Higher-Order Tree Transducers and Their Expressive Power

Kazuhiro Inaba

at Dagstuhl Seminar, May 2013

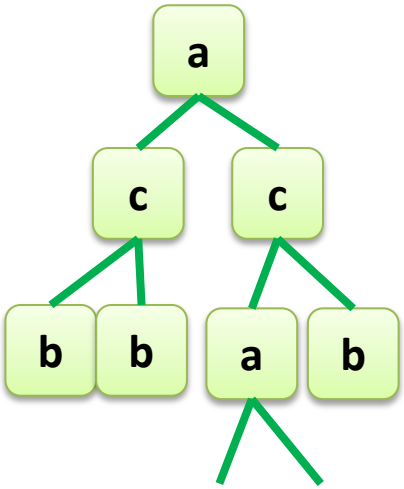
Regular Tree Grammar

input tree

(Top-Down) Tree Transducer

- Δ = {a, b, c, ...}
- N = {F, G, H, ..., S, T, ...}
- R = set of rules
- S ∈ N

$S \rightarrow a\ T\ T$
$S \rightarrow b$
$T \rightarrow c\ S\ S$



- Σ = {s, t, u, ...}
- Δ = {a, b, c, ...}
- N = {F, G, H, ..., S, T, ...}
- R = set of rules
- S ∈ N

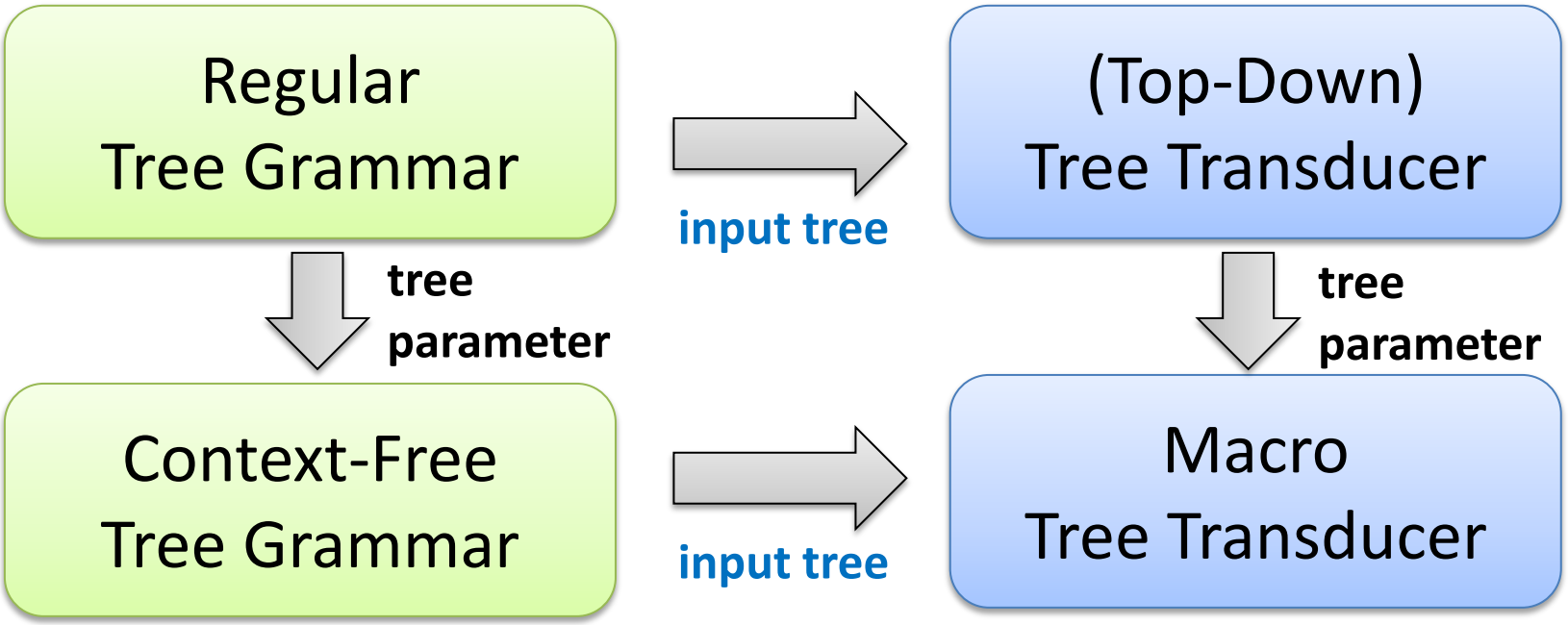
$S\ (s\ x_1\ x_2) \rightarrow a\ (T\ x_1)\ (T\ x_2)$
$S\ (t) \rightarrow b$
$T\ (s\ x_1\ x_2) \rightarrow c\ (S\ x_1)\ (S\ x_2)$

