

---

# Theory of XML Transformations: Tree Transducers

## *An Overview*

Sebastian Maneth

LIACS Leiden University  
The Netherlands

# 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

# 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”?

# XML = Trees

Giorgio Busatto  
Agnes-Miegel Str. 33  
28279 Bremen

.  
. .  
.

Agnes Meier  
Bremen Str. 22

.  
. .  
.

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>
```

.  
.

Text file of addresses

XML file of addresses

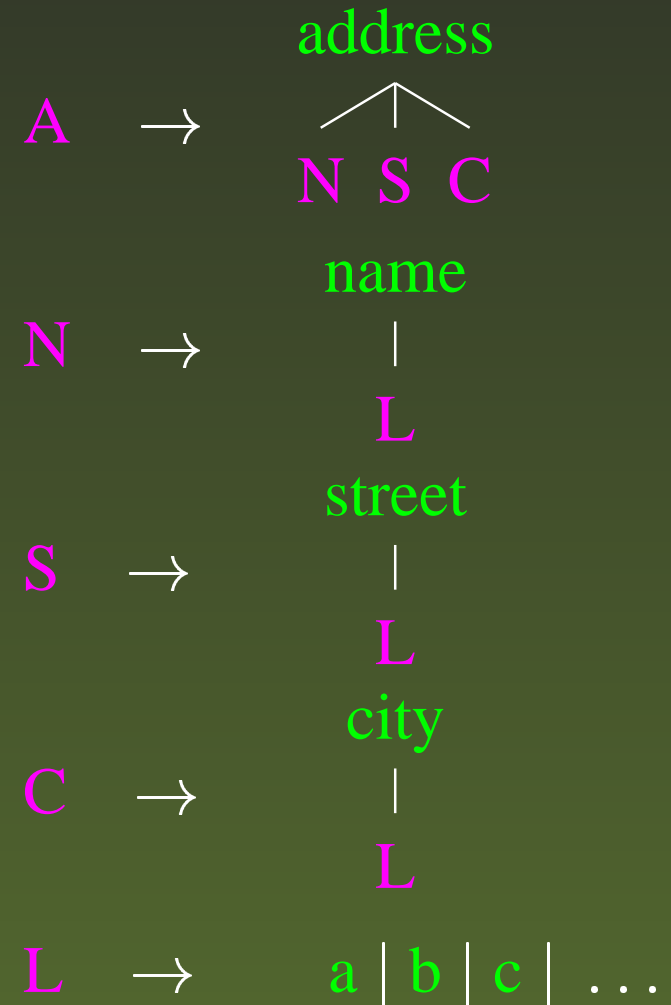
Vianu: Model for manipul. of data on the web: **Tree Transducers**

# 1. XML Types = Regular Tree L's

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

Formalisms:

E.g. Regular Tree Grammar



# 1. XML Types = Regular Tree L's

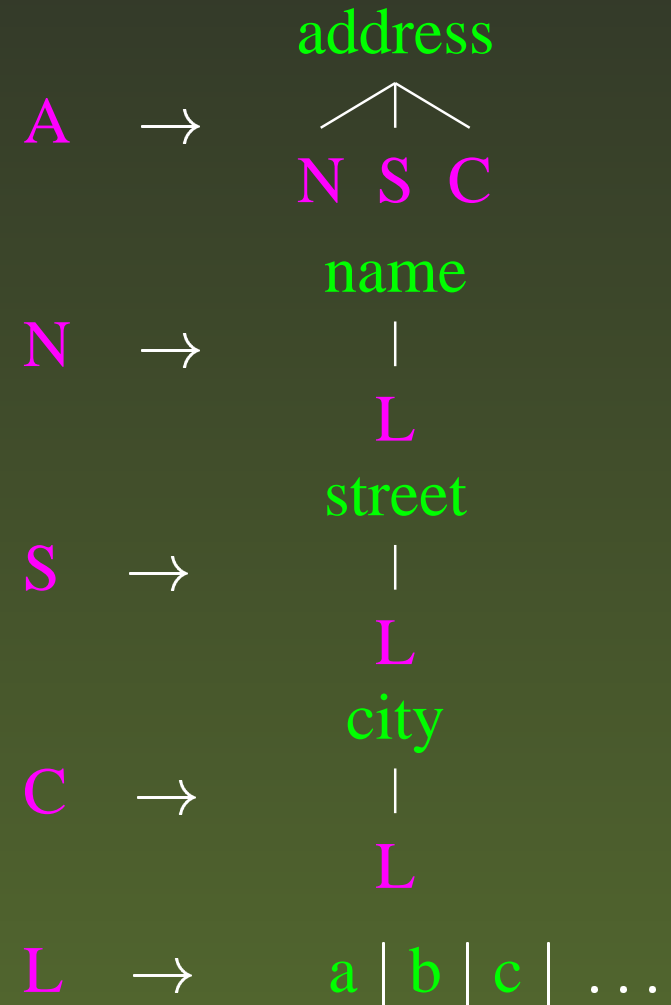
Def.: An **XML type** is a regular tree language.

