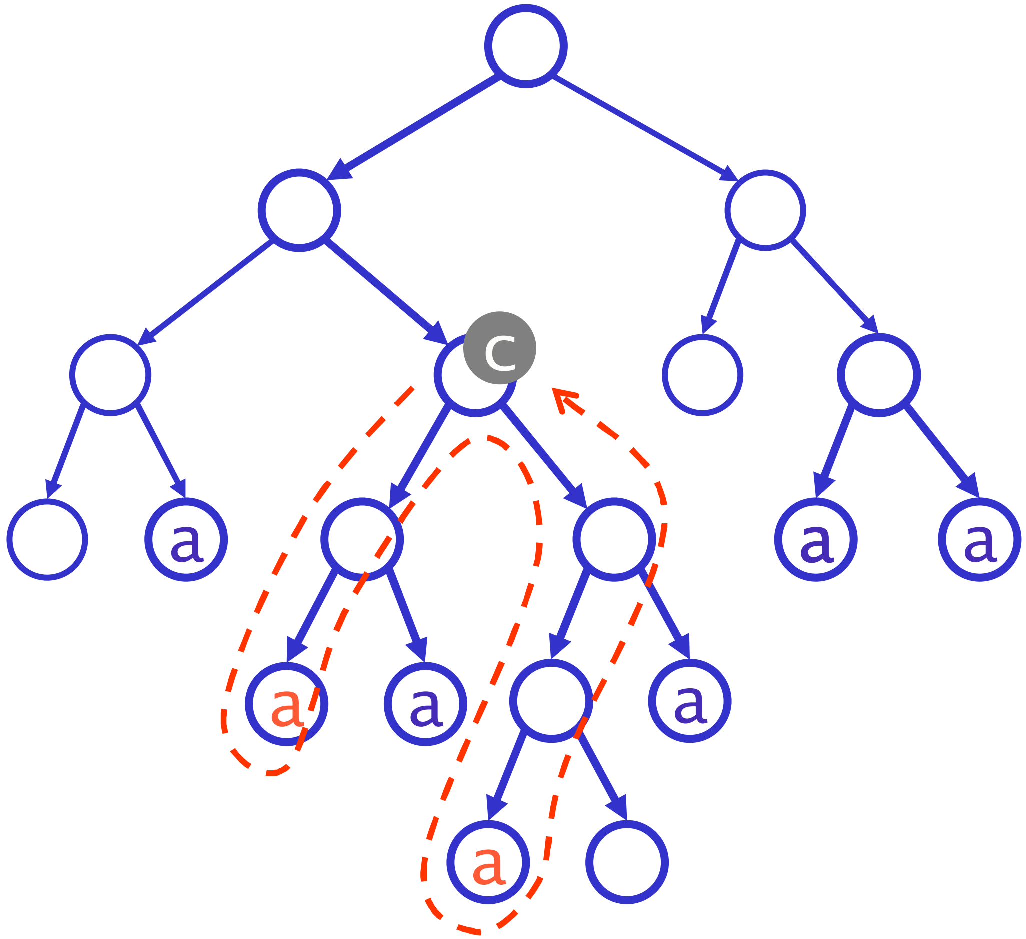
(q',drop_c)

(q',lift_c)

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

example: inspecting a subtree

using a pebble



tree-walking pebble automata

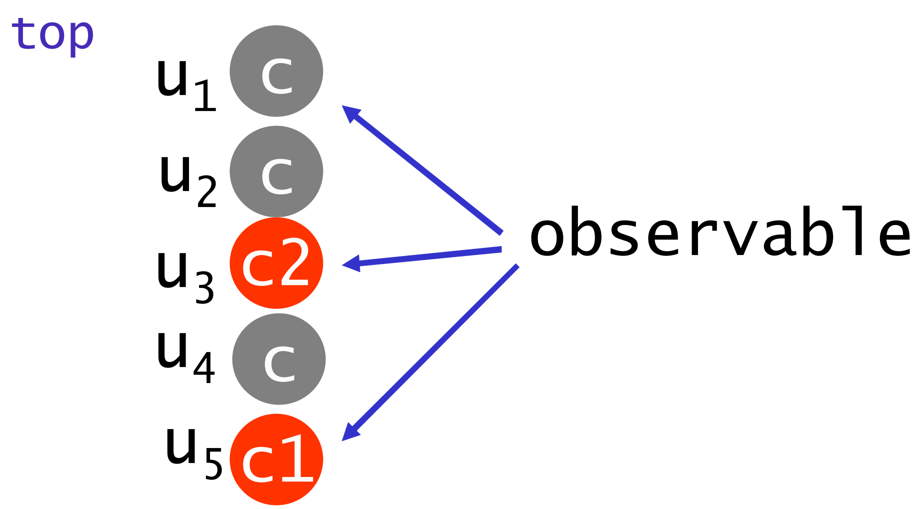
c with visible pebbles
'colours' used once
always observable

☹ do not recognize all regular tree languages
≡ MSO properties

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

☺ recognize regular
& decidable type checking
& better complexity

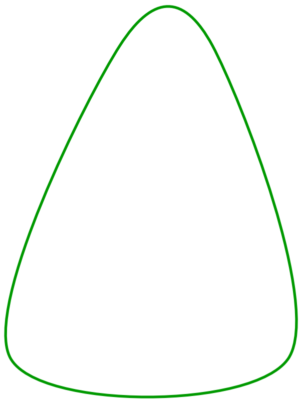
stack behaviour of pebbles!
(avoid 'counting')



