

# Theory of XML Transformations: Tree Transducers

## An Overview

Sebastian Maneth

LIACS Leiden University  
The Netherlands

Theory of XML Transformations – Feb'03 Lausanne – p.1/20

## Outline

1. XML Types = Regular Tree Languages
2. XML Transformations = Tree Transducers
3. Applications:
  - Type Checking
  - Almost Always Type Checking
  - “Static Garbage Collection”
  - Efficient Subclasses
4. Conclusions / Further Research

Theory of XML Transformations – Feb'03 Lausanne – p.2/20

## XML = Trees

```
Giorgio Busatto  
Agnus-Miegel Str. 33  
28279 Bremen
```

```
.  
. .  
Agnes Meier  
Bremen Str. 22
```

Text file of addresses

How to search for “name = Agnes”?

Theory of XML Transformations – Feb'03 Lausanne – p.3/20

## XML = Trees

```
Giorgio Busatto  
Agnus-Miegel Str. 33  
28279 Bremen
```

```
.  
. .  
Agnes Meier  
Bremen Str. 22
```

Text file of addresses

TAG it up!

How to search for “name = Agnes”?

```
<address>  
<name> Giorgio Busatto </name>  
<street> Agnes-Miegel Str. 33  
</street>  
... </address>
```

```
<address>  
<name> Agnes Meier </name>
```

XML file of addresses

Vianu: Model for manipul. of data on the web: **Tree Transducers**

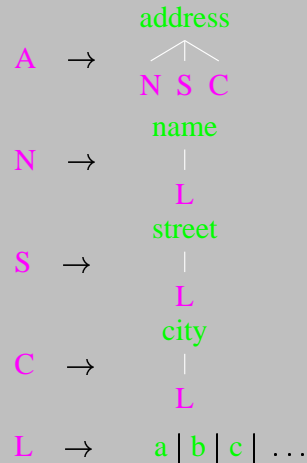
Theory of XML Transformations – Feb'03 Lausanne – p.3/20

# 1. XML Types = Regular Tree L's

Def.: An **XML type** is a regular tree language.

Formalisms:

E.g. Regular Tree Grammar

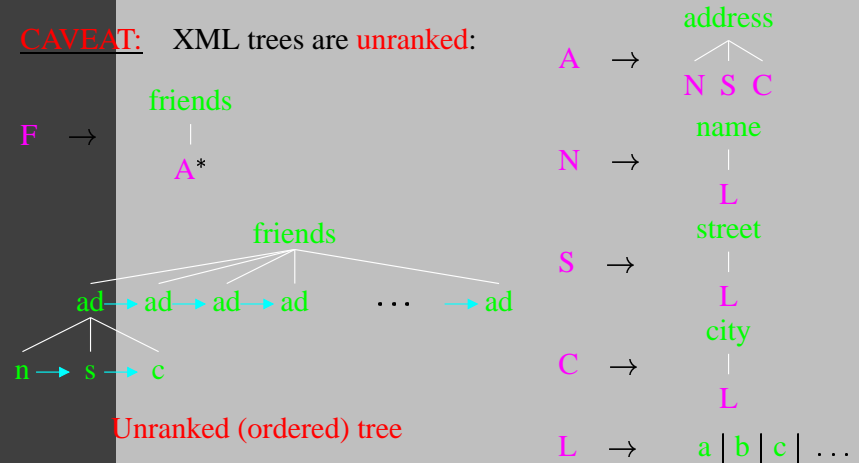


Theory of XML Transformations – Feb'03 Lausanne – p.4/20

# 1. XML Types = Regular Tree L's

Def.: An **XML type** is a regular tree language ( $\in \text{REGT}$ ).

