
Theory of XML Transformations: Tree Transducers

Pebbles vs. Parameters

Sebastian Maneth

LIACS Leiden University
The Netherlands

Outline

- XML Motivation
- The n-Pebble Tree Transducer (n-PTT)
- Result 1: Decomposition into 0-PTTs
- Result 2: $0\text{-PTT} \subseteq \text{Macro Tree Transducer (MTT)}$
- Result 3: As Query Languages: $\text{PTT} = \text{MTT}$
- Type Checking Problems
- Further Research Topics

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

.
. .
. .

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: Theoretical Model for data on the web: **Tree Transducers**

XML Types = Regular Tree L's

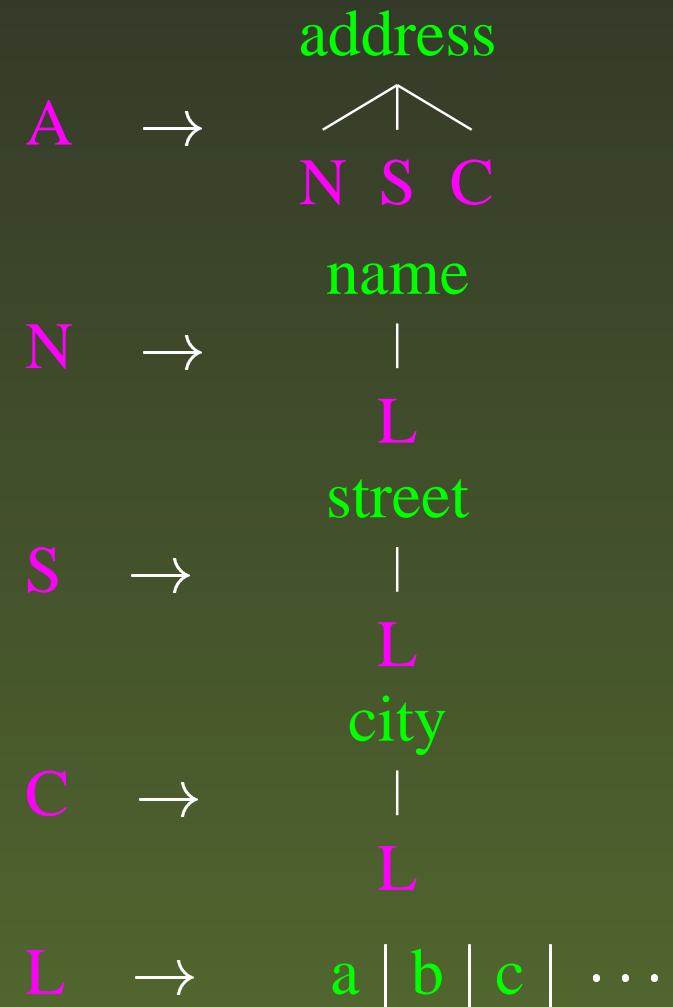
Def.: An **XML type** is a regular tree language.

XML Types = Regular Tree L's

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

Formalisms:

E.g. Regular Tree Grammar



XML Types = Regular Tree L's

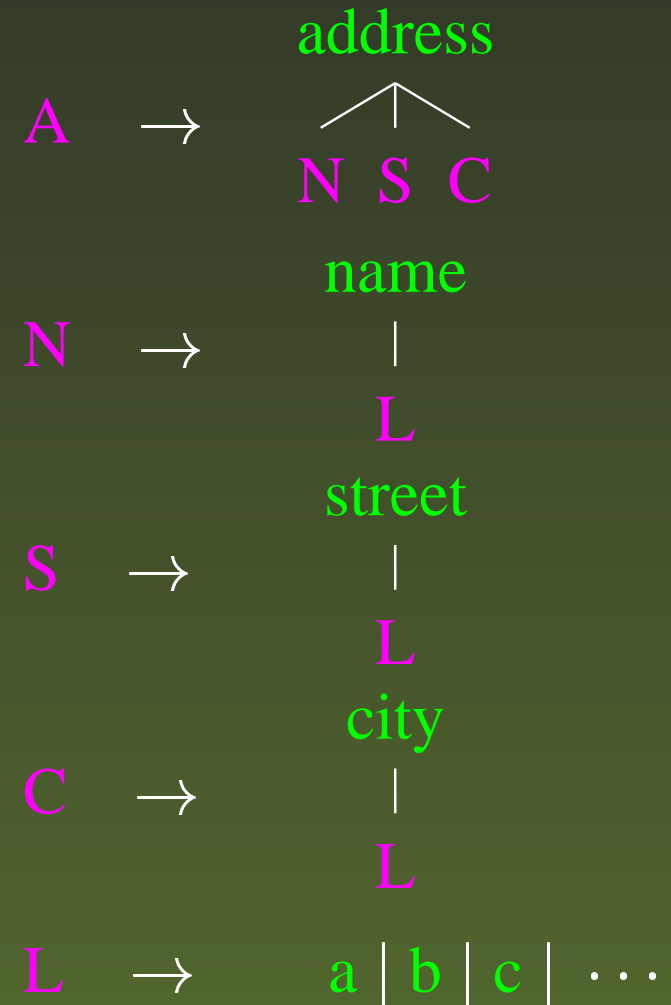
Def.: An **XML type** is a regular tree language.

Formalisms:

E.g. Regular Tree Grammar

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

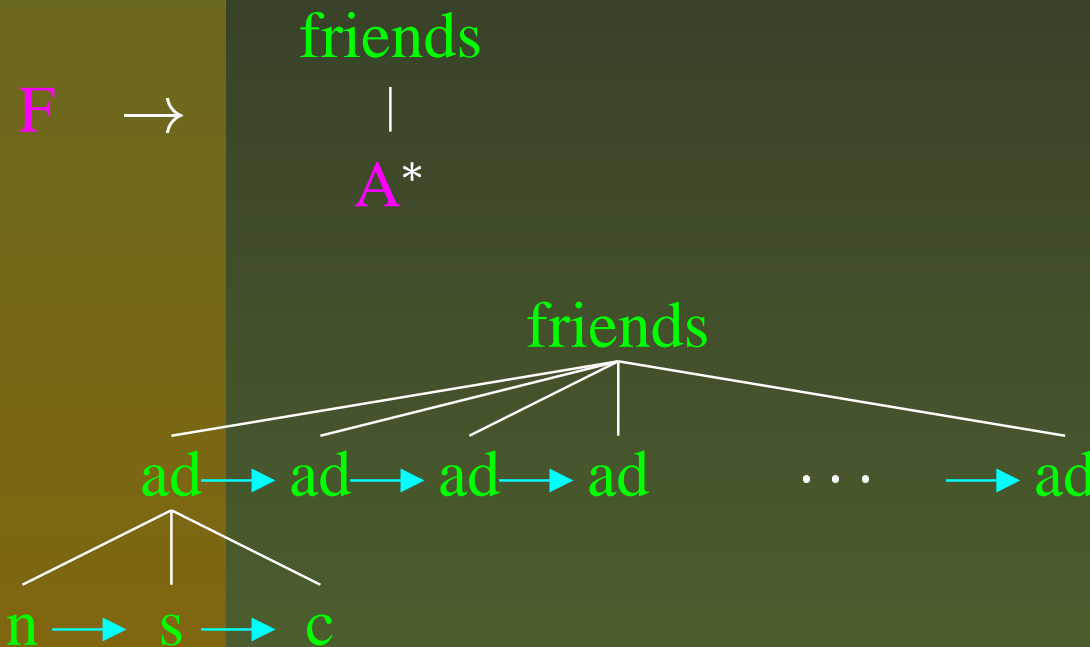
Class: **REGT**



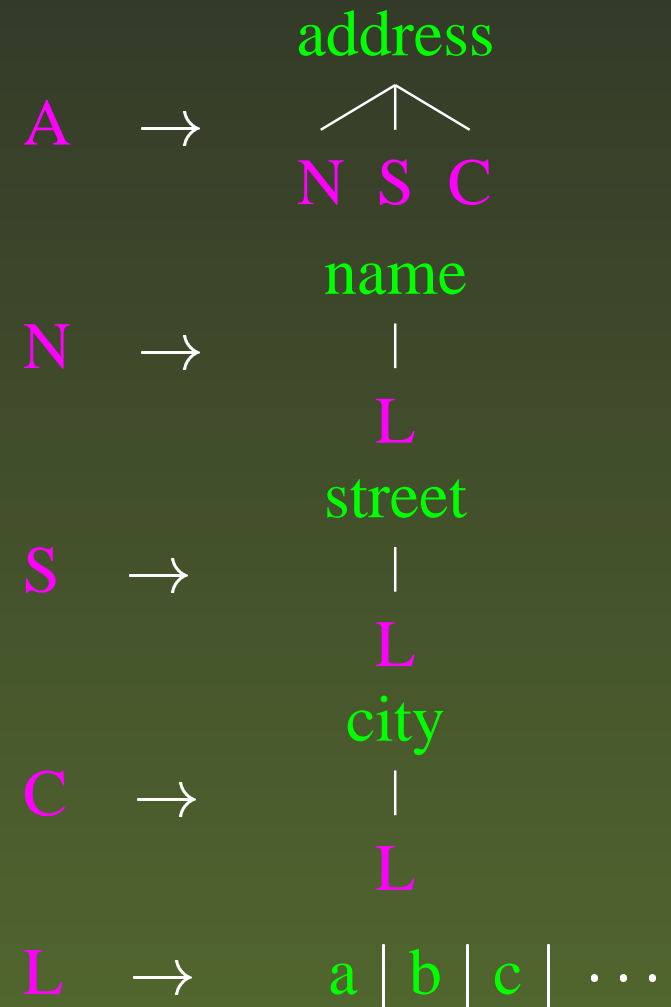
XML Types = Regular Tree L's

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

CAVEAT: XML trees are **unranked**:



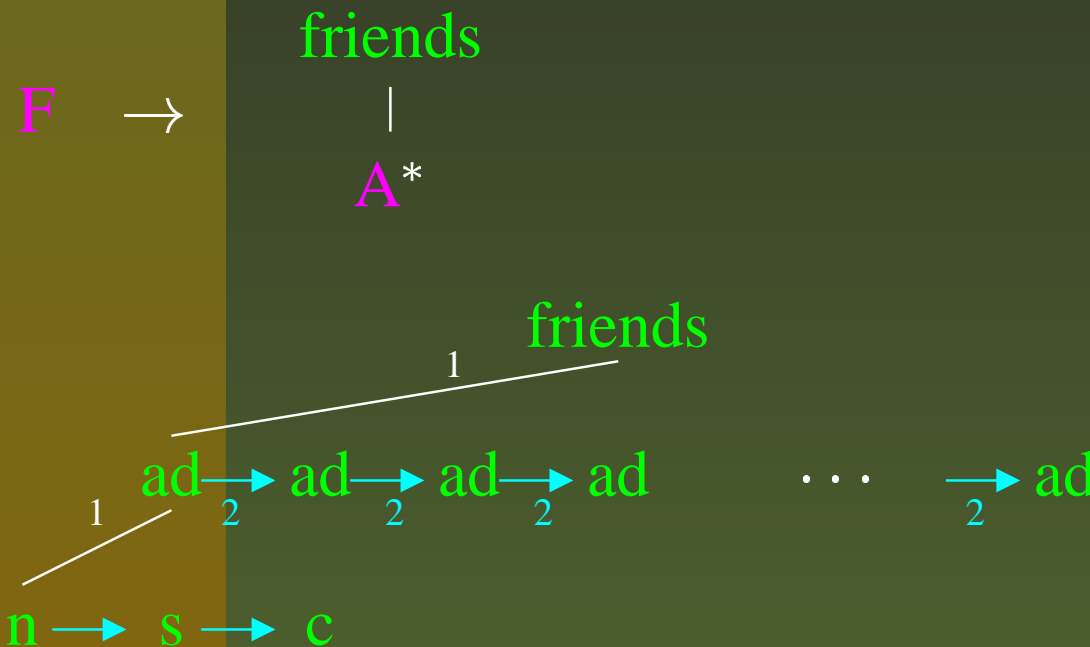
Unranked (ordered) tree



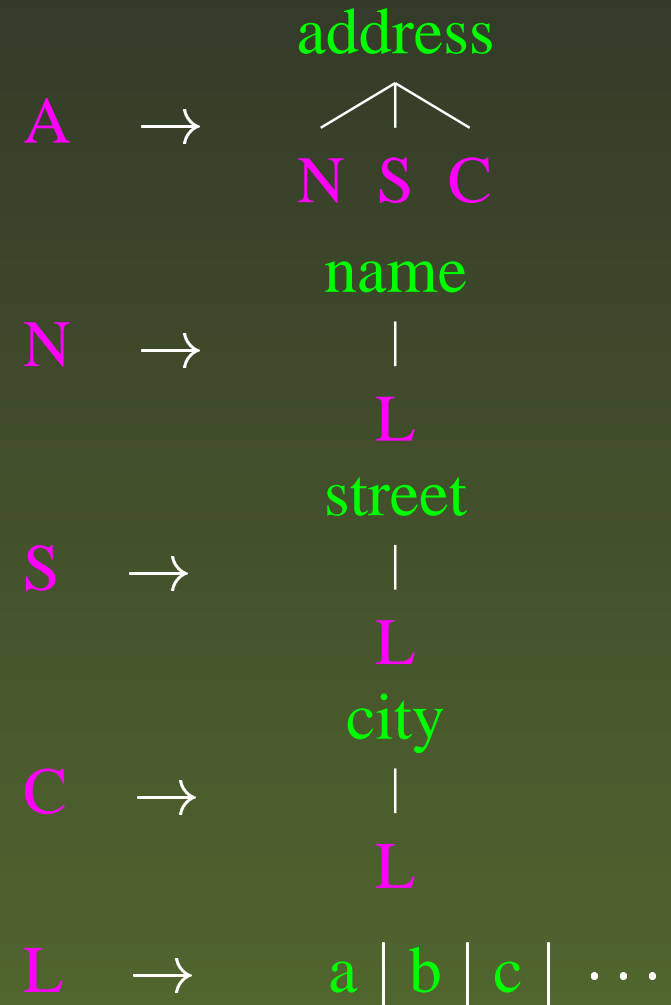
XML Types = Regular Tree L's

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

CAVEAT: XML trees are **unranked**:



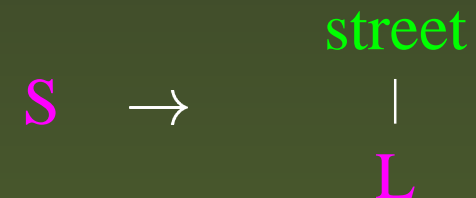
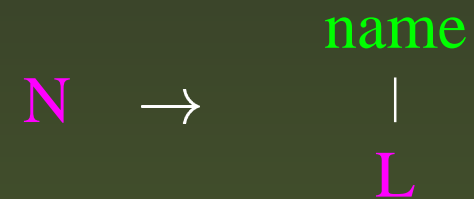
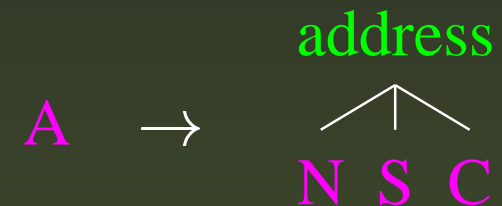
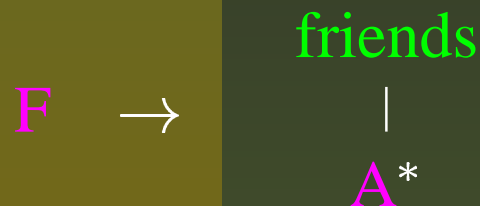
Encoding as a binary tree



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 and Transformations

Def.: An **XML query** is a τ mapping from input to output XML documents (= trees).

XML Queries and Transformations

Def.: An **XML query** is a τ mapping from input to output XML documents (= trees).

Def.: An **XML view** is a tree language $\tau(R)$, where

- τ is an XML query and
- R is an input type (i.e., $R \in \text{REGT}$)

XML Queries and Transformations

Def.: An **XML query** is a τ mapping from input to output XML documents (= trees).

Def.: An **XML view** is a tree language $\tau(\mathbb{R})$, where

- τ is an XML query and
- \mathbb{R} is an input type (i.e., $\mathbb{R} \in \text{REGT}$)

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 Query Languages

Most popular XML Query Languages:

- XSLT
- XQuery

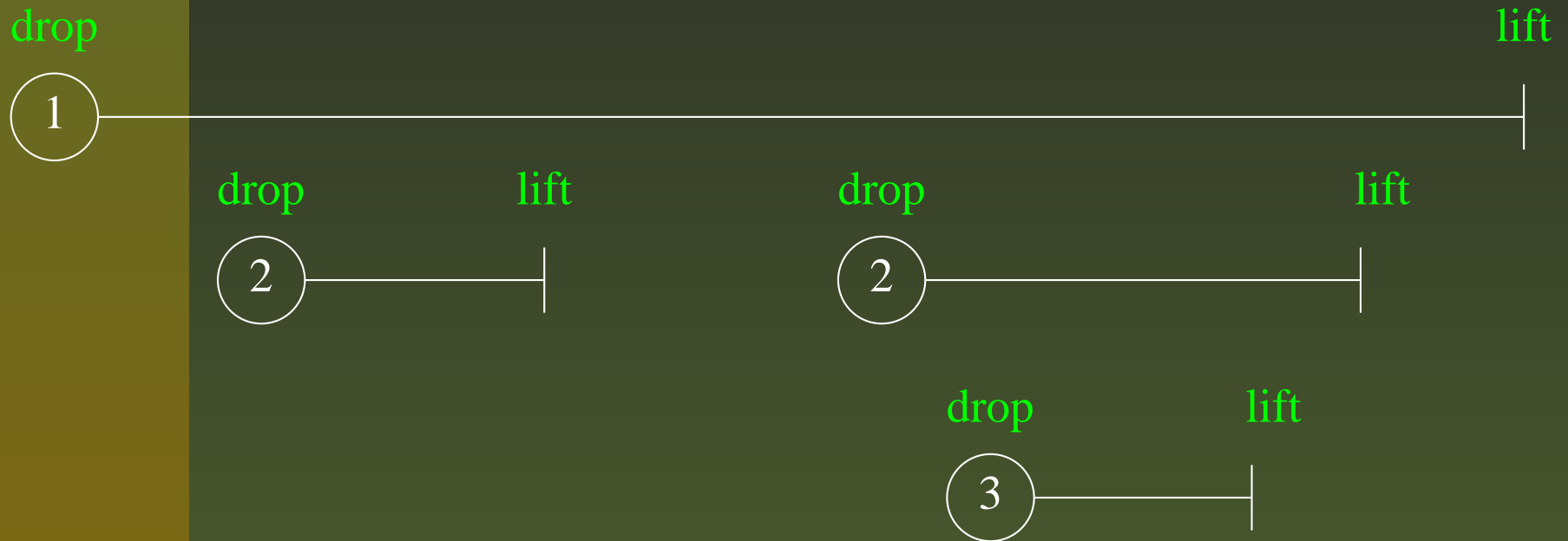
Theoretical model which

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

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

Nested Pebbles

Lifetimes of pebbles must be **nested**:



If $k \leq n$ pebbles are present then it is possible to

- **drop** (if $k+1 \leq n$)
- **lift** (if $k \geq 1$)

Pebbles in Automata Theory

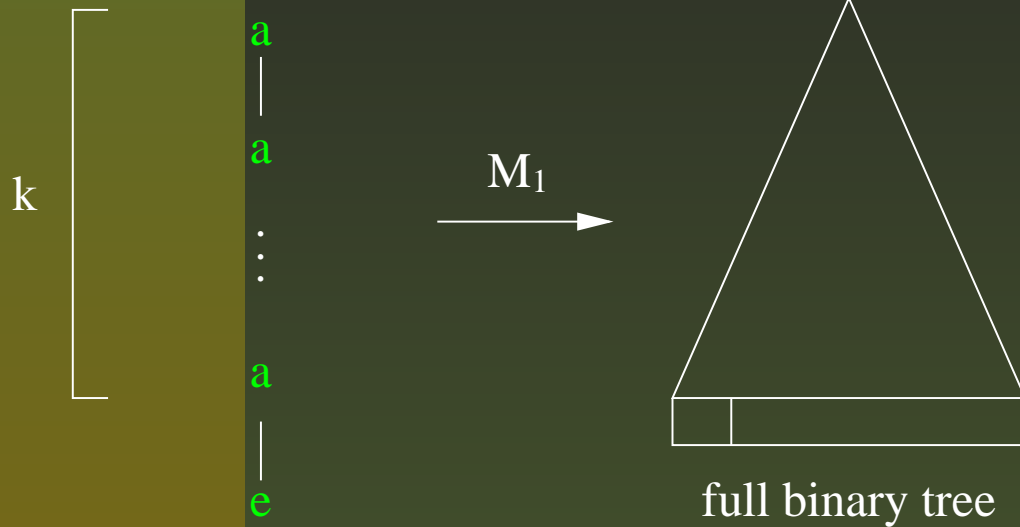
On strings:

- FA + 1 pebble = FA [Blum/Hewitt1967]
- FA + pebbles = LOGSPACE [Hartmanis1972]
- FA + nested pebbles = FA [Globermann/Harel1996]
- FT + nested pebbles [Engelfriet/M 2002]

On Trees:

- FTA + nested pebbles = FTA [Engelfriet/Hoobeboom1999]
- FTT + nested pebbles [Milo/Suciu/Vianu2000]

