Trees and Invisible Pebbles

Joost Engelfriet
Hendrik Jan Hoogeboom

Universiteit Leiden

AutoMathA, Liège, June 2009
document transformation

Gauwin Niehren Tison: Earliest query answering ...

\[ \begin{align*}
\text{select nodes} & \quad & \text{& reformat}
\end{align*} \]
ranked trees $\sim$ terms

$$\Sigma, \text{rank}$$

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

$$\Sigma_k \quad \text{rank } k$$

$\Sigma_0 = \{a, b\}$

$\Sigma_2 = \{\delta\}$

$\Sigma_3 = \{\sigma\}$

$T_\Sigma$ trees over $\Sigma$

child number

child-sibling coding

$f \in \Sigma_k$

$f(x_1x_2...x_k)$

$fx_1x_2...x_k$
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
finding a model

transformation

select nodes:

validation

navigation

pattern matching

MSO
bottom-up tree automaton

rules:

bottom-up evaluation
tree walking automaton

walk along edges

cf. two-way finite state automaton
example: pre order tree traversal

walk along edges, moves based on
- state
- node label
- child number

(= incoming edge)
tree automata

bottom-up evaluation

REG ≡ MSO

need to:

- verify input tree
- select nodes (both based on a MSO property)
“tree walking automata easily lose their way”
'branching structure' of even length

Bojańczyk & Colcombet

TWA $\subset \subset \subset$ REG

branching?

not by TWA
but FO
pebbles to mark nodes

Bojańczyk & Colcombet

TWA $\subseteq$ 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’

PTWA ⊆ REG

‘regular’ extension
beware of the pebble

avoid counting

► nest!

► bounded number!
power of tree walking automata

\[ \text{TWA} \subseteq \text{REG} \]

Bojańczyk & Colcombet STOC’05

\[ \text{TWA} \subset \text{REG} \]

\[ \text{PTWA} \subseteq \text{REG} \]

Engelfriet & H ‘99

\[ \text{PTWA} \subset \text{REG} \]

Bojańczyk, Samuelides, Schwentick & Segoufin ICALP’06

“tree walking automata easily lose their way”
(even with the help of pebbles)
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

Milo Suciu Vianu PODS2000
type checking for XML transformers is decidable

Engelfriet & H & Samwel PODS2007

Slutzki 1985
‘two-way backtracking pushdown tree automata’

TWTT
+ pebbles
+ ‘invisible’ pebbles
tree-walking automata

local configuration
- q state
- \( \sigma \) node label
- j child number
- \( j=0 \) root

B pebble colours
- \( B \subseteq \mathbb{C} \)

instructions
- \((q, \sigma, B, j) \rightarrow (q', \text{stay})\)
- \((q', \text{up})\)
- \((q', \text{down}_i)\)
- \((q', \text{drop}_c)\)
- \((q', \text{lift}_c)\)

- finite set \( \mathbb{C} \) of pebbles ‘colours’
- nested lifetimes: distributed stack
  - only topmost can be lifted
- classical: all observable, finite
- set: keep order in finite state
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

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

- do not recognize all regular tree languages
- ≡ MSO properties

- 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**

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

Each $q$ works on separate copy input tree

- **tdtt** - $q_i$ point to children ($\downarrow$)
- **twtt** - $q_i$ point to same node

$q$'s may move up $\uparrow$ and down $\downarrow$ in between
tree-walking pebble tree transducers

recursively generate output

output production

\[(q, \sigma, B, j) \rightarrow \delta(q_1, q_2 \ldots q_n)\]
without pebbles

example: moving the root

walk down

\[(\downarrow, b, -, j) \rightarrow (\downarrow, \text{down}_1)\]
\[(\downarrow, b, -, j) \rightarrow (\downarrow, \text{down}_2)\]

copy up

\[(\uparrow, b, -, 1) \rightarrow b(\uparrow_1, c_2)\]
\[(\uparrow, b, -, 2) \rightarrow b(c_1, \uparrow_2)\]
\[(\uparrow_i, b, -, i) \rightarrow (\uparrow, \text{up})\]

copy down

\[(\text{copy}, a, -, j) \rightarrow a()\]
\[(\text{copy}, b, -, j) \rightarrow b(c_1, c_2)\]
\[(c_i, b, -, j) \rightarrow (\text{copy}, \text{down}_i)\]

\[j=0,1,2 \quad i=1,2\]
Trans-Siberian express

with invisible pebbles

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

input: list of cities
output: list of itineraries

1111...
0111
1011
0011

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

exponential size output
Pebble Tree Transducers

\[ V_kI\text{-}PTT \quad \text{visible + invisible} \]
\[ V_k\text{-}PTT \quad k \text{ visible pebbles} \quad \text{Milo et al.} \]
\[ I\text{-}PTT \quad \text{invisible only} \]
\[ TT \quad \text{tree-walking (no pebbles)} \]