*Each nonterminal generates (a set of) **trees**.*

$F : O$

*Each nonterminal takes an **input tree** and generates (a set of) **trees**.*

$F : I \rightarrow O$



$S \rightarrow T\ d\ d$
$T\ y_1\ y_2 \rightarrow T\ (b\ y_1)\ (c\ y_2)$
$T\ y_1\ y_2 \rightarrow a\ y_1\ y_2$

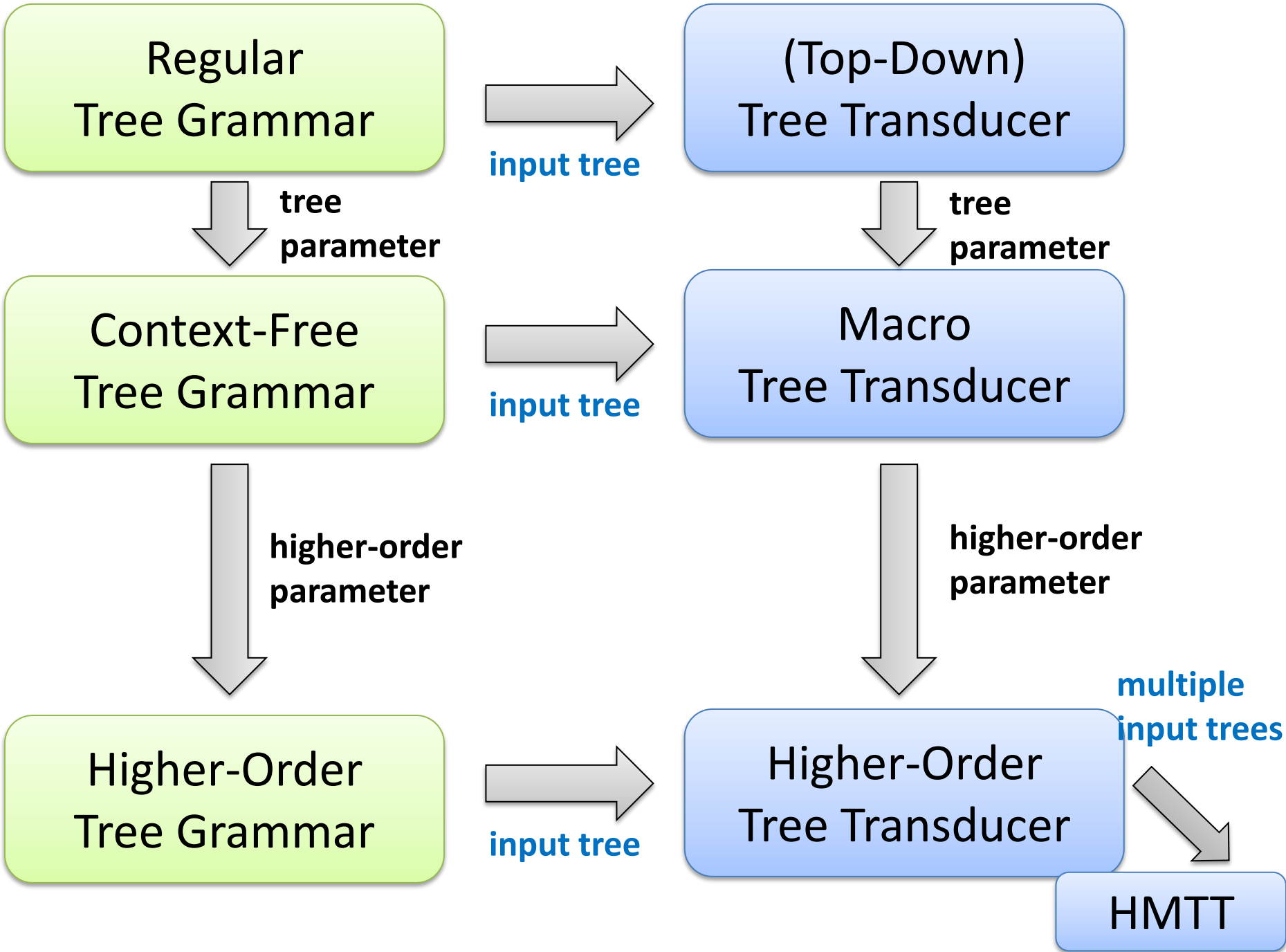
$S\ (s\ x_1) \rightarrow T\ x_1\ d\ d$
$T\ (s\ x_1)\ y_1\ y_2 \rightarrow T\ x_1\ (b\ y_1)\ (c\ y_2)$
$T\ z\ y_1\ y_2 \rightarrow a\ y_1\ y_2$

*Each nonterminal takes parameter **trees** and generates (a set of) **trees**.*

$F : O^k \rightarrow O$

*Each nonterminal takes an **input tree** and parameter **trees**, and generates **trees**.*