Examples of 0-PTTs

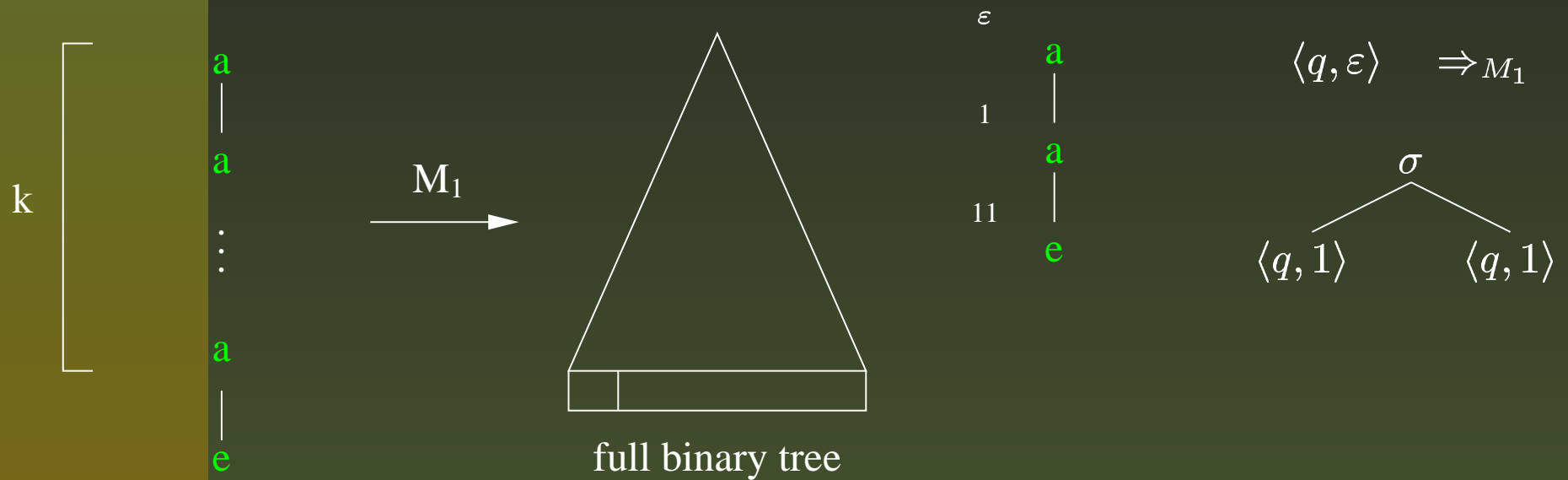


$M_1 :$

$$\langle q, a, \lambda, 0/1 \rangle \rightarrow \sigma(\langle q, \text{down}_1 \rangle, \langle q, \text{down}_1 \rangle)$$

$$\langle q, e, \lambda, 0/1 \rangle \rightarrow e$$

Examples of 0-PTTs

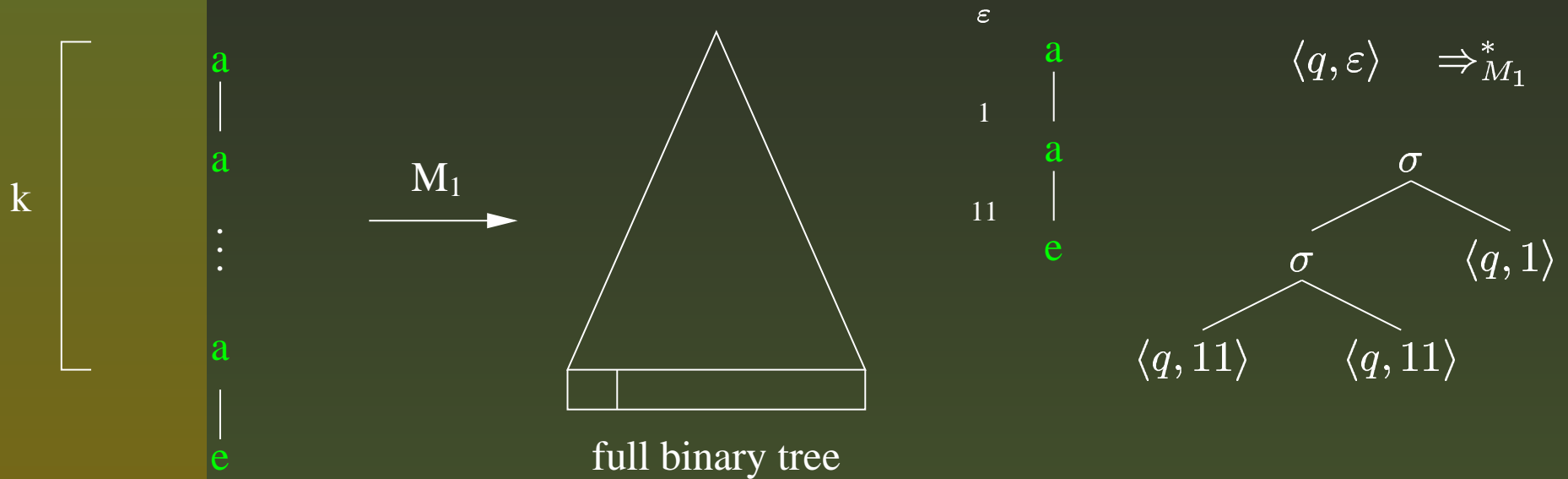


$M_1 :$

$$\langle q, a, \lambda, 0/1 \rangle \rightarrow \sigma(\langle q, \text{down}_1 \rangle, \langle q, \text{down}_1 \rangle)$$

$$\langle q, e, \lambda, 0/1 \rangle \rightarrow e$$

Examples of 0-PTTs

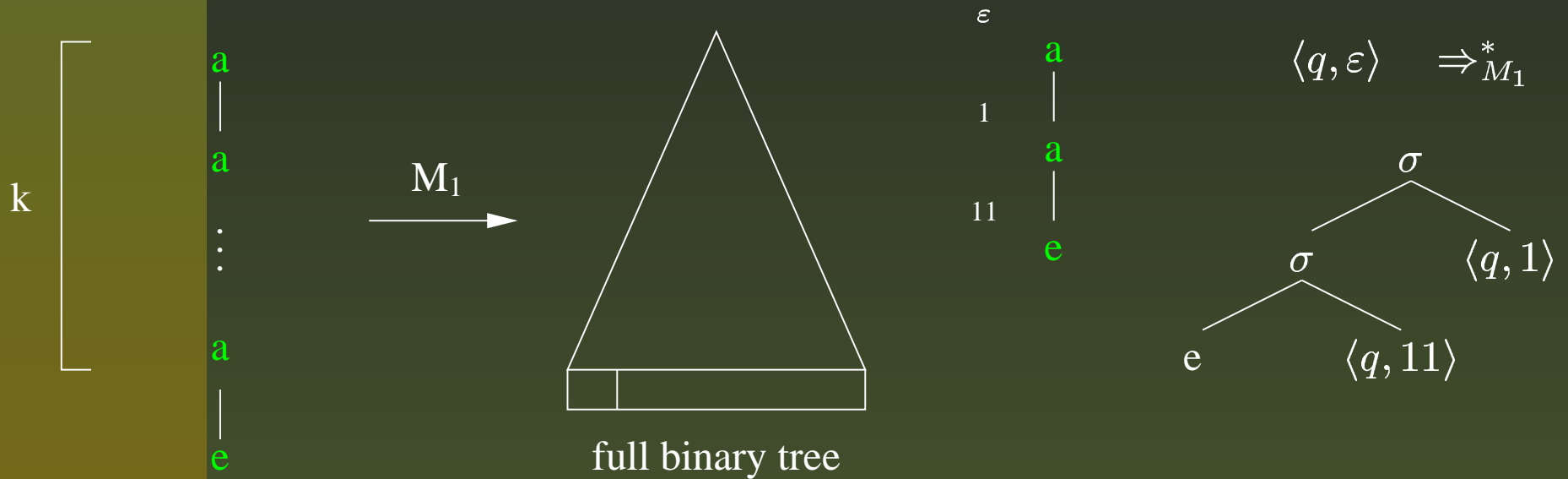


$M_1 :$

$$\langle q, a, \lambda, 0/1 \rangle \rightarrow \sigma(\langle q, \text{down}_1 \rangle, \langle q, \text{down}_1 \rangle)$$