**CAVEAT:** XML trees are **unranked**:



Theory of XML Transformations – Feb'03 Lausanne – p.4/20

# 1. XML Types = Regular Tree L's

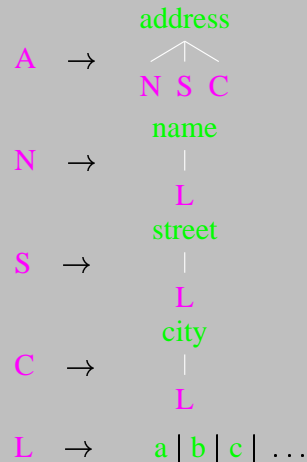
Def.: An **XML type** is a regular tree language.

Formalisms:

E.g. Regular Tree Grammar

(or: Finite Tree Automata,  
MSO, DTD,  
Schema, ...)

Class: **REGT**



Theory of XML Transformations – Feb'03 Lausanne – p.4/20

# 1. XML Types = Regular Tree L's

Def.: An **XML type** is a regular tree language ( $\in \text{REGT}$ ).

**CAVEAT:** XML trees are **unranked**:

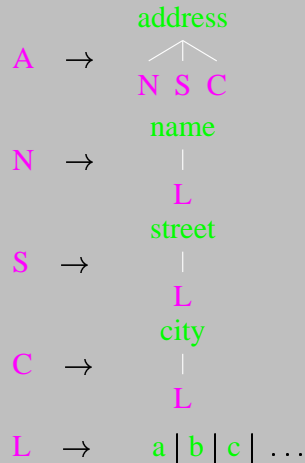


Theory of XML Transformations – Feb'03 Lausanne – p.4/20

# 1. XML Types = Regular Tree L's

Def.: An **XML type** is a regular tree language ( $\in \text{REGT}$ ).

**CAVEAT:** XML trees are **unranked**:



**Unrankedness is not important:**

En-/Decoding: VERY EASY!!!

(see also [Segoufin, PODS'03])

# XML Queries / Transformations

Def.:

- An **XML query/transformation** is a mapping  $\tau$  mapping from input to output XML documents (= trees).
- An **XML view** is a tree language  $\tau(R_{in})$ , where R is an input type (i.e.,  $R_{in} \in \text{REGT}$ ).

A **query language** should be closed under **composition**.

Def.: For a class of tree translations X, the **query language of X** is the composition closure  $X^* = \bigcup_{k \geq 1} X^k$ .

# XML Queries / Transformations

Def.:

- An **XML query/transformation** is a mapping  $\tau$  mapping from input to output XML documents (= trees).
- An **XML view** is a tree language  $\tau(R_{in})$ , where R is an input type (i.e.,  $R_{in} \in \text{REGT}$ ).

# XML Query Languages

Most popular XML Query Languages:

- **XSLT** [W3]
- **XQuery** [W3]

Theoretical model which  
“subsumes all known XML Qu./Trans. Languages”:

**The n-Pebble Tree Transducer [Milo/Suciu/Vianu2000]**

# XML Query Languages

Most popular XML Query Languages: Much nicer (I think):

- XSLT [W3] fxgrep / fxt
- XQuery [W3] [Seidl/ea.2002]

Theoretical model which  
 “subsumes all known XML Qu./Trans. Languages”:

The n-Pebble Tree Transducer [Milo/Suciu/Vianu2000]

# 2. XML Transfo. = Tree Transducers

A Brief History of Tree Transducers:

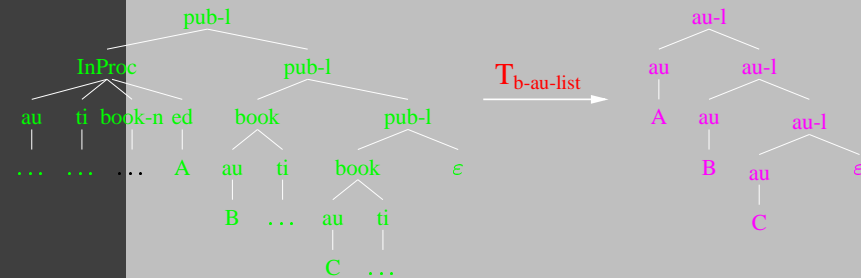
- SD-Translation [Irons1960]
- Attribute Grammar [Knuth1968]
- Top-Down / Bottom-Up [Thatcher,Wright1970]
- SD Trans. Schemes, Tree-Walking Tr’s [Aho/Ullman1980]
- Macro Tree Transducer [Courcelle/FrZ1982, Engelfriet/Vogler1985]
- 1998: Tree Transducer Book, Fülöp/Vogler
- Pebble Tree Transducer [Milo/Suciu/Vianu2000]