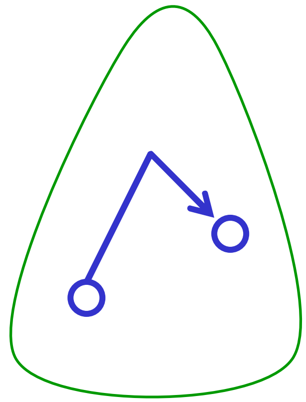
$(q,\sigma,b,j) \to (q',\text{stay})$

b contains
-all visible pebbles
-invisible when topmost

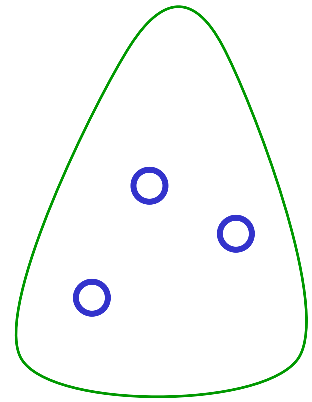
automaton defines …



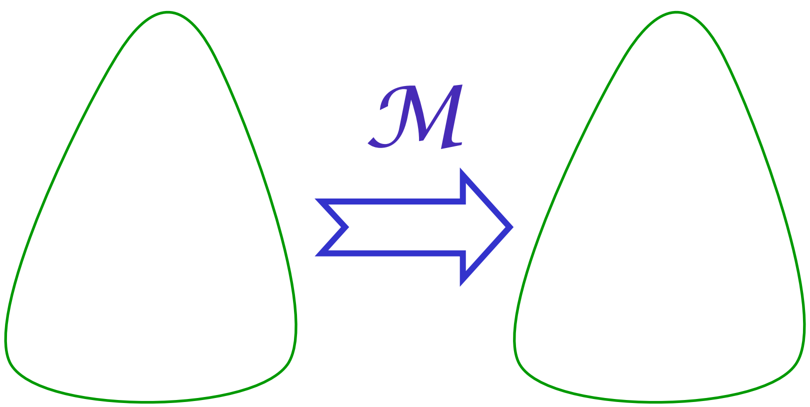
validation



navigation



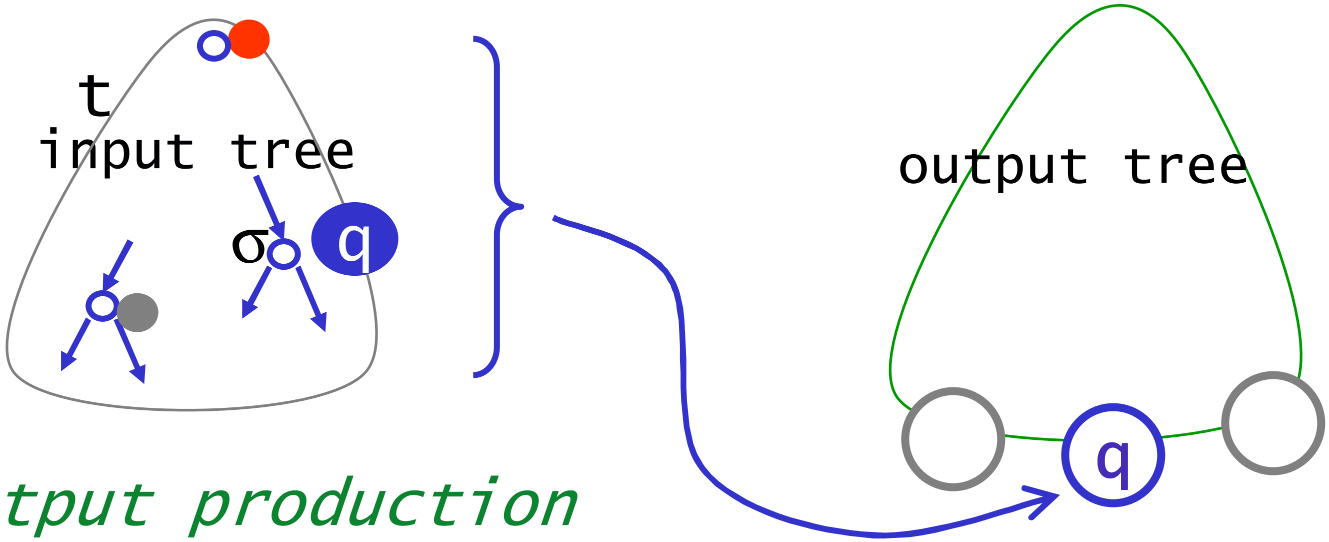
pattern matching



transformation

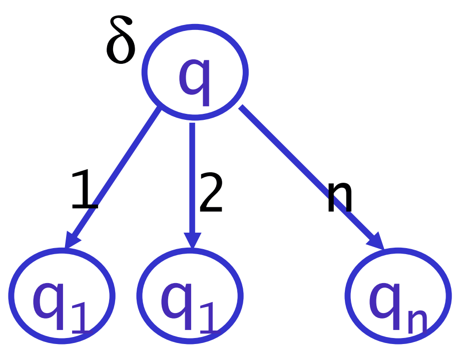
tree-walking pebble tree *transducers*