$$\langle q, e, \lambda, 0/1 \rangle \rightarrow e$$

Examples of 0-PTTs

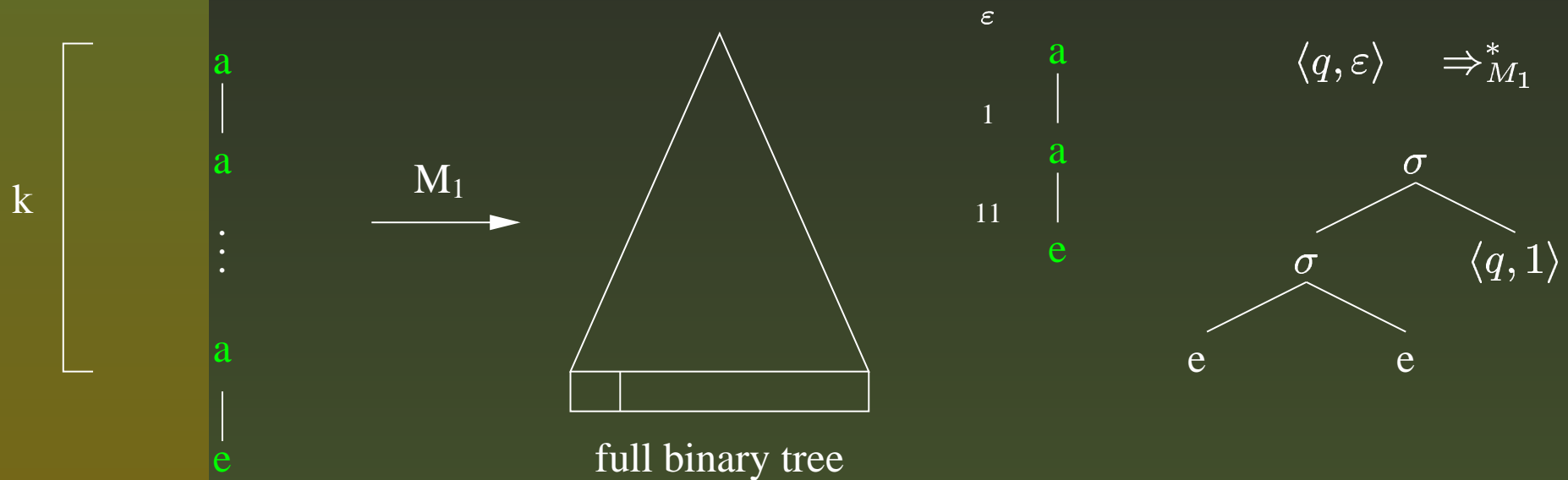


$M_1 :$

$$\langle q, a, \lambda, 0/1 \rangle \rightarrow \sigma(\langle q, \text{down}_1 \rangle, \langle q, \text{down}_1 \rangle)$$

$$\langle q, e, \lambda, 0/1 \rangle \rightarrow e$$

Examples of 0-PTTs

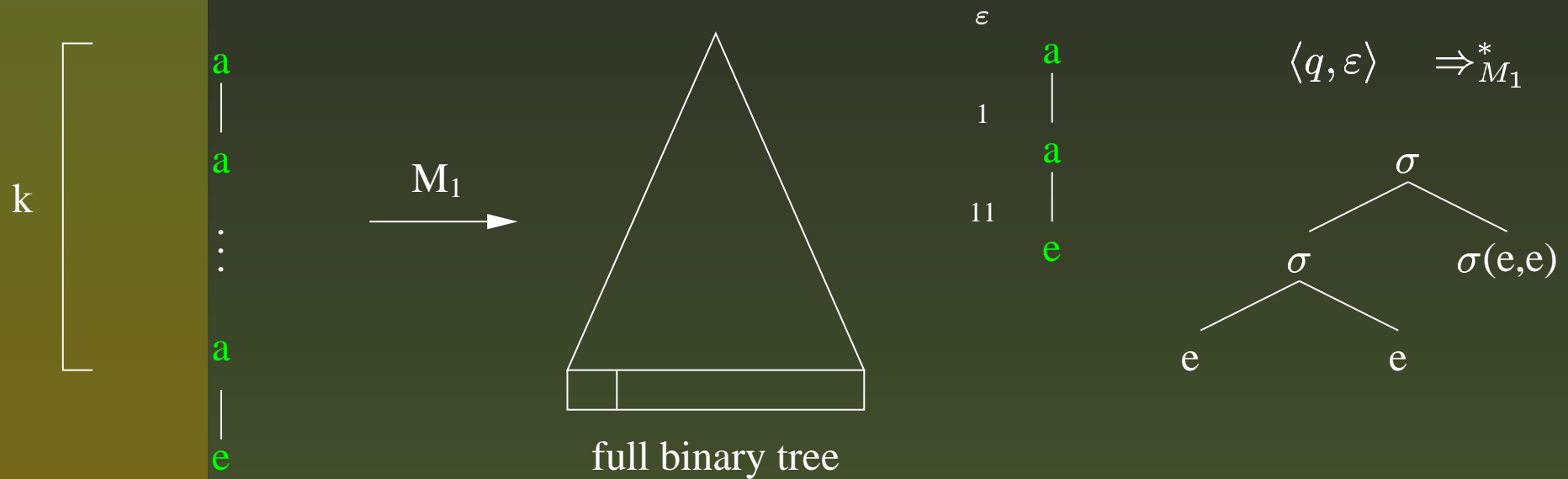


$M_1 :$

$$\langle q, a, \lambda, 0/1 \rangle \rightarrow \sigma(\langle q, \text{down}_1 \rangle, \langle q, \text{down}_1 \rangle)$$

$$\langle q, e, \lambda, 0/1 \rangle \rightarrow e$$

Examples of 0-PTTs

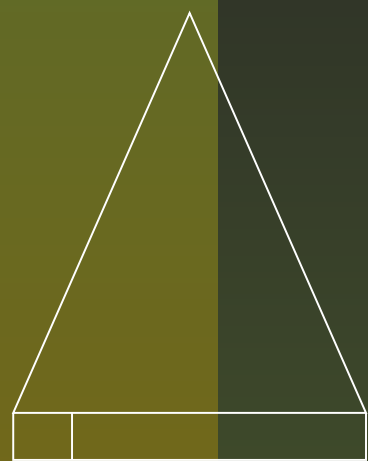


$M_1 :$

$$\langle q, a, \lambda, 0/1 \rangle \rightarrow \sigma(\langle q, \text{down}_1 \rangle, \langle q, \text{down}_1 \rangle)$$