Formalisms:

E.g. Regular Tree Grammar

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

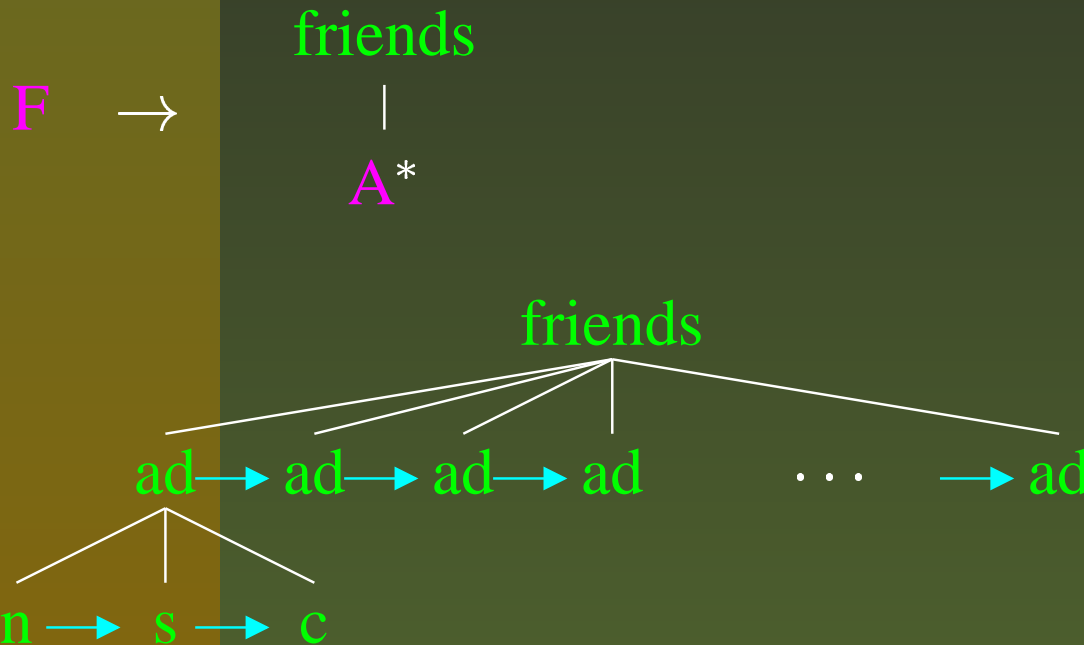
Class: **REGT**



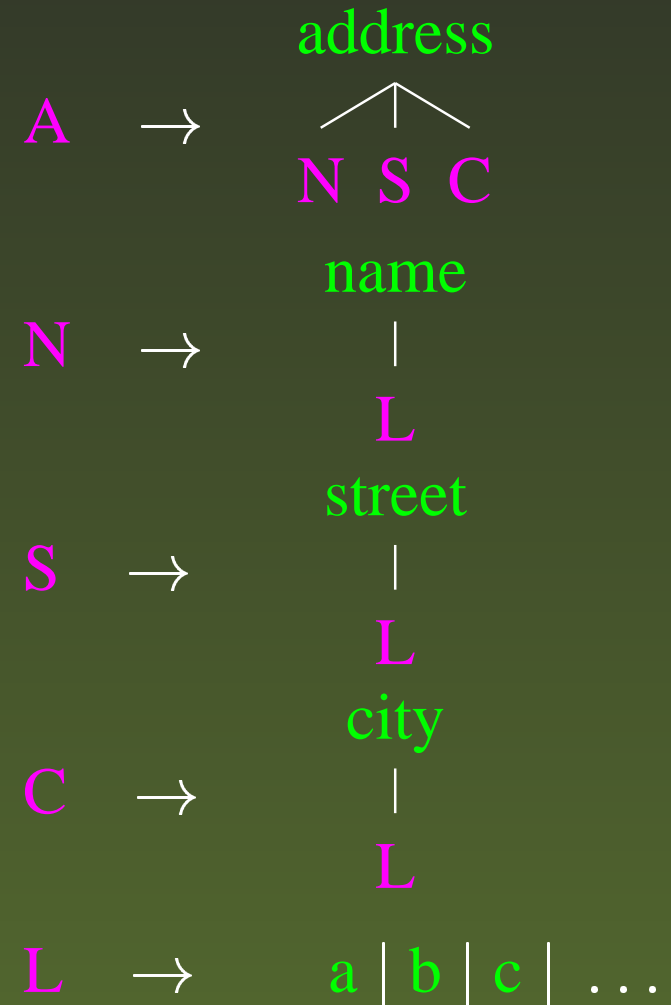
# 1. XML Types = Regular Tree L's

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

CAVEAT: XML trees are **unranked**:



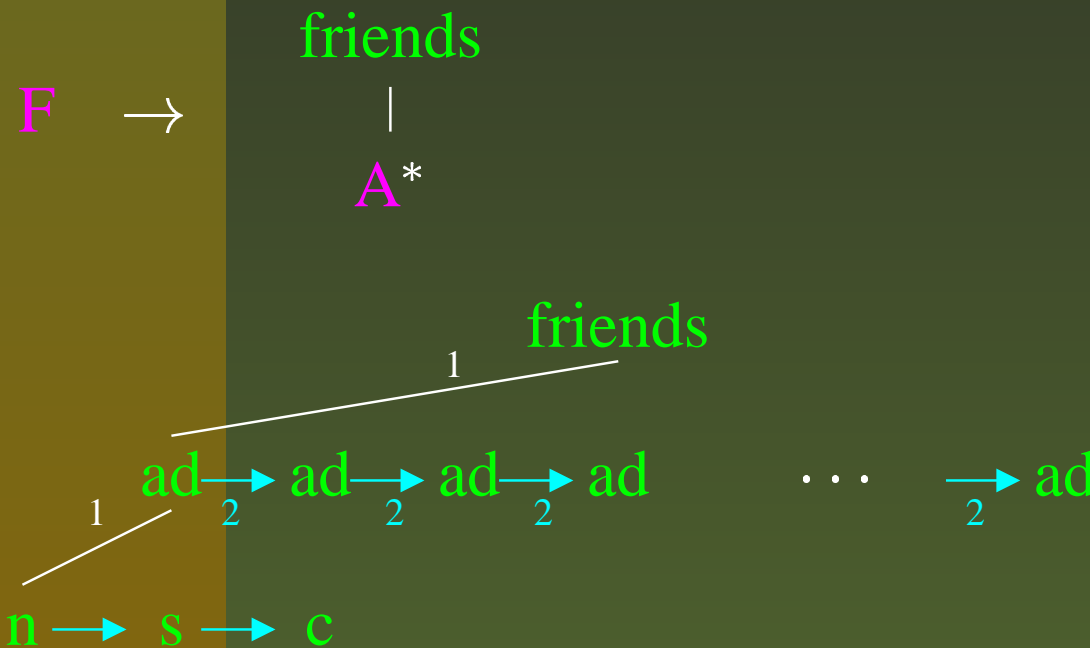
Unranked (ordered) tree



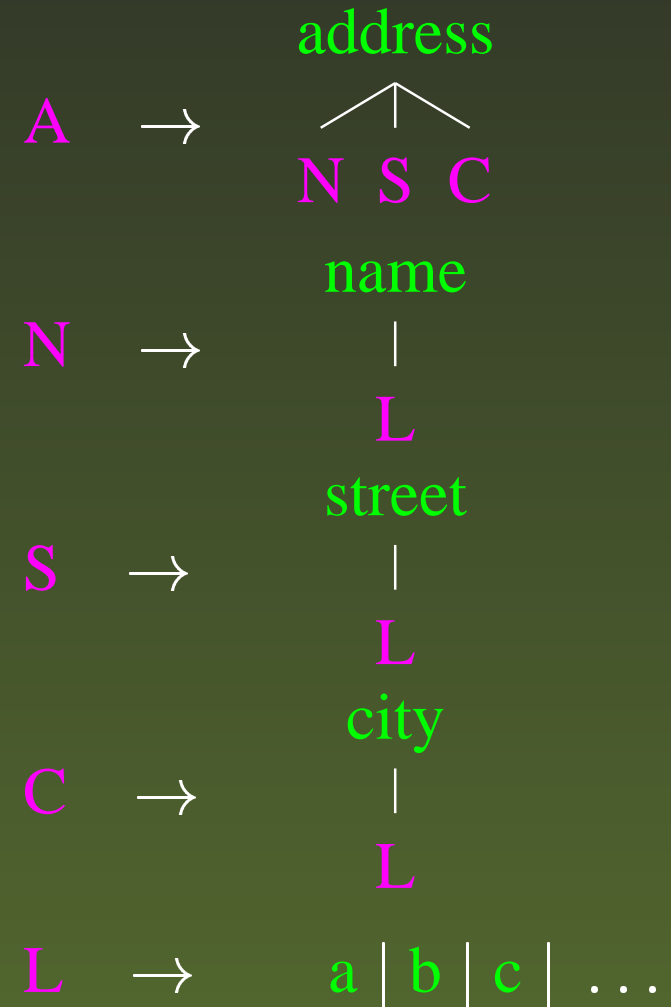