recursively generate output



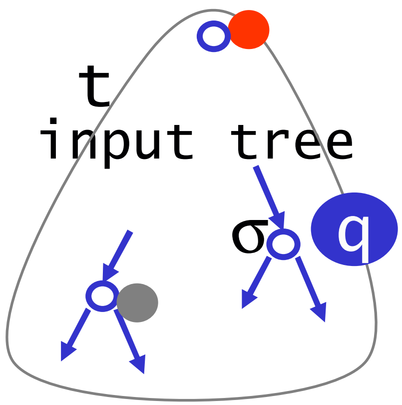
output production

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



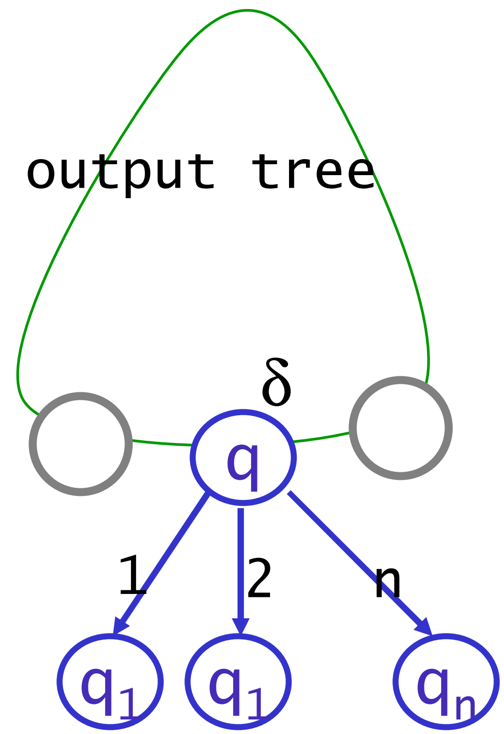
tree-walking pebble tree transducers

recursively generate output



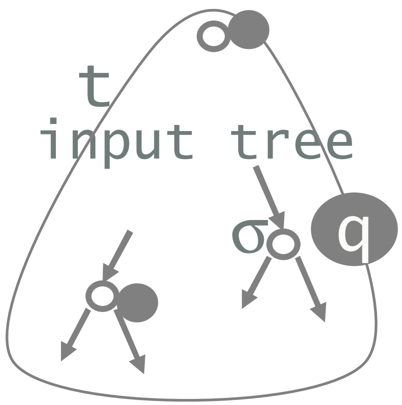
output production

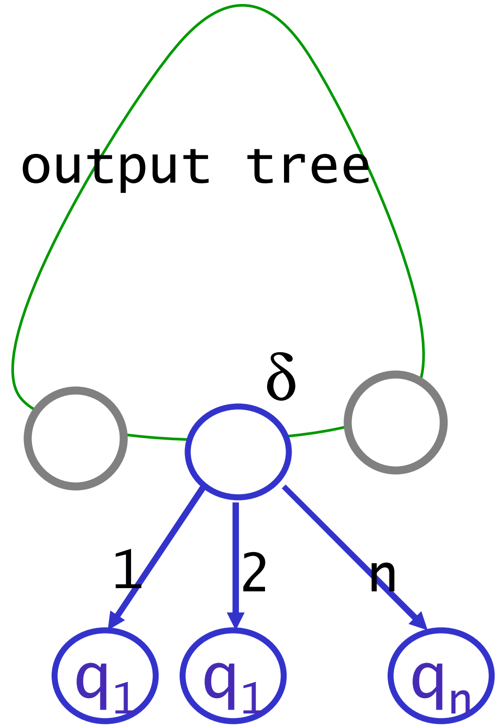
$$(q,\sigma,b,j) \rightarrow \delta(q_1,q_2 \ldots q_n)$$