$F : I \rightarrow O^k \rightarrow O$


Regular
Tree Grammar
input tree
(Top-Down)
Tree Transducer
tree
parameter
tree
parameter
Context-Free
Tree Grammar
input tree
Macro
Tree Transducer
higher-order
parameter
higher-order
parameter
multiple
input trees
Higher-Order
Tree Grammar
input tree
Higher-Order
Tree Transducer
HMTT

Example of a higher-order transducer

Mult : I → O

Mult (**pair** x_1 x_2) → Iter x_1 (Add x_2) **z**

Iter : I → (O → O) → O → O

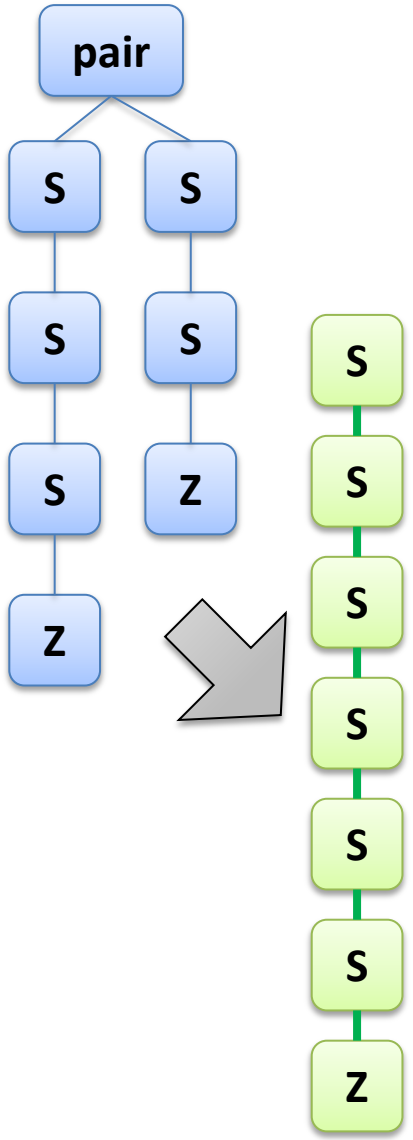
Iter (**s** x_1) f y → Iter x_1 f (f y)

Iter **z** f y → y

Add : I → O → O

Add (**s** x_1) y → Add x_1 (**s** y)

Add **z** y → y



(Examples of) problems we should be interested in

Membership

Given a higher-order grammar **G** and a tree **t**, decide whether $t \in [G]$ or not.

Type Checking

Given a higher-order tree transducer **f** and regular tree languages **S** and **T**, decide whether $f(S) \subseteq T$ or not.

Model Checking

Given a deterministic higher-order grammar **G** representing a (possibly infinite) single tree **t**, and a MSO sentence ϕ, decide whether **t** satisfies ϕ.

Equi-Expressivity

What is the automata-like of the models?
Can they be “decomposed” to simpler models?

Agenda

- Introduction
- Two notions of “higher-order” types.
- Review of known results.
- Context-sensitiveness of “safe” higher-order languages [I. and Maneth, 2008]

Two Notions of "Higher Order" Types (1)

$D_0 = \mathbf{O}$

"Trees" are order-0.

$D_{i+1} = \{ D_i^k \rightarrow D_i \mid k \in N \}$

Functions from order-i objects to order-i objects are order-(i+1).

$order(t) = i \quad \text{if } t \in D_i$

- "Derived Types"
  - OI-Hierarchy [Damm 82]
  - High-Level Tree Tranducer [Engelfriet & Vogler 88]

Two Notions of "Higher Order" Types (2)

- Recently actively studied in context of program verification [Ong 06, ...] or linguistics [Kobele&Salvati 13, ...].

D ::= **O** | D→D

"Trees" are order-0, and ...

order(**O**) = 0

order($t_1 \rightarrow t_2$) =

max(order(t_1)+1, order(t_2))

The Difference

- Functions parameters of "Derived Types" have decreasing order

$$D_n \rightarrow (D_{n-1} \rightarrow (D_{n-2} \rightarrow \ldots (o \rightarrow o) \ldots))$$

which does not contain, e.g.,

$$\lambda x.\ \lambda f.\ \lambda y.\ f\ x\ : o \rightarrow (o \rightarrow o) \rightarrow o \rightarrow o$$

It implies:

Safety [Knapik&Niwinski&Urzyczyn 01, 02]

No order-k subterm can contain order <k free variables.

Safety

Safety [KNU 01, 02]

No order-k subterm can contain order <k free variables.

[KNU 01, 02] [Blum&Ong 09]

In ***safe*** grammars/λ-calculus, **you don't need to care about variable capturing** while substitution.

Unsafe example: λy. ((λx. λy. **a** x y) y)

!!