# 1. XML Types = Regular Tree L's

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

CAVEAT: XML trees are **unranked**:



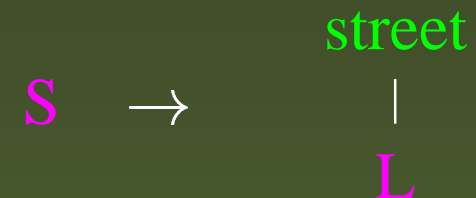
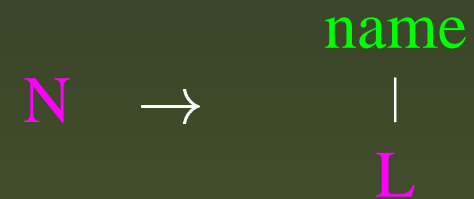
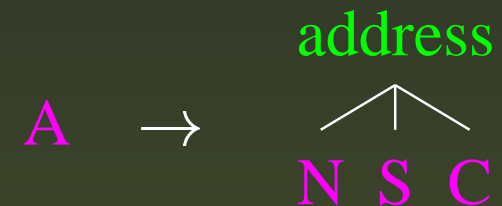
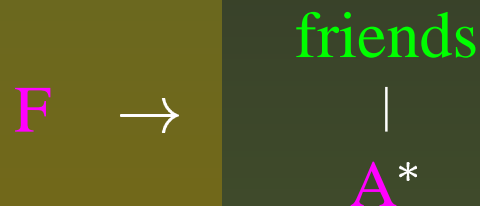
Encoding as a binary tree



# 1. XML Types = Regular Tree L's

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