tree-walking pebble tree transducers

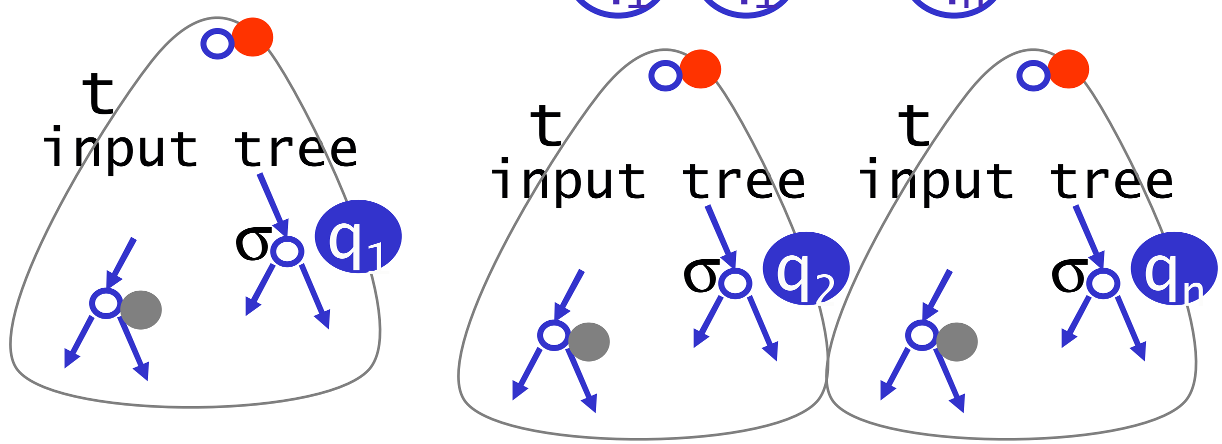
recursively generate output



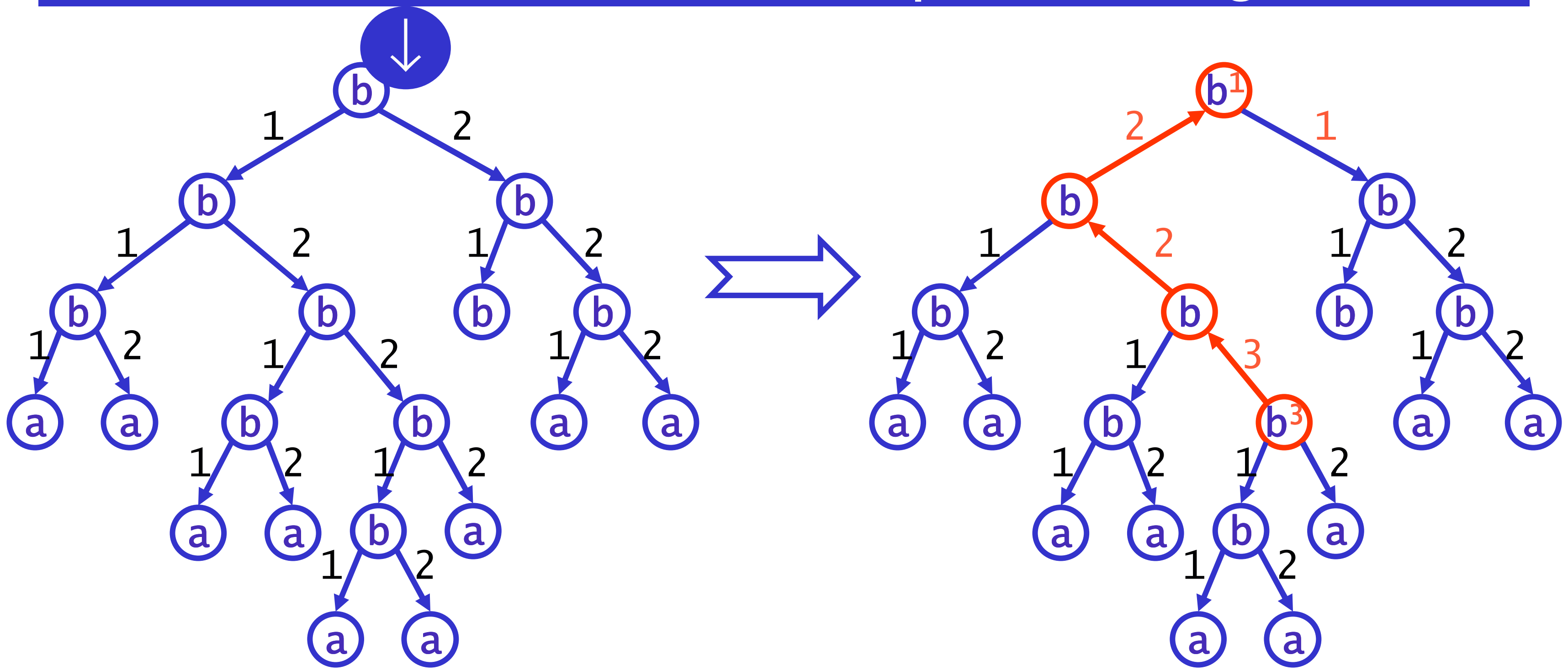


output production

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



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

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

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

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

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