➔ λy. ((λy. **a** x y)[x/y])

➔ λy. (λy. **a** y y) this is wrong

"Safe" :: $D_{i+1} = \{D_i^k \to D_i\}$	"Unsafe" :: $D \to D$
Grammars	
• MSO model checking is decidable. [KNU 01, 02]	• MSO model checking is decidable. [Ong 06, Kobayashi 09]
• Hierarchy is strict. [Damm 82]	• Hierarchy is strict. [Kartzow&Parys 12]
• Equivalent to "iterated pushdown automata" [Da 82] (= (stack of)* stacks)	• Equivalent to "*collapsible* pushdown automata" [Hague&Murawski&Ong&Serre 08]
• Context-sensitive. [Maneth 02][I.&Maneth 08]	• ????
Transducers [EV88]	
• n-DHTT = (1-DHTT) ^{n}	
• n-NHTT $\subseteq$ (1-NHTT) ^{n}	• ????

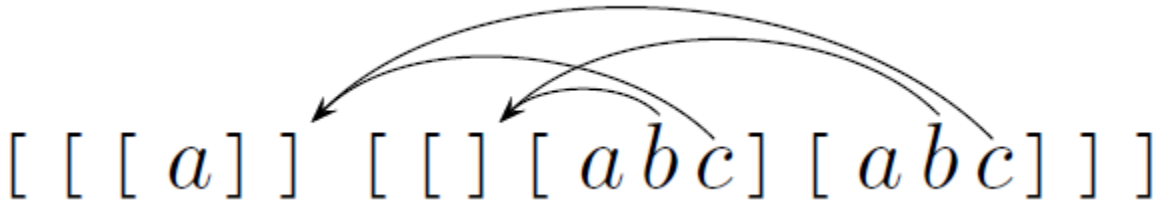
"Safe" :: $D_{i+1} = \{D_i^k \rightarrow D_i\}$	"Unsafe" :: $D \rightarrow D$
Grammars	
• MSO model checking is decidable. [KNU 01, 02]	• MSO model checking is decidable. [Ong 06, Kobayashi 09]
• Hierarchy is strict. [Damm 82]	• Hierarchy is strict. [Kartzow&Parys 12]
• Equivalent to "iterated pushdown automata" [Da 82] (= (stack of)* stacks)	• Equivalent to "*collapsible* pushdown automata" [Hague&Murawski&Ong&Serre 08]
• Context-sensitive. [Maneth 02][I.&Maneth 08]	• ????
Transducers [EV88]	
• n-DHTT = (1-DHTT) ^{n}	
• n-NHTT $\subseteq$ (1-NHTT) ^{n}	• ????

"Safe" :: $D_{i+1} = \{D_i^k \rightarrow D_i\}$	"Unsafe" :: $D \rightarrow D$
Grammars	
• MSO model checking is decidable. [KNU 01, 02]	• MSO model checking is decidable. [Ong 06, Kobayashi 09]
• Hierarchy is strict. [Damm 82]	• Hierarchy is strict. [Kartzow&Parys 12]
• Equivalent to "iterated pushdown automata" [Da 82] (= (stack of)* stacks)	• Equivalent to "*collapsible* pushdown automata" [Hague&Murawski&Ong&Serre 08]
• Context-sensitive. [Maneth 02][I.&Maneth 08]	• ????
Transducers [EV88]	
• n-DHTT = (1-DHTT) ^{n}	
• n-NHTT $\subseteq$ (1-NHTT) ^{n}	• ????

“Collapsible” Pushdown Automata

[Hague at al. 08]

- Order-n collapsible pushdown store is
  - (stack of)^n symbols
  - with each symbol associated with “links”



- Push_1 : pushes a symbol and link to the top.
- Dup_k : duplicates the top order-k stack.
- Pop_k : pops the top order-k stack.
- Collapse : moves the top to the pointee of the top link.

"Safe" :: $D_{i+1} = \{D_i^k \to D_i\}$	"Unsafe" :: $D \to D$

Grammars

- MSO model checking is decidable. [KNU 01, 02]
- Hierarchy is strict. [Damm 82]
- Equivalent to "iterated pushdown automata" [Da 82]
  (= (stack of)* stacks)
- Context-sensitive. [Maneth 02][I.&Maneth 08]

- MSO model checking is decidable. [Ong 06, Kobayashi 09]
- Hierarchy is strict. [Kartzow&Parys 12]
- Equivalent to "*collapsible* pushdown automata" [Hague&Murawski&Ong&Serre 08]
- ????

2-unsafe = 2-safe [Aehlig&Miranda&Ong 05]
2-unsafe-det $\not\subset$ n-safe-det [Parys 11, 12]

Transducers [EV88]

- n-DHTT = (1-DHTT) ^{n}
- n-NHTT $\subseteq$ (1-NHTT) ^{n}

- ????