Pebble Tree Automata

\[ V_kI\text{-}PTA \]
\[ V_k\text{-}PTA \]
\[ I\text{-}PTA \]
beware of the pebble (again)

avoid counting

only topmost observable
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:
- $V_n^{PTT} \subseteq dTT^{n+1} \subseteq dMTT^{n+1}$
- $dMTT \subseteq dTT^3$

add invisible pebbles

complexity per pebble
\[ V_k^{I-PTT} \subseteq dTT \cdot V_{k-1}^{I-PTT} \]
decomposition (1) preprocessing

preprocessing

copying can be done without pebbles
decomposition (2) simulation

\[ M \]

Drop / lift
First visible pebble

\[ M' \]

Move up / down
Into subtree
\[ V_k I-dPTT \subseteq dTT \cdot V_{k-1} I-dPTT \]
\[ I-dPTT \subseteq TT \cdot dTT \]

(deterministic)

nondeterministic

guess number of pebbles

THEOREM

\[ V_k I-PTT \subseteq TT^{k+2} \]
\[ V_k -PTT \subseteq TT^{k+1} \]
introduction: finding the right model

tree walking transducers
with invisible pebbles

technical results:
  decomposition
  type checking & regularity
  pattern matching

Milo et al.
inverse type inference

given $\mathcal{M}$ 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 $\mathcal{M}$ can be constructed
in exponential time

inverse type inference is solvable
$\Rightarrow$ for $\mathcal{M}$ in exponential time
$\Rightarrow$ for $\mathcal{M}^k$ in $k$-fold exponential time
We can typecheck
\[ \Rightarrow \mathcal{TT}^k \text{ in } (k+1)\text{-fold exponential time} \]
\[ \Rightarrow V_{k\text{-PTT}} \text{ in } (k+2)\text{-fold exponential time} \]
\[ \Rightarrow V_{k\text{I}-\text{PTT}} \text{ in } (k+3)\text{-fold exponential time} \]

Invisible pebbles are almost for free!
regular trees

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

REGT ⊆ I-PTA

postorder evaluation
regular trees

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

\[ \text{REGT} \subseteq \text{I-PTA} \]
\[ \text{REGT} \not\subseteq \text{V}_k\text{-PTA} \quad \text{Bojańczyk et al.} \]

\[ \text{V}_k\text{I-PTT} \subseteq \text{TT}^{k+2} \]

\[ \text{V}_k\text{I-PTA} \subseteq \text{REGT} \]

pebble++ automata recognize regular tree languages
introduction: finding the right model
  tree walking transducers with invisible pebbles
technical results:
  decomposition
  type checking & regularity
  pattern matching
regular trees

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

REGT ⊆ I-PTA

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

\textit{drop pebble, evaluate, return}
I-PTA can
- evaluate *marked* trees
- test their visible configuration

drop pebble, evaluate, return
VI-PTA can test $\varphi(x_1, \ldots, x_n)$ with $n-2$ visible pebbles (using head)
pattern matching

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

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

example

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

only 2 visible pebbles!
introduction: finding the right model

tree walking transducers
with invisible pebbles

technical results:
decomposition
type checking & regularity
pattern matching

conclusion
• extends known models

\[ V\text{-PTT} \quad \text{Milo, Suciu, Vianu} \]
\[ I\text{-PTT} = TL \quad \text{Maneth et al. PODS'05} \]

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 \((\text{input})\) &
context-free tree grammars \((\text{output})\)

\[
q(x_1 x_2 x_3) \rightarrow t \in T_\Delta [Q(X_k)] \quad \text{rank}(\sigma) = k
\]

regular

\[
q(\sigma(x_1 \ldots x_k)) \rightarrow t \in T_\Delta [Q(X_k)]
\]

\[
\text{state + subtree \& node (input)}
\]
macro tree transducers

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

context-free

\[ q(y_1, y_2) \]

\[ \sigma \]

\[ x_1 \quad x_2 \quad x_3 \]

\[ \rightarrow \]

\[ \delta \]

\[ \beta \rightarrow \]

\[ B \]

\[ A \]

\[ \delta \]

\[ q/(x_1) \]

\[ q//x_2 \]

\[ y_2 \]

\[ a \]

\[ \text{state + subtree \ node (input) + parameters (output)} \]

\[ q(\sigma(x_1 \ldots x_k), y_1 \ldots y_m) \rightarrow \]

\[ t \in T_{\Delta \cup Q(x_k)}[Y_m] \]

\[ \text{rank}(\sigma) = k, \text{rank}(q) = m \]