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

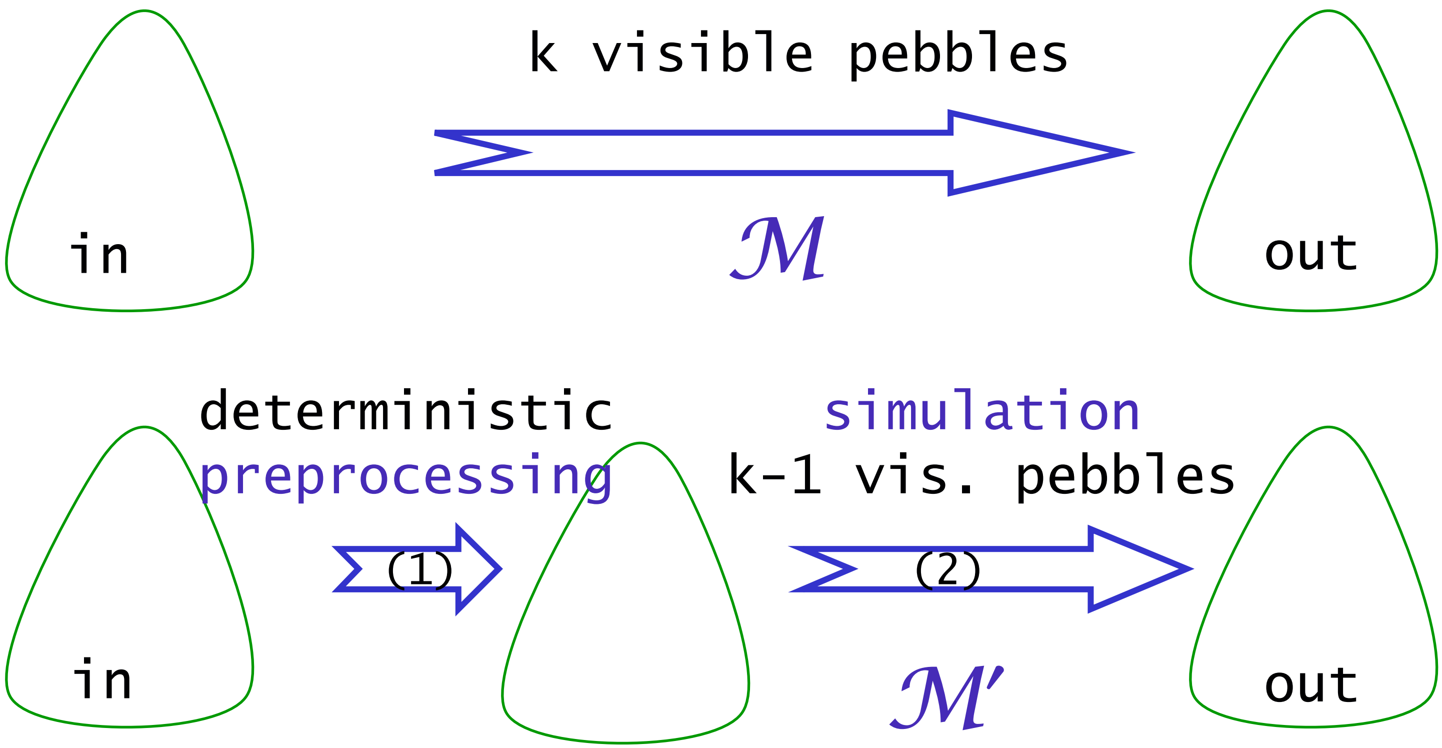
V_kI-PTA
V_k-PTA
I-PTA

contents

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

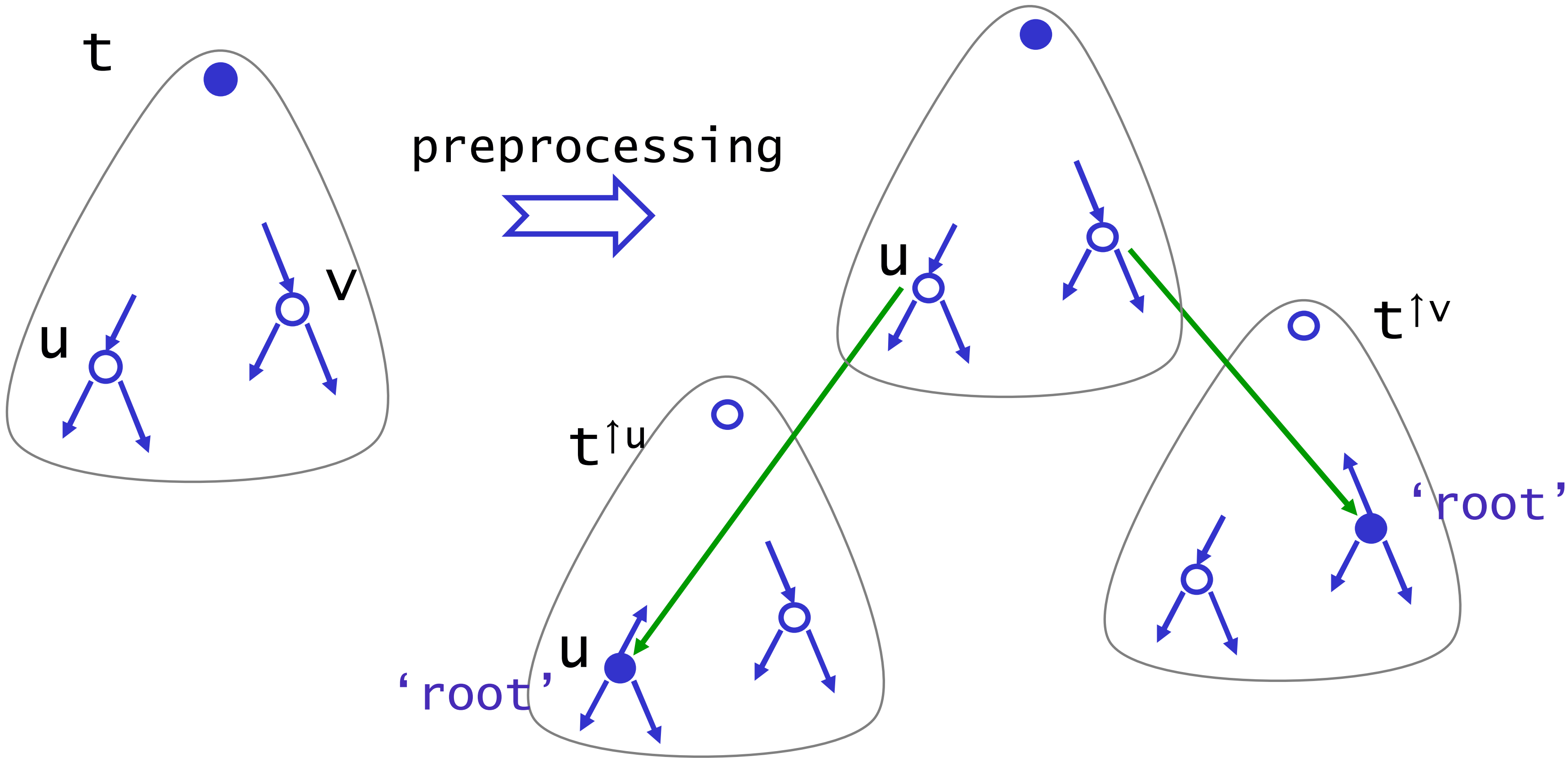
decomposition visible pebbles

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



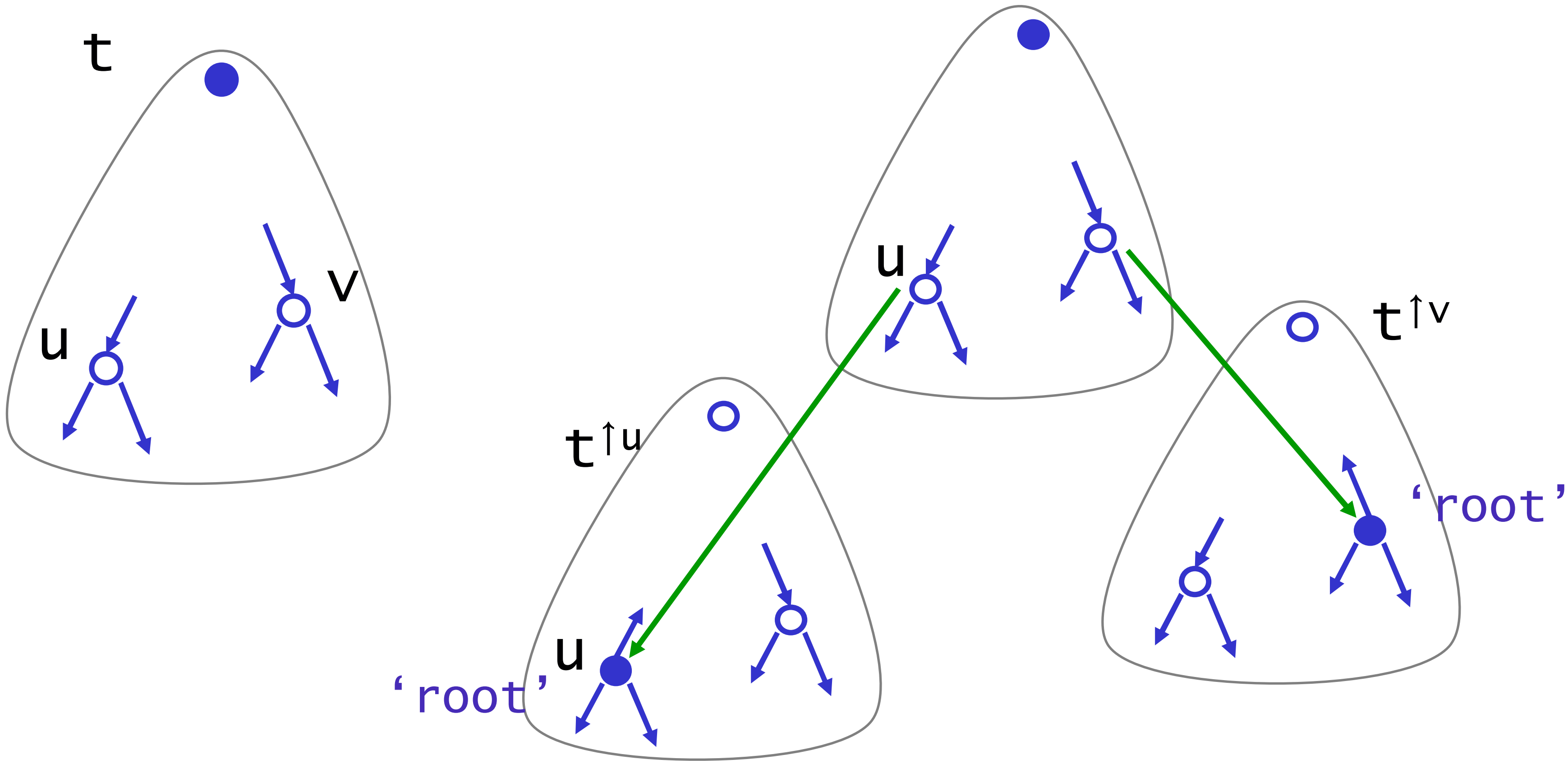
iterate $$V_k\text{I-dPTT} \subseteq \text{dTT}^k \circ \text{I-dPTT}$$

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