# 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.,  $R_{in} \in REGT$ ).

---

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 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:

- XSLT [W3]
- XQuery [W3]

Much nicer (I think):

fxgrep / fxt

[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]

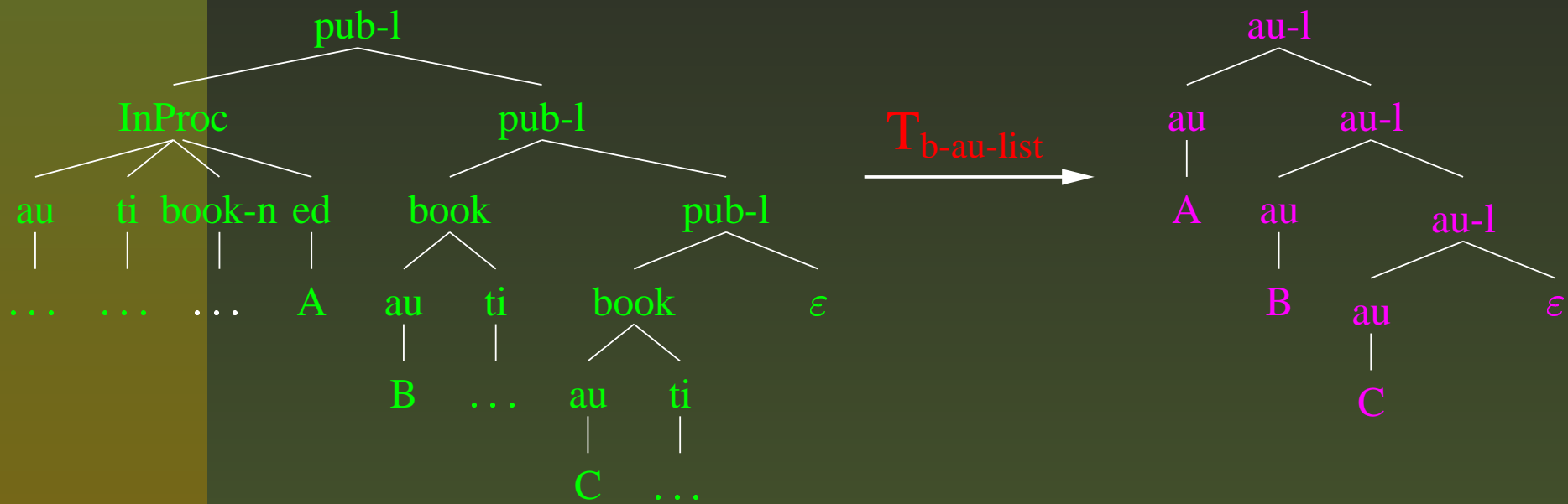
## 2. XML Transfo. = Tree Transducers

---

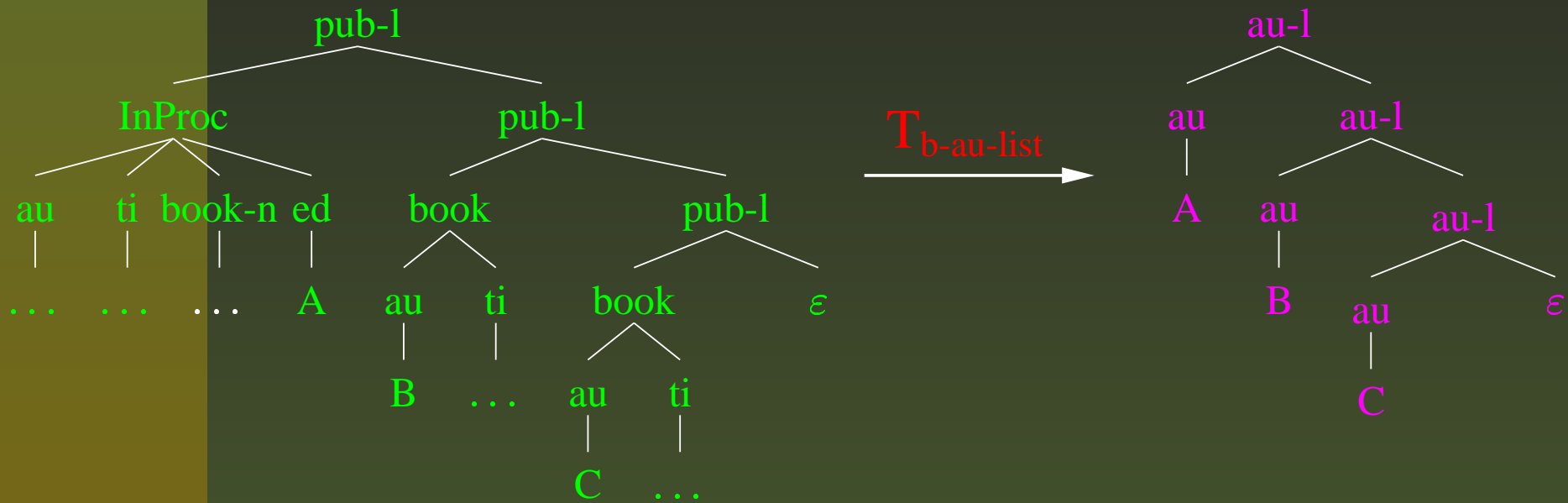
### A Brief History of Tree Transducers:

- 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.