"Safe" :: $D_{i+1} = \{D_i^k \to D_i\}$	"Unsafe" :: $D \to D$
Grammars	
• MSO model checking is decidable. [KNU 01, 02]	• MSO model checking is decidable. [Ong 06, Kobayashi 09]
• Hierarchy is strict. [Damm 82]	• Hierarchy is strict. [Kartzow&Parys 12]
• Equivalent to "iterated pushdown automata" [Da 82] (= (stack of)* stacks)	• Equivalent to "*collapsible* pushdown automata" [Hague&Murawski&Ong&Serre 08]
• Context-sensitive. [I.&Maneth 08]	• ????
Transducers [EV88]	
• n-DHTT = (1-DHTT) ^{n}	
• n-NHTT $\subseteq$ (1-NHTT) ^{n}	• ????

1^{st} order Decomposition of Safe HTT

[Engelfrier&Vogler 86,88] [Caucal 02]

$$\text{Safe-n-DHTT} = (\text{Safe-1-DHTT})^n$$

$$\text{Safe-n-NHTT} \subseteq (\text{Safe-1-NHTT})^n$$

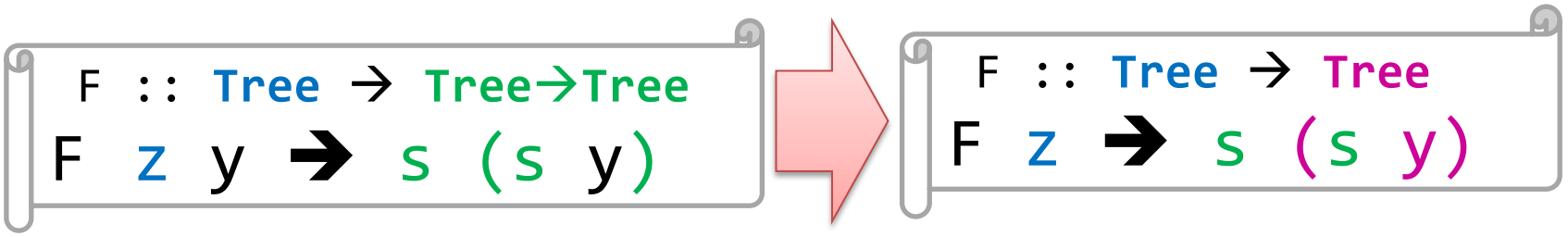
n-th order tree transducer is representable by a n-fold composition of 1^{st}-order tree transducers.

Note: Higher order grammars can be simulated by Out(HTT).

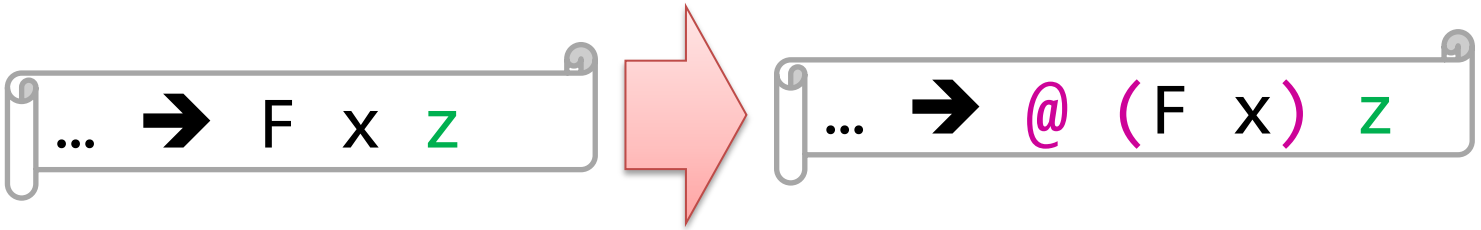
Proof: n-HTT = (n-1)-HTT ; 1-HTT

Idea:

Represent 1st-order term `Tree→Tree` by a `Tree`.



Represent 1st-order application symbolically, too.



Proof: n-HTT = (n-1)-HTT ; 1-HTT

Represent 1st-order things symbolically.