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

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

(deterministic)

THEOREM

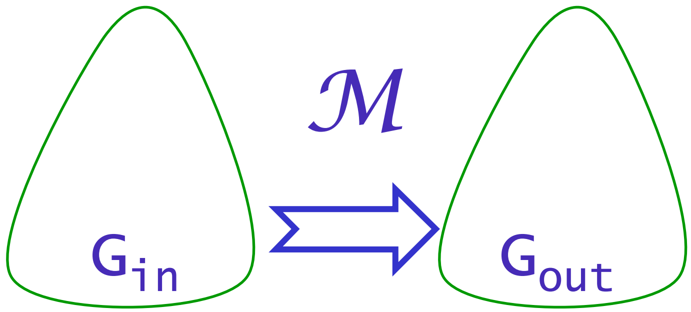
$V_k\text{-}PTT \subseteq TT^{k+1}$

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

contents

inverse type inference

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



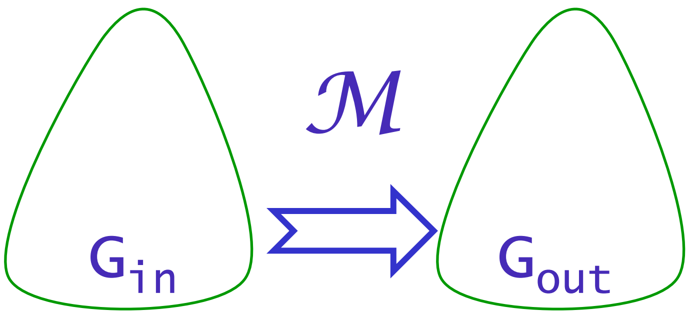
Bartha 1982

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

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'