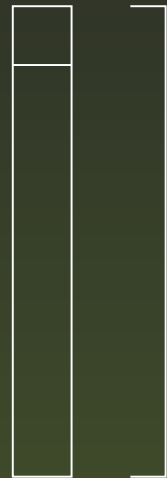
$$\langle q, e, \lambda, 0/1 \rangle \rightarrow e$$

Examples of 0-PTTs

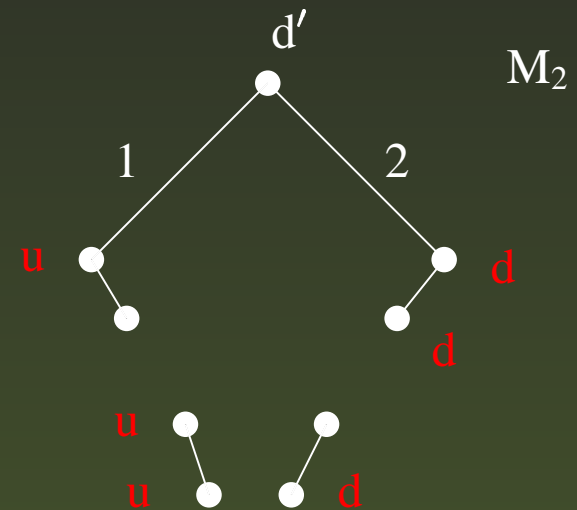


binary tree

M_2 →



“yield”



M_2 : Initial State: **d** (own)

In state **d**

go to leftmost leaf

output 'a' and change to state **u**

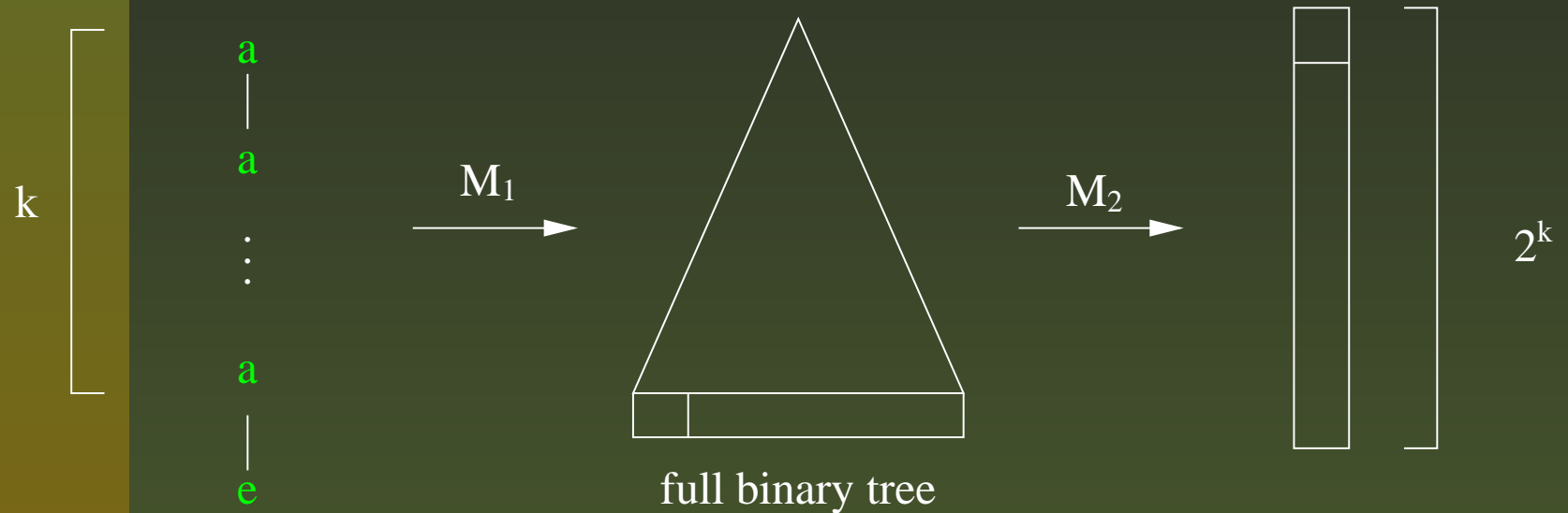
in state **u**

go “up-2” as long as possible

if not-root go up to **d'** and down to **d**

Non-Closure under Composition

Example: Non-closure under composition



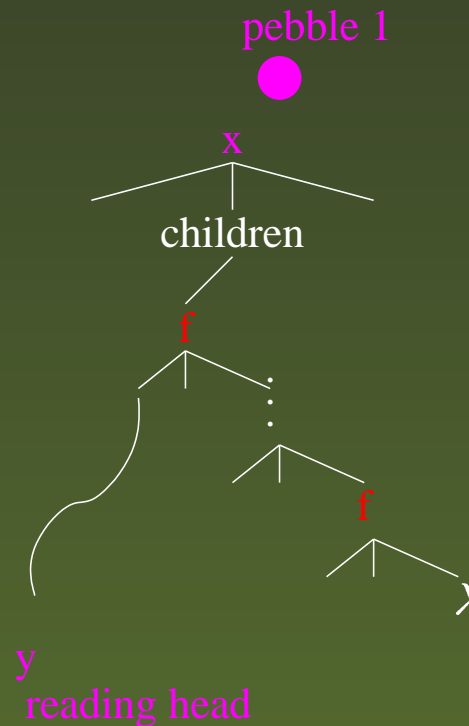
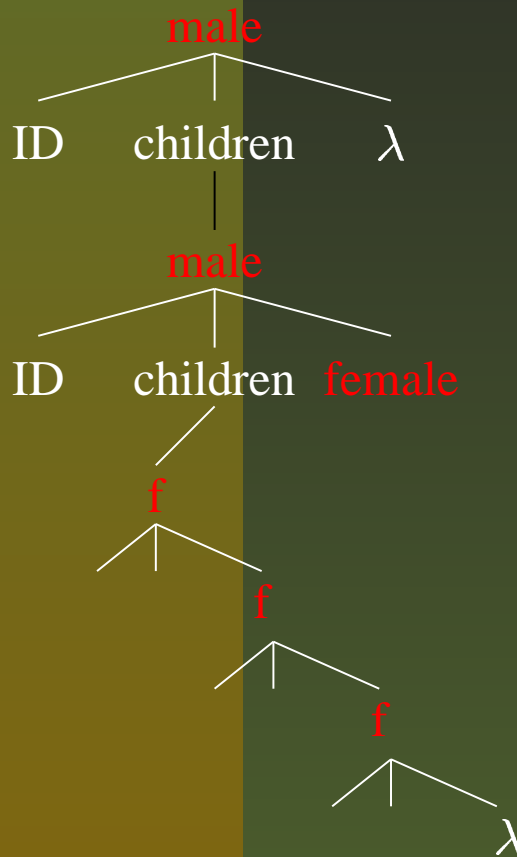
$\tau = M_1 \circ M_2 \in \text{EXPSIZE} \Rightarrow \tau \notin \text{PTT!}$

Example of a 1-PTT

Look for:

x: has only daughters

y: male descendant of **x**



Result 1: Decomposition

Theorem: For all $n \geq 1$,
 $n\text{-PTT} \subseteq 0\text{-PTT}^n$ and $n\text{-DPTT} \subseteq 0\text{-DPTT}^n$.