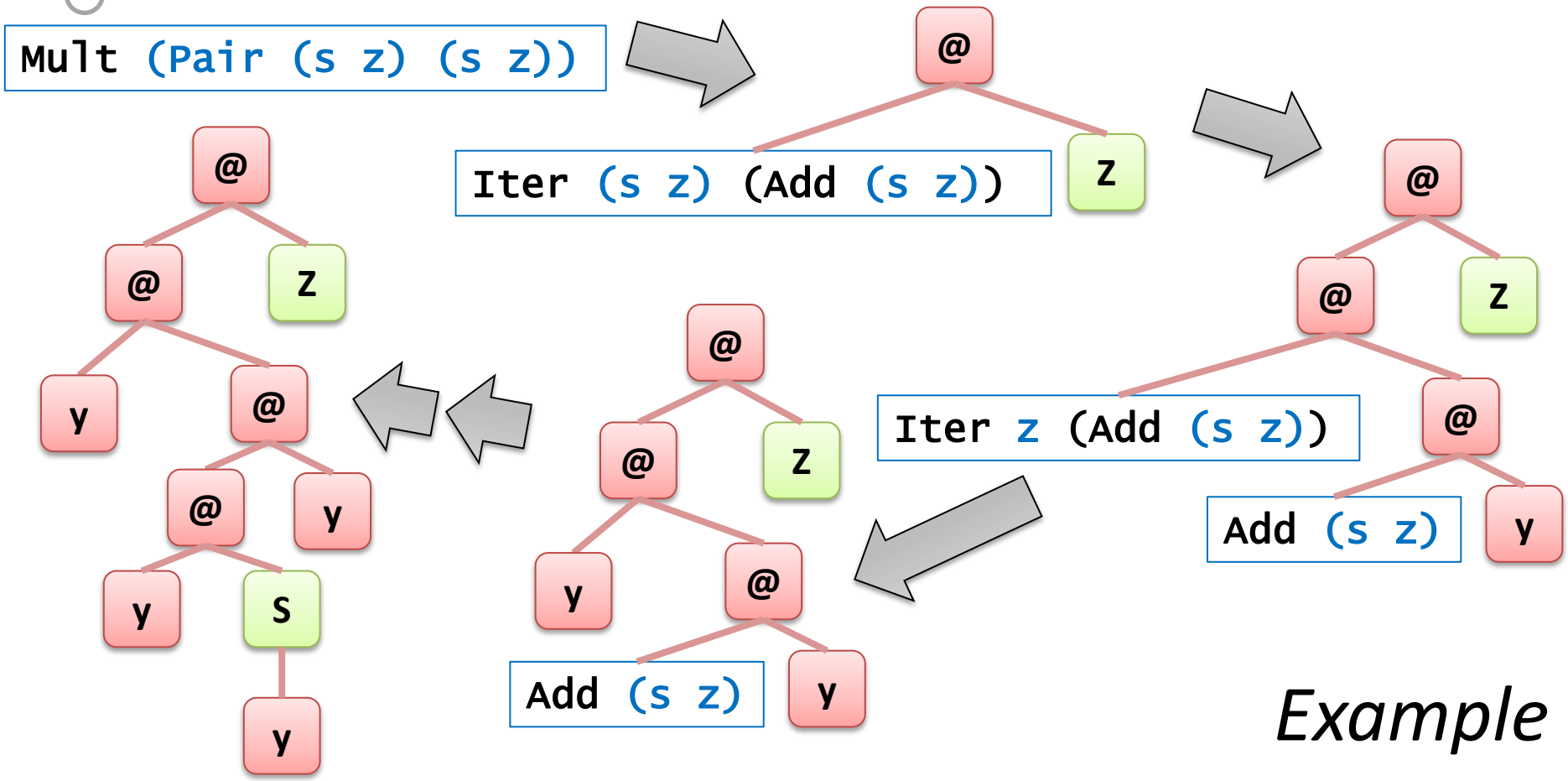
```
F :: Tree → Tree
F z ➔ s (s y)
```

```
… ➔ @ (F x) z
```

Then a 1-HTT performs the actual “application”.