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

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

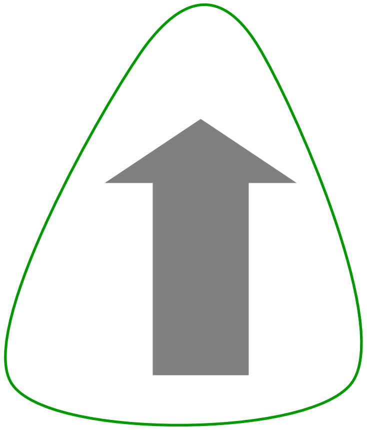
we can typecheck
⇒ TT^k in $(k+1)$-fold exponential time
⇒ V_k-PTT in $(k+2)$-fold exponential time
⇒ V_kI-PTT in $(k+3)$-fold exponential time

invisible pebbles are almost for free!

contents

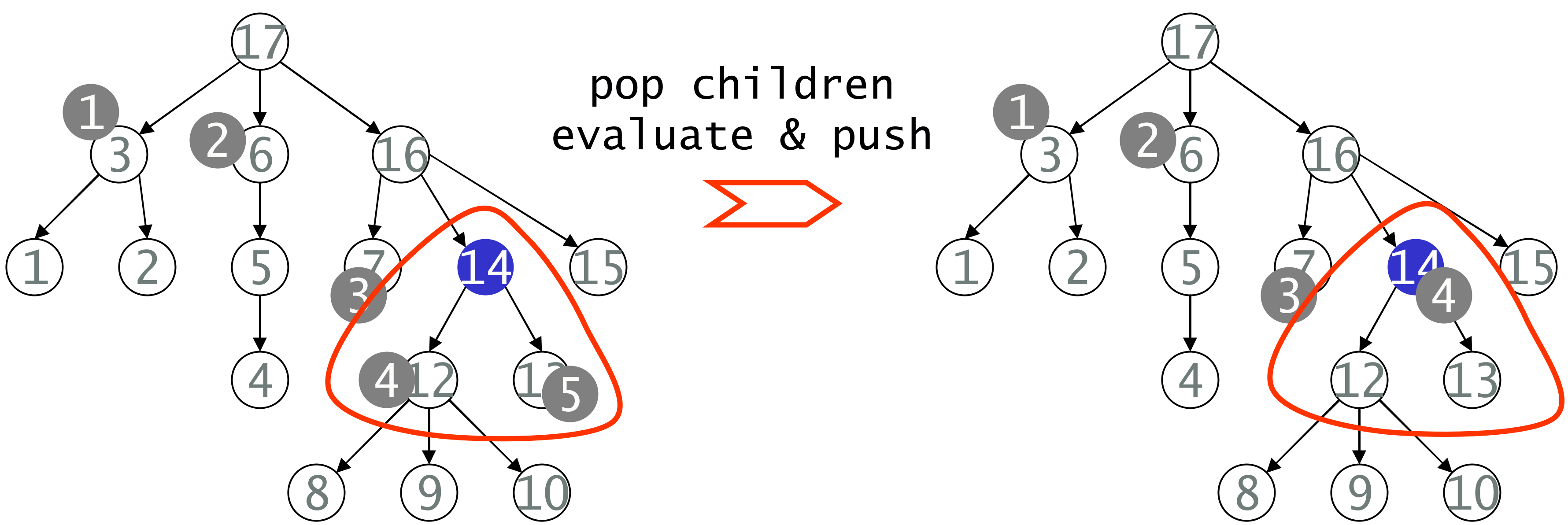
1. automata with pebbles
2. decomposition
3. typechecking
4. regular trees
5. document navigation
6. pattern matching
7. conclusion

regular trees

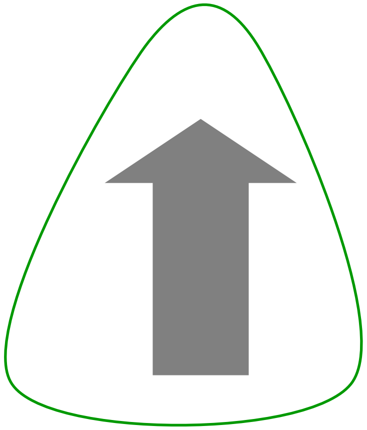


regular tree language
≡ bottom-up tree evaluation
≡ post-order evalation with stack

REGT $\subseteq$ I-PTA



regular trees



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

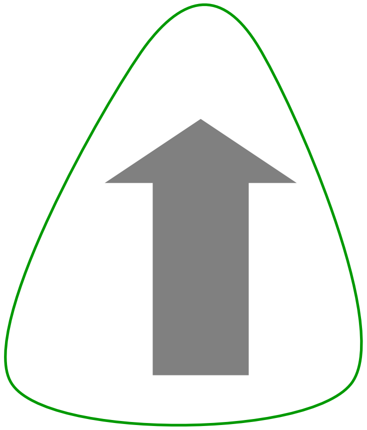
REGT ⊆ I-PTA

REGT ⊈ V_k-PTA Bojańczyk etal.