$T_{b-au-list}$

states =  $\{ q_0, q_{id} \}$

rules:

$\langle q_0, \text{pub-l} \rangle$	$\rightarrow$	$\text{au-l} ( \langle q_0, \downarrow_1 \rangle, \langle q_0, \downarrow_2 \rangle )$
$\langle q_0, \text{InProc} \rangle$	$\rightarrow$	$\langle q_0, \downarrow_4 \rangle$
$\langle q_0, \text{book} \rangle$	$\rightarrow$	$\langle q_0, \downarrow_1 \rangle$
$\langle q_0, \text{ed} \rangle$	$\rightarrow$	$\text{au} ( \langle q_{id}, \downarrow_1 \rangle )$
$\langle q_0, \text{au} \rangle$	$\rightarrow$	$\text{au} ( \langle q_{id}, \downarrow_1 \rangle )$
$\langle q_0, \epsilon \rangle$	$\rightarrow$	$\epsilon$

# 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]

# Adding Parameters: Macro Tr. Tr.

Rule of a Macro Tree Transducer (MTT):

E.g.

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

where

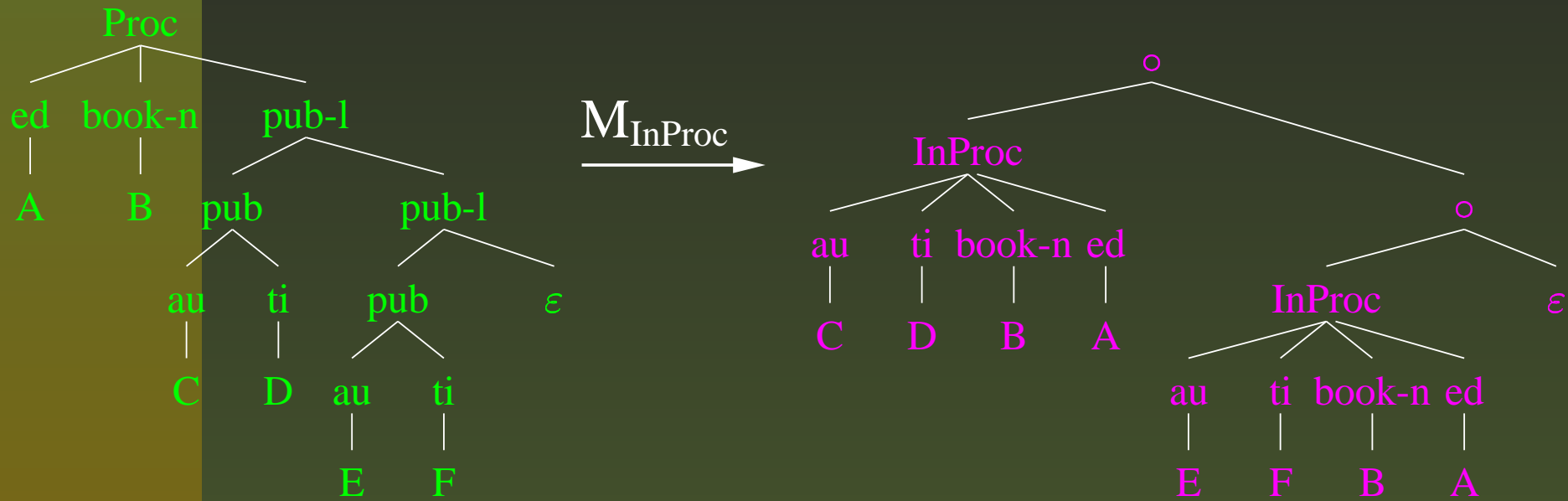
$\sigma$  : input symbol (of rank 2)

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

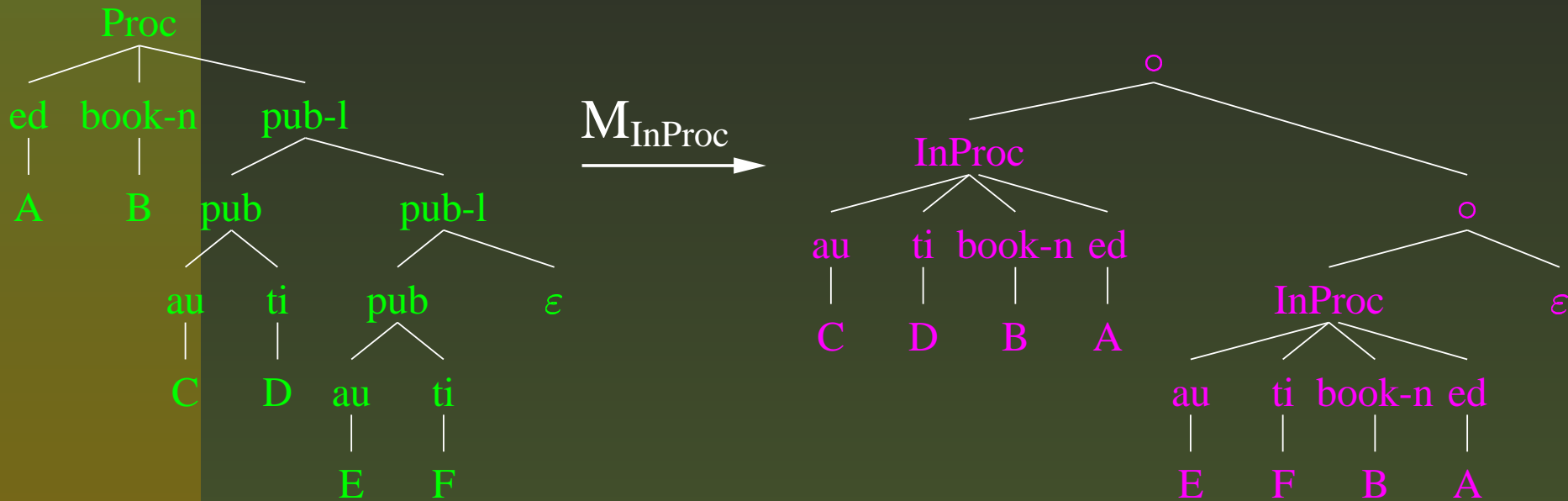
$r$  : state of rank 2

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

# Example MTT



# Example MTT

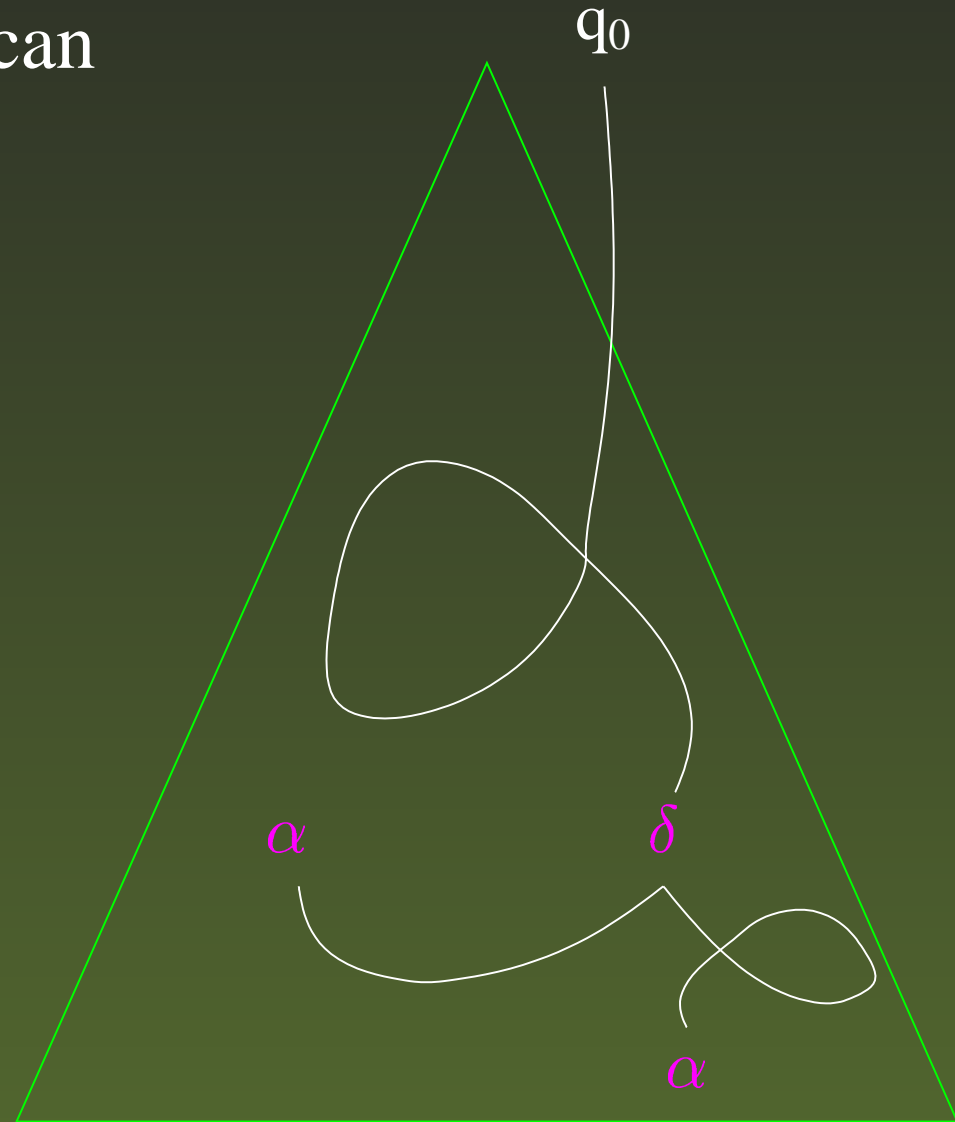


$M_{\text{InProc}}$	$\langle q_0, \text{Proc} \rangle$	$\rightarrow$	$\langle q, \downarrow_3 \rangle ( \langle q_{\text{skip}}, \downarrow_1 \rangle, \langle q_{\text{skip}}, \downarrow_2 \rangle )$
states =	$\langle q, \text{pub-l} \rangle ( y_1, y_2 )$	$\rightarrow$	$\circ ( \text{InProc} ( \langle q_{\text{skip}}, \downarrow_1 \rangle, \langle q_{\text{skip}}, \downarrow_2 \rangle, y_1, y_2 ), \langle q, \downarrow_2 \rangle )$