How does the proof go:

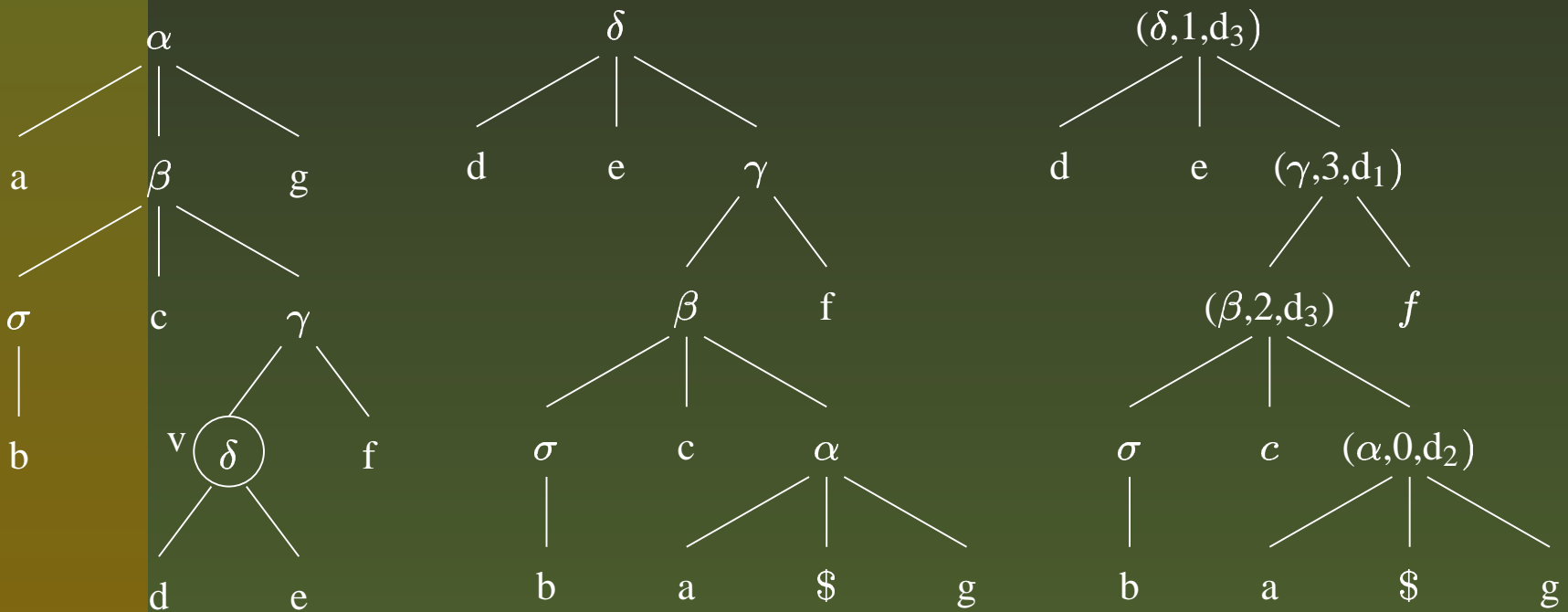
Lemma: For all $n \geq 1$,

- $n\text{-PTT} \subseteq 0\text{-DPTT} \circ (n-1)\text{-PTT}$ and
- $n\text{-DPTT} \subseteq 0\text{-DPTT} \circ (n-1)\text{-DPTT}$.

The 0-DPTT realizes the fixed translation **EncPeb**.

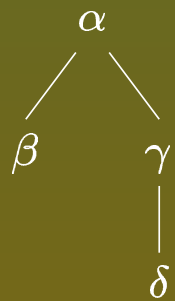
Result 1: Decomposition

$$d_i = \{(\text{up}, \text{down}_i), (\text{down}_i, \text{up})\}.$$

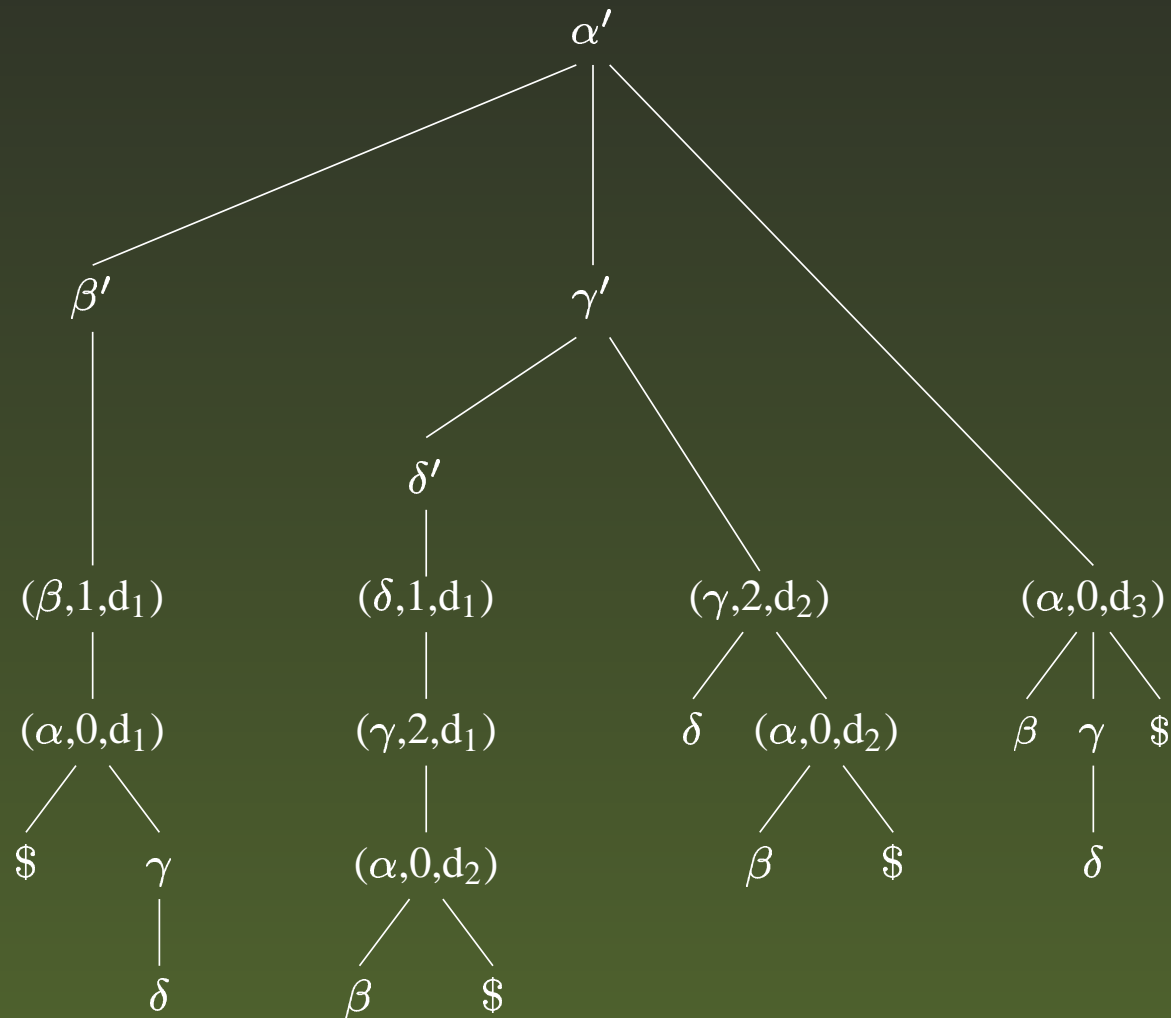


The trees S , S_v , and S_v^{dir} .