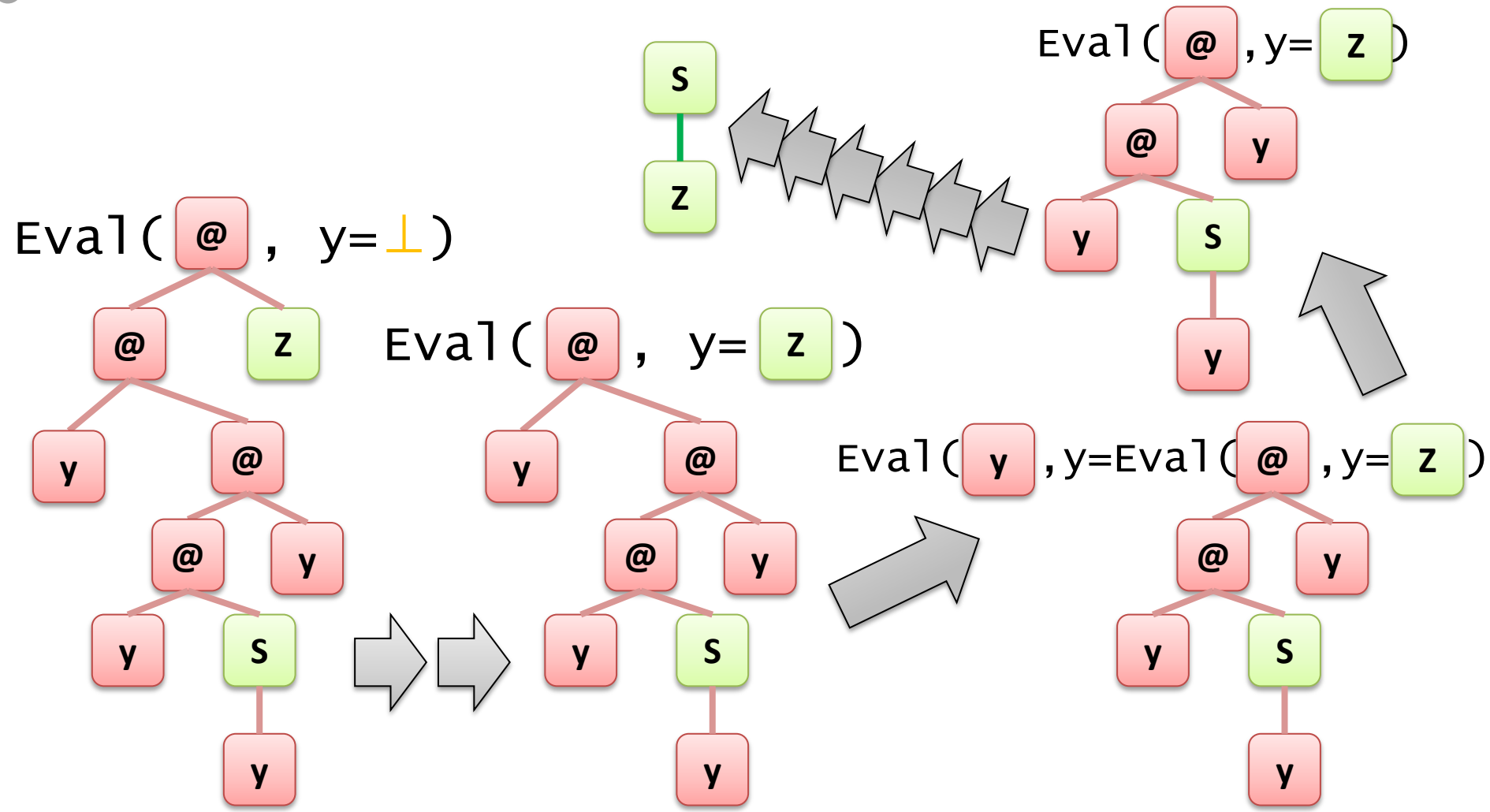
```
Eval (@ f b) y   ➔ Eval f (Eval b y)
Eval y y         ➔ y
Eval (s x) y     ➔ s (Eval x y)
Eval z y         ➔ z
```

```
Mult (pair x1 x2)  ➔ @ (Iter x1 (Add x2)) z
Iter (s x) f       ➔ @ (Iter x f) (@ f y)
Iter z f           ➔ y
Add (s x)          ➔ @ (Add x) (s y)
Add z              ➔ y
```



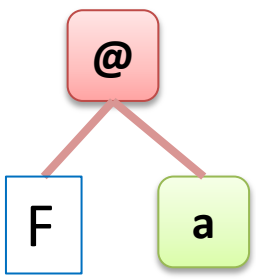
Example

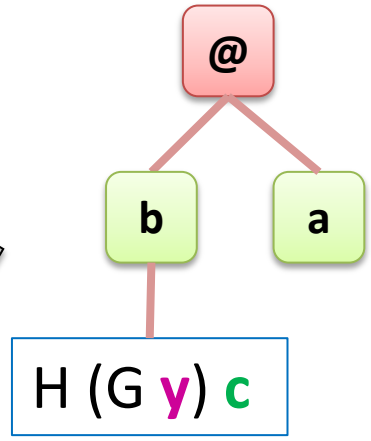
```
Eval (@ f b) y   ➔ Eval f (Eval b y)
Eval y y         ➔ y
Eval (s x) y     ➔ s (Eval x y)
Eval z y         ➔ z
```



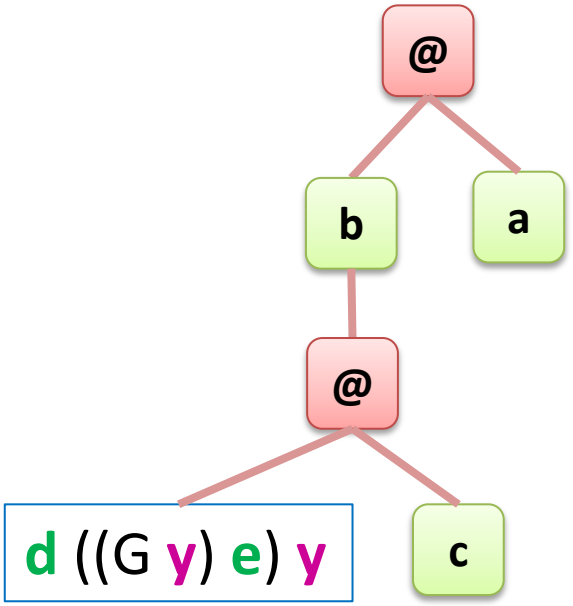
If it's unsafe...