# 2. XML Transfo. = Tree Transducers

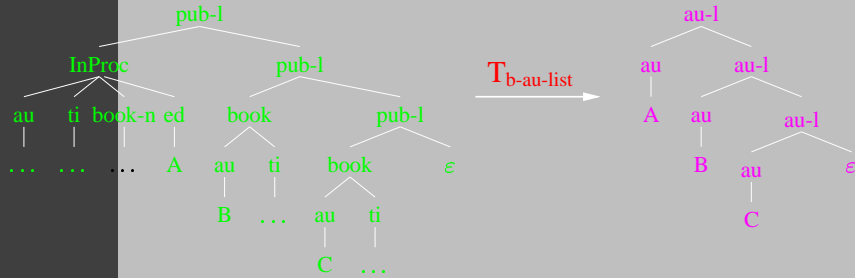
A Brief History of Tree Transducers:

- SD-Translation [Irons1960]
- Attribute Grammar [Knuth1968]
- Top-Down / Bottom-Up [Thatcher,Wright1970]
- SD Trans. Schemes, Tree-Walking Tr’s [Aho/Ullman1980]
- Macro Tree Transducer [Courcelle/FrZ1982, Engelfriet/Vogler1985]
- 1998: Tree Transducer Book, Fülöp/Vogler
- Pebble Tree Transducer [Milo/Suciu/Vianu2000]

# Example: Top-Down Tr. Tr.



## Example: Top-Down Tr. Tr.



$T_{b-au-list}$  rules:  $\langle q_0, pub-l \rangle \rightarrow au-l(\langle q_0, \downarrow_1 \rangle, \langle q_0, \downarrow_2 \rangle)$   
 states =  $\{q_0, q_{id}\}$   
 $\langle q_0, InProc \rangle \rightarrow \langle q_0, \downarrow_4 \rangle$   
 $\langle q_0, book \rangle \rightarrow \langle q_0, \downarrow_1 \rangle$   
 $\langle q_0, ed \rangle \rightarrow au(\langle q_{id}, \downarrow_1 \rangle)$   
 $\langle q_0, au \rangle \rightarrow au(\langle q_{id}, \downarrow_1 \rangle)$   
 $\langle q_0, \epsilon \rangle \rightarrow \epsilon$

## Adding Parameters: Macro Tr. Tr.

Rule of a Macro Tree Transducer (MTT):

E.g.

$$\langle q, \sigma \rangle (y_1, y_2, y_3) \rightarrow \langle p, \downarrow_2 \rangle (y_3, \langle r, \downarrow_1 \rangle (y_1, y_2), \alpha)$$

where

$\sigma$  : input symbol (of rank 2)

$q, p$  : states of rank 3 (i.e., with 3 parameters)

$r$  : state of rank 2

$\alpha$  : output symbol (of rank 0)

## Top-Down Tree Transducers

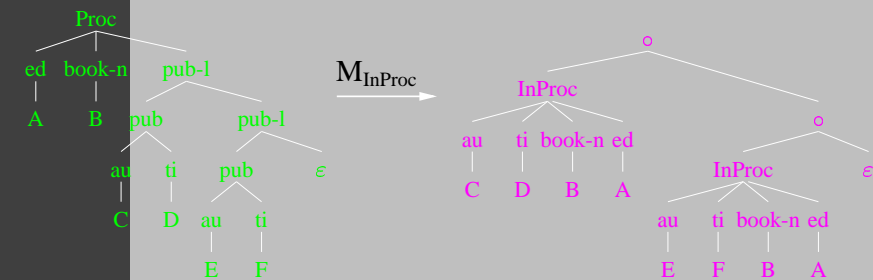
### Advantages:

- Linear TOPs preserve REGT:  $\text{lin-TOP}(\text{REGT}) \subseteq \text{REGT}$ .  
Applications: e.g. forward type inference
- Deterministic TOPs are closed under composition
- Equivalence is decidable! [Esik81]

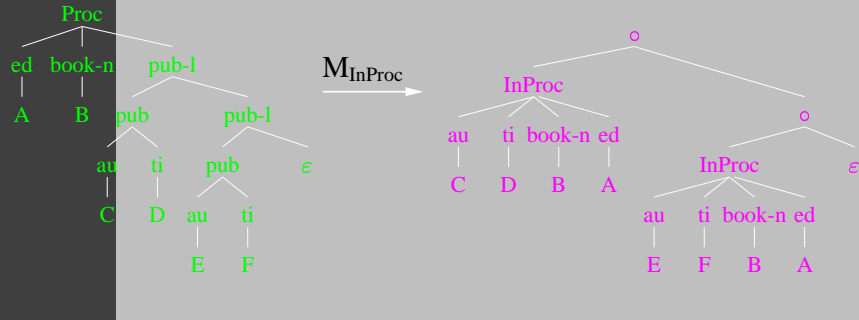
### Disadvantages:

- heavily depend on data representation of input trees
- sensitive to unrankedness: TOPs on bin. encodings CANNOT do all unranked TOPs! [M/Neven1999]

## Example MTT

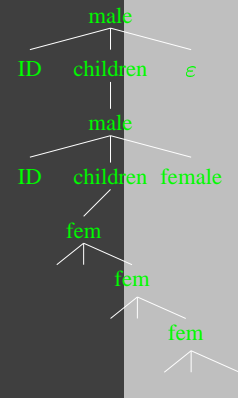


# Example MTT



$M_{InProc} \langle q_0, Proc \rangle \rightarrow \langle q, \downarrow_3 \rangle ( \langle q_{skip}, \downarrow_1 \rangle, \langle q_{skip}, \downarrow_2 \rangle )$   
 states =  $\langle q, pub-l \rangle (y_1, y_2) \rightarrow o ( InProc ( \langle q_{skip}, \downarrow_1 \rangle, \langle q_{skip}, \downarrow_2 \rangle, y_1, y_2 ), \langle q, \downarrow_2 \rangle )$   
 $\{ q_0^{(0)}, q_{skip}^{(0)}, q_{id}^{(0)} \}$   
 $\langle q, \epsilon \rangle (y_1, y_2) \rightarrow \epsilon$

# Example 1-PTT: Finding Patterns



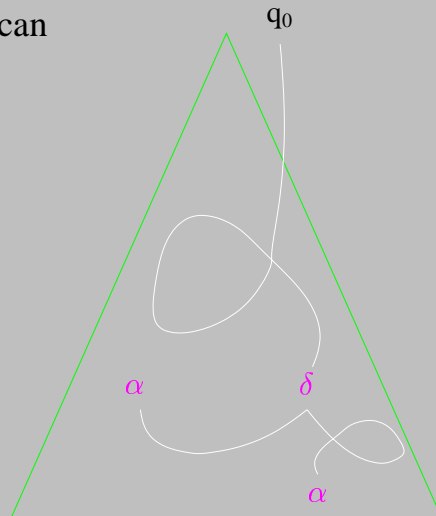
Look for:

- x : has only daughters
- y : male descendant of x

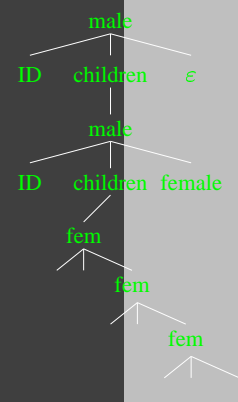
# New: The n-Pebble Tr. Tr. (n-PTT)