Result 1: Decomposition



EncPeb



Result 2: Simulation of PTT by MTT

Eliminating up-moves by **introducing parameters**:

Lemma: $0\text{-PTT} \subseteq 0\text{-PMTT}_{\text{no-up}} = \text{sMTT}$.

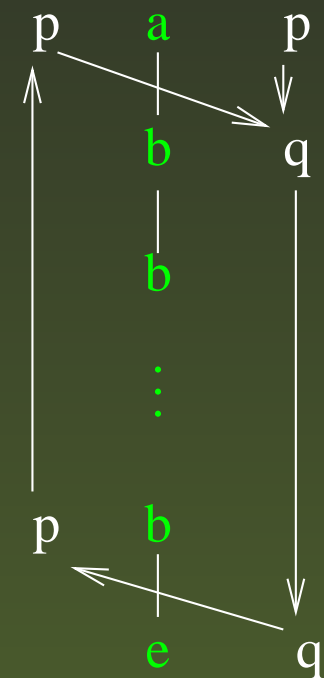
$$\langle (p,0), \mathbf{a}, \lambda, 0 \rangle \rightarrow \langle (q,2), \text{down}_1 \rangle (\langle (q,0), \text{stay} \rangle, \langle (p,0), \text{stay} \rangle)$$

$$\langle (q,2), \mathbf{b}, \lambda, 0 \rangle (y_1, y_2) \rightarrow \langle (q,2), \text{down}_1 \rangle (\langle (q,2), \text{stay} \rangle (y_1, y_2), \langle (p,2), \text{stay} \rangle (y_1, y_2)) = \zeta$$

$$\langle (q,2), \mathbf{e}, \lambda, 0 \rangle (y_1, y_2) \rightarrow y_2$$

$$\langle (p,2), \mathbf{b}, \lambda, 0 \rangle (y_1, y_2) \rightarrow y_2$$

$$\langle (p,2), \mathbf{l}, \lambda, 0 \rangle (y_1, y_2) \rightarrow \zeta$$



Result 2: Simulation of PTT by MTT

Eliminating up-moves by **introducing parameters**:

Lemma: $0\text{-PTT} \subseteq 0\text{-PMTT}_{\text{no-up}} = \text{sMTT}$.

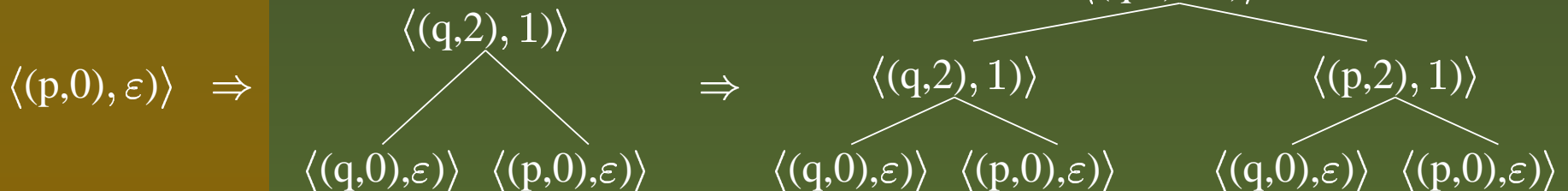
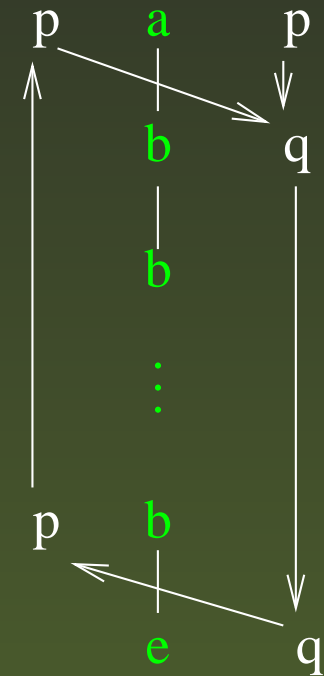
$$\langle (p,0), a, \lambda, 0 \rangle \rightarrow \langle (q,2), \text{down}_1 \rangle (\langle (q,0), \text{stay} \rangle, \langle (p,0), \text{stay} \rangle)$$

$$\langle (q,2), b, \lambda, 0 \rangle (y_1, y_2) \rightarrow \langle (q,2), \text{down}_1 \rangle (\langle (q,2), \text{stay} \rangle (y_1, y_2), \langle (p,2), \text{stay} \rangle (y_1, y_2)) = \zeta$$

$$\langle (q,2), e, \lambda, 0 \rangle (y_1, y_2) \rightarrow y_2$$

$$\langle (p,2), b, \lambda, 0 \rangle (y_1, y_2) \rightarrow y_2$$

$$\langle (p,2), l, \lambda, 0 \rangle (y_1, y_2) \rightarrow \zeta$$



Result 2: Simulation of PTT by MTT

Only possible for Determinism:

Eliminating stay-moves (and **infinite** computations!)

Theorem: $D_sMTT = DMTT$.

Proof Idea:

Apply rules to stay moves in the right-hand sides, keeping track of possible **cycles**.

technical complication: lazy evaluation... (OI)

3 Query Power: $\text{PTT}^* = \text{MTT}^*$

Theorem: For all $n \geq 1$,
 $n\text{-PTT} \subseteq \text{sMTT}^{n+1}$ and $n\text{-DPTT} \subseteq \text{DMTT}^{n+1}$

(the first MTTs n realize EncPeb)

Other Direction:

Lemma: $\text{sMTT} \subseteq 0\text{-PTT}^4$ and $\text{DMTT} \subseteq 0\text{-DPTT}^3$.

Theorem: The queries languages of PTT and MTT coincide:

- $\text{PTT}^* = \text{sMTT}^*$ and
- $\text{DPTT}^* = \text{DMTT}^*$.

Applications

Three important properties of compositions τ of MTTs:

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

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

Applications of I: Type Checking

Type Checking Problem

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

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 $R_{\text{in}}, R_{\text{out}} \in \text{REGT}$, translation τ

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

Decidable: (using **I**: τ^{-1} preserves REGT)

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

and inclusion of REGT is decidable.

Almost Always Type Checking I+II

Almost Always Type Checking Problem

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

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 τ

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.

Conclusions

I – III hold for (compositions of) DPTTs!!

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])

Further Research

New Result: Collapse of DMTT hierarchy, for Linear Size Increase functions (LSIZE)

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

$= MSOTT$ [Engelfriet/M2001]

$\Rightarrow MSOTT$ is a **decidable** subclass of MTT^* !!

Further Research

- Investigate pebble MACRO tree transducer (PMTTs)
- Characterize DPTT inside of $\text{DPTT}^* = \text{MTT}^*$.

Conjecture: $\text{DPTT} = \text{DPTT}^* \cap \text{POLYSUB}$

POLYSUB: #SUBtrees in output is POLYnomial in size of input.

Further Research

- Investigate pebble MACRO tree transducer (PMTTs)
- Characterize DPTT inside of $DPTT^* = MTT^*$.

Conjecture: $DPTT = DPTT^* \cap POLYSUB$

POLYSUB: #SUBtrees in output is POLYnomial in size of input.

THE END