	$\{ q_0^{(0)}, q^{(2)}, q_{\text{skip}}^{(0)}, q_{\text{id}}^{(0)} \}$		
	$\langle q, \epsilon \rangle ( y_1, y_2 )$	$\rightarrow$	$\epsilon$

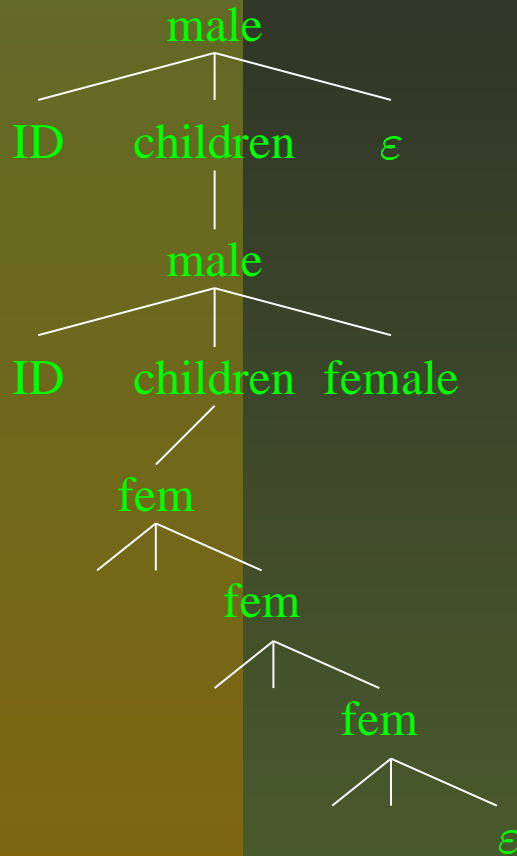
# New: The n-Pebble Tr. Tr. (n-PTT)

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

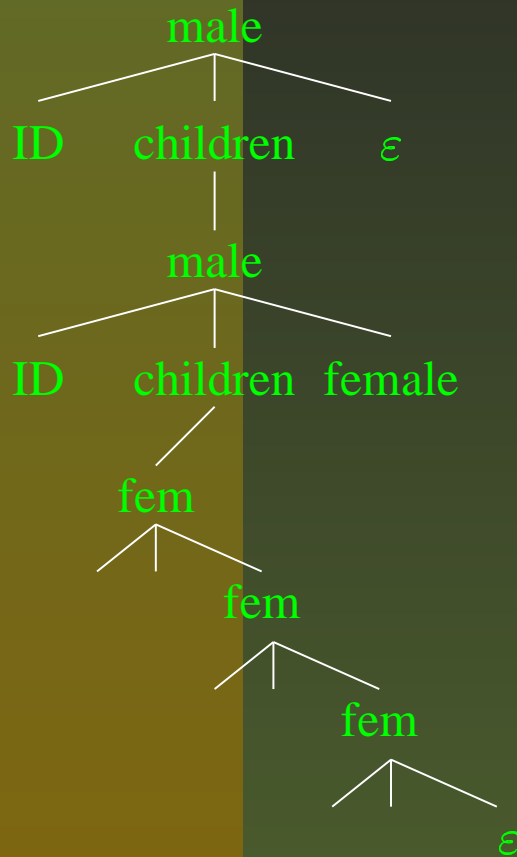


Look for:

**x** : has only daughters

**y** : male descendant of **x**

# Example 1-PTT: Finding Patterns

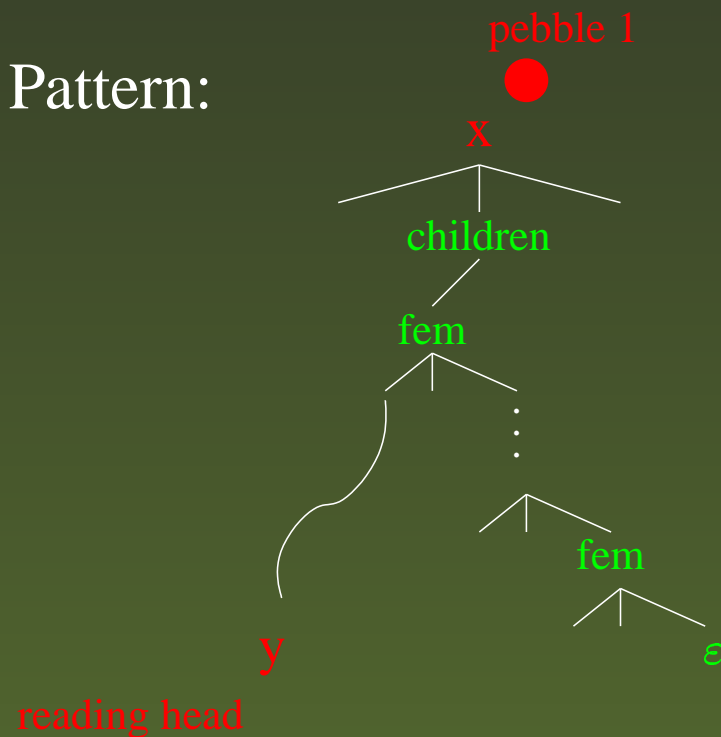


Look for:

**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 (DEXPTIME-complete!!)</li><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><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  $R_{\text{in}}, R_{\text{out}} \in \text{REGT}$ .