V_kI-PTT ⊆ TT^{k+2}

V_kI-PTA ⊆ REGT

regular trees



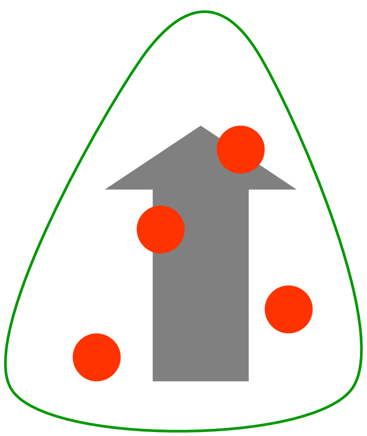
regular tree language
≡ bottom-up tree evaluation
≡ post-order evalation with stack

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

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

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

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



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

contents

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

document navigation

- *Pebble Cat*

caterpillar expressions + pebbles ↔ I-PTA programs

node expressions

φ_0 ::= lab_σ | isleaf | isroot |
isfirst | islast | $pebble_c$

tests

φ ::= φ_0 | $\neg\varphi$ | $\varphi\wedge\varphi$

path expressions

α_0 ::= child | parent | right | left |
$drop_c$ | lift

moves
pebbles

α ::= α_0 | $?\varphi$ | $\alpha\cup\alpha$ | α/α | α^*

Kleene

semantics

$$[?\varphi]_f = \{ ((u,\pi),(u,\pi)) \mid (u,\pi)\in[\varphi]_f \}$$

head pebble stack

document navigation

- *Pebble Cat*

  caterpillar expressions + pebbles

  ↔ I-PTA programs   MSO complete ☺

- *PCat*   ↔ V-PTA programs

  Goris,Marx LICS'05

- *Pebble XPath*

  extends Regular XPath with invisible pebbles

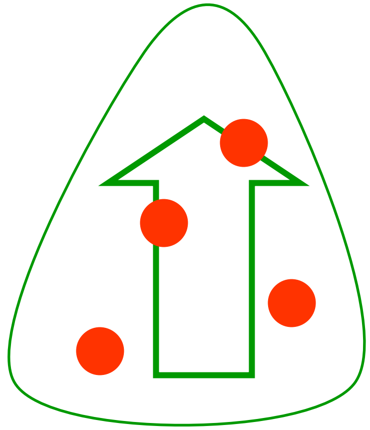
$$\varphi ::= \varphi_0 \mid \langle\alpha\rangle \mid \neg\varphi \mid \varphi\wedge\varphi$$

$$[\langle\alpha\rangle]_f = \{ (u,\pi) \mid \exists(v,\pi'):((u,\pi),(v,\pi'))\in[\alpha]_f \}$$

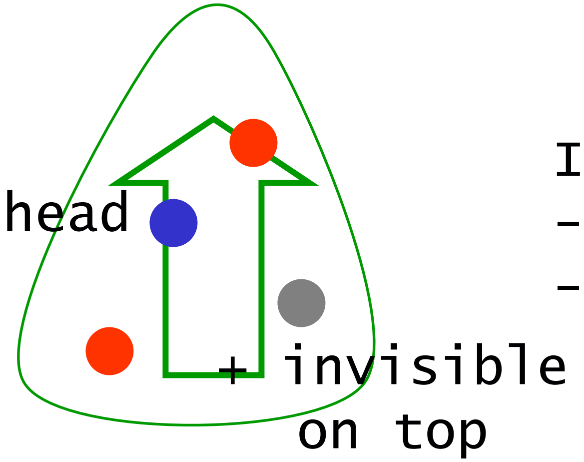
⇒ *look-ahead tests*

contents

pattern matching



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



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

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

pattern matching

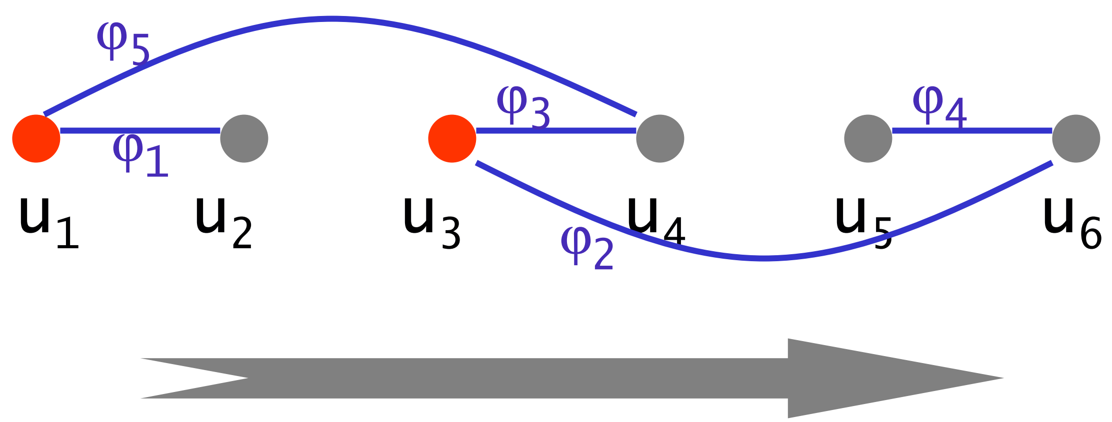
general test $\varphi(x_1,\dots,x_n)$

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

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

‘tossing Pebbles’

Joost Engelfriet
Hendrik Jan Hoogeboom
Bart Samwel
Leiden NL

thank you …