At each moment an n-PTT can

- move up/down/stay
- drop new pebble
- lift last pebble
- generate output and spawn new (indep.) computations



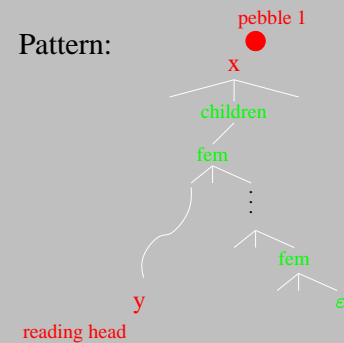
# Example 1-PTT: Finding Patterns



Look for:

- x : has only daughters
- y : male descendant of x

Pattern:



## Comparison DPTT with DMTT

1.  $n\text{-DPTT} \subseteq 0\text{-DPTT}^{n+1} \subseteq \text{DMTT}^{n+1}$ .
2.  $\text{DMTT} \subseteq 0\text{-DPTT}^3$ .

$\Rightarrow \text{DPTT}^* = \text{DMTT}^*$ : Both have the **same query power**.

## Comparison DPTT with DMTT

1.  $n\text{-DPTT} \subseteq 0\text{-DPTT}^{n+1} \subseteq \text{DMTT}^{n+1}$ .
2.  $\text{DMTT} \subseteq 0\text{-DPTT}^3$ .

$\Rightarrow \text{DPTT}^* = \text{DMTT}^*$ : Both have the **same query power**.

DPTT	DMTT
<ul style="list-style-type: none"> <li>● operational</li> <li>● might not terminate</li> </ul> <p>(DEXPTIME-complete!!)</p> <ul style="list-style-type: none"> <li>● <math>\text{REGT} \subseteq \text{Dom}(\text{DPTT})???</math></li> <li>● size inc. <math>\leq \text{exp}</math></li> </ul>	<ul style="list-style-type: none"> <li>● declarative</li> <li>● always terminate</li> </ul> <ul style="list-style-type: none"> <li>● <math>\text{REGT} = \text{Dom}(\text{DMTT})</math></li> <li>● size inc. <math>\leq \text{double exp}</math></li> </ul>

## 3. Applications

Three important properties of compositions of MTTs:

Let  $\tau \in \text{MTT}^*$  and  $\mathbf{R}_{\text{in}}, \mathbf{R}_{\text{out}} \in \text{REGT}$ .

- I**  $\tau^{-1}$  is effectively regular [Engelfriet/Vogler1985]
- II** Finiteness of  $\tau(\mathbf{R}_{\text{in}})$  decidable [Drewes/Engelfriet1998]
- III** Translation  $\tau(s)=t$  can be computed (for  $\tau \in \text{DMTT}^*$ )  
in  $\text{DTIME}(O(|s| + |t|))$  [M2002]

## Applications of I: Type Checking

Type Checking Problem

<b>input:</b>	types $\mathbf{R}_{\text{in}}, \mathbf{R}_{\text{out}} \in \text{REGT}$ , translation $\tau$
<b>output:</b>	$\begin{cases} \text{"yes"} & \text{if } \tau(\mathbf{R}_{\text{in}}) \subseteq \mathbf{R}_{\text{out}} \\ \text{"no"} & \text{otherwise.} \end{cases}$

## Applications of I: Type Checking

Type Checking Problem

<b>input:</b>	types $R_{in}, R_{out} \in \text{REGT}$ , translation $\tau$
<b>output:</b>	$\begin{cases} \text{"yes"} & \text{if } \tau(R_{in}) \subseteq R_{out} \\ \text{"no"} & \text{otherwise.} \end{cases}$

Decidable: (using I:  $\tau^{-1}$  preserves REGT)

$$\tau(R_{in}) \subseteq R_{out} \text{ iff } \underbrace{\tau^{-1}(R_{out})}_{\in \text{REGT}} \subseteq R_{in}$$

and inclusion of REGT is decidable.