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

# Applications of I: Type Checking

## Type Checking Problem

**input:** types  $R_{in}, R_{out} \in \text{REGT}$ , translation  $\tau$

**output:**  $\begin{cases} \text{“yes”} & \text{if } \tau(R_{in}) \subseteq R_{out} \\ \text{“no”} & \text{otherwise.} \end{cases}$

# Applications of **I**: Type Checking

## Type Checking Problem

**input:** types  $R_{in}, R_{out} \in \text{REGT}$ , translation  $\tau$

**output:**  $\begin{cases} \text{“yes”} & \text{if } \tau(R_{in}) \subseteq R_{out} \\ \text{“no”} & \text{otherwise.} \end{cases}$

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

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

and inclusion of REGT is decidable.

# Almost Always Type Checking I+II

## Almost Always Type Checking Problem

**input:** types  $R_{in}, R_{out} \in \text{REGT}$ , translation  $\tau$

**output:**  $\begin{cases} \text{“yes”} & \text{if } \tau(R_{in}) - R_{out} \text{ is finite} \\ \text{“no”} & \text{otherwise.} \end{cases}$

# Almost Always Type Checking I+II

## Almost Always Type Checking Problem

**input:** types  $R_{in}, R_{out} \in REGT$ , translation  $\tau$

**output:**  $\begin{cases} \text{“yes”} & \text{if } \tau(R_{in}) - R_{out} \text{ is finite} \\ \text{“no”} & \text{otherwise.} \end{cases}$

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

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

can be computed.

# 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

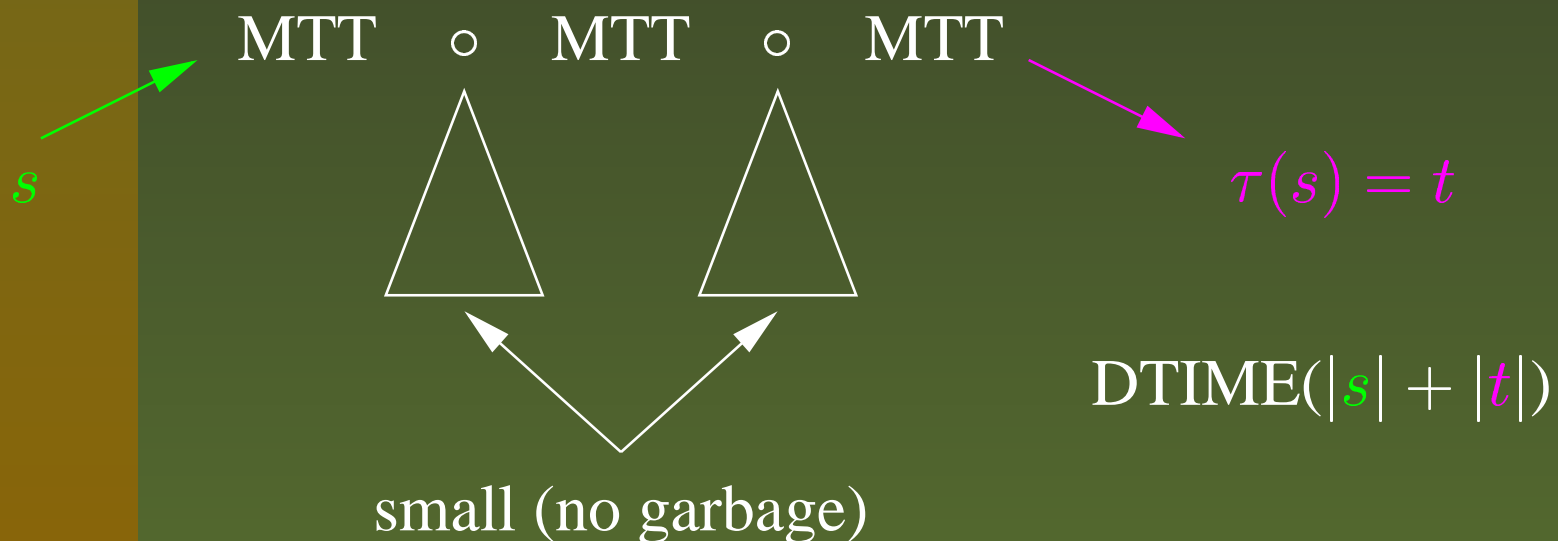
MTTs are fu. programs based on **primitive recursion**.

Elimination of intermediate data structures (e.g. deforestation).

Using composition properties of DMTTs:

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

**III**: “automatic generation of garbage-free compilers”



# 4. Conclusions / Further Reserach

## Cross Fertilization:

Tree Language Theory



XML Processing/Databases

E.g.:

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

In particular:

- **Almost Always Type Checking is decidable.**
- **Computable in  $D\text{TIME}(O(|\text{input}| + |\text{output}|))$ .**

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

# 4. Conclusions / Further Reserach

---

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

$$\text{DMTT}^* \cap \text{LSIZE} \subseteq \text{DMTT} \cap \text{LSIZE} \quad [\text{M2003}]$$

$$= \text{MSOTT} \quad [\text{Engelfriet/M2001}]$$

$\Rightarrow$  MSOTT is a **decidable** subclass of DMTT\*!!

---

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

# 4. Conclusions / Further Reserach

---

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

$DMTT^* \cap LSIZE \subseteq DMTT \cap LSIZE$  [M2003]

$= MSOTT$  [Engelfriet/M2001]

$\Rightarrow$  MSOTT is a **decidable** subclass of DMTT\*!!

---

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

**THE END**