## Almost Always Type Checking I+II

Almost Always Type Checking Problem

<b>input:</b>	types $R_{in}, R_{out} \in \text{REGT}$ , translation $\tau$
<b>output:</b>	$\begin{cases} \text{"yes"} & \text{if } \tau(R_{in}) - R_{out} \text{ is finite} \\ \text{"no"} & \text{otherwise.} \end{cases}$

If “yes” then the finite set of “input exceptions”  $E$

$$E = R_{in} - \tau^{-1}(\overline{R_{out}})$$

can be computed.

## Almost Always Type Checking I+II

Almost Always Type Checking Problem

<b>input:</b>	types $R_{in}, R_{out} \in \text{REGT}$ , translation $\tau$
<b>output:</b>	$\begin{cases} \text{"yes"} & \text{if } \tau(R_{in}) - R_{out} \text{ is finite} \\ \text{"no"} & \text{otherwise.} \end{cases}$

## Applications of III

MTTs are fu. programs based on **primitive recursion**.  
Elimination of intermediate data structures (e.g. deforestation).  
Using composition properties of DMTTs:

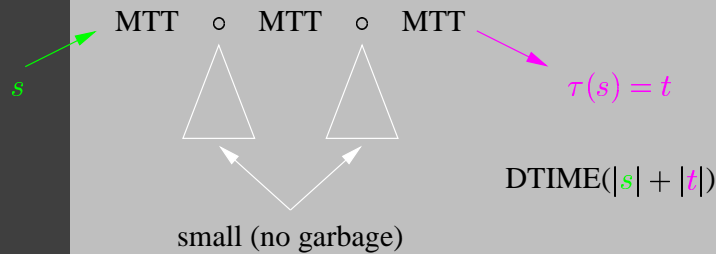
$$\text{DTOP} \circ \text{DMTT} \circ \text{DTOP} \subseteq \text{DMTT}.$$

# Applications of III

MTTs are fu. programs based on **primitive recursion**.  
 Elimination of intermediate data structures (e.g. deforestation).  
 Using composition properties of DMTTs:

$$DTOP \circ DMTT \circ DTOPT \subseteq DMTT.$$

III: “automatic generation of garbage-free compilers”



Theory of XML Transformations – Feb’03 Lausanne – p.18/20

## 4. Conclusions / Further Reserach

### Cross Fertilization:



E.g.:

**I – III** hold for (compositions of) PTTs!!

In particular:

- **Almost Always Type Checking is decidable.**
- **Computable in DTIME(O(|input| + |output|)).**

(which improve the results of [Milo/Suciu/Vianu2000])

Theory of XML Transformations – Feb’03 Lausanne – p.19/20

## 4. Conclusions / Further Reserach

**New Result:** Collapse of DMTT hierarchy, for Linear Size Increase:

$$\begin{aligned} DMTT^* \cap LSIZE &\subseteq DMTT \cap LSIZE && [M2003] \\ &= MSOTT && [Engelfriet/M2001] \end{aligned}$$

⇒ MSOTT is a **decidable** subclass of DMTT\*!!

- Complexity of Inv. Type Inference for subclasses?
- What if subclasses of REGT are considered?

Theory of XML Transformations – Feb’03 Lausanne – p.20/20

## 4. Conclusions / Further Reserach

**New Result:** Collapse of DMTT hierarchy, for Linear Size Increase:

$$\begin{aligned} DMTT^* \cap LSIZE &\subseteq DMTT \cap LSIZE && [M2003] \\ &= MSOTT && [Engelfriet/M2001] \end{aligned}$$

⇒ MSOTT is a **decidable** subclass of DMTT\*!!

- Complexity of Inv. Type Inference for subclasses?
- What if subclasses of REGT are considered?

**THE END**

Theory of XML Transformations – Feb’03 Lausanne – p.20/20