S → F **a**
F y → **b** (H (G y) **c**)
G x y → **...**
H f y → **d** (f **e**) y





S → **@** F **a**
F → **b** (**@** (H (**@** G **y**)) **c**)
G → ..
H f → **d** (**@** f **e**) **y**



Consequences of First-Order Decomposition

[EV88]

$f^{-1}(T) \in \mathrm{REG}$ if $f \in$ Safe-HTT and $T \in \mathrm{REG}$

- Proof: because MTT (1-HTT) has the property.

- This gives decidable "type checking".
  - $f(S) \subseteq T \Leftarrow f^{-1}(T) \subseteq S$: inclusion of REG is decidable.

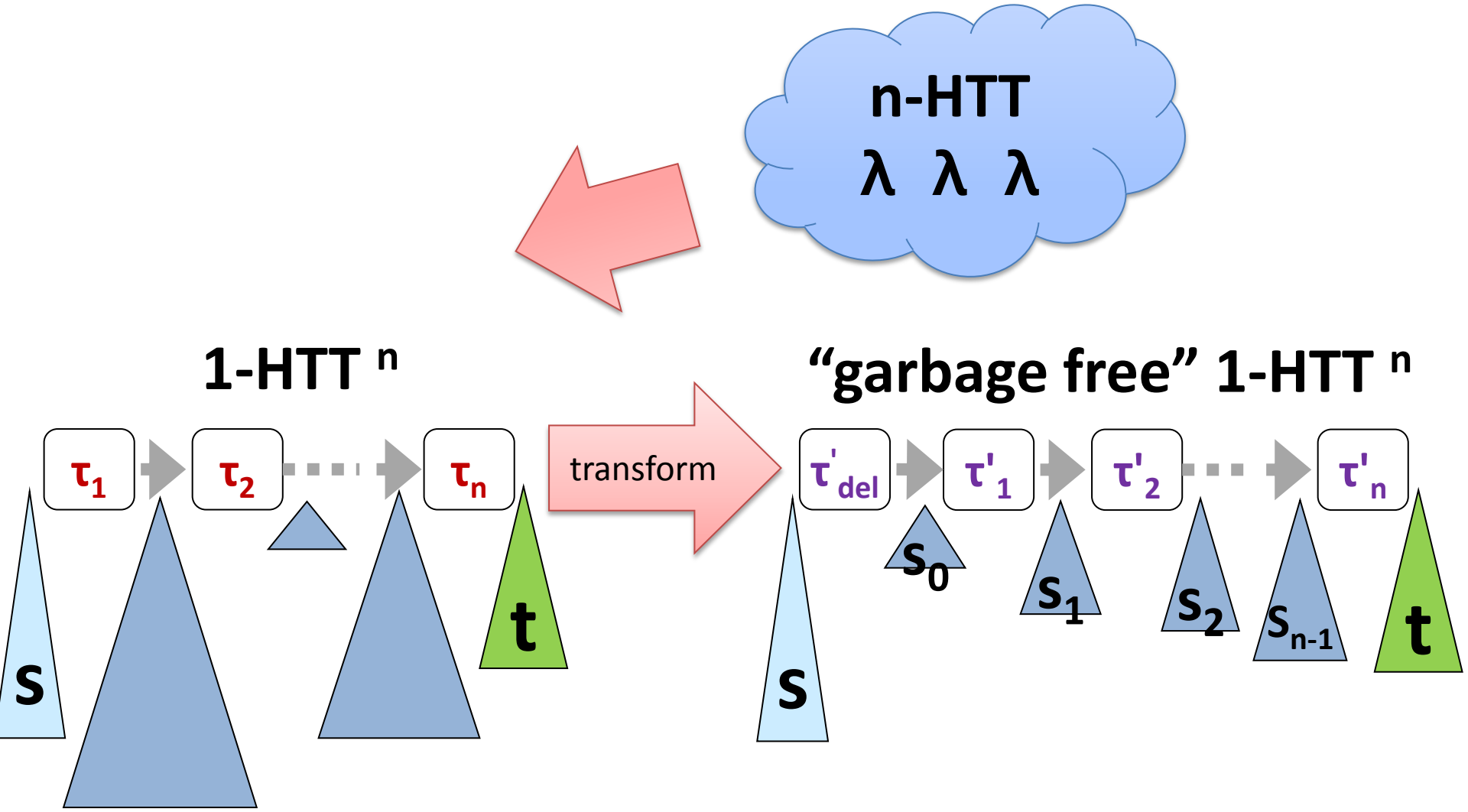
Consequences of First-Order Decomposition

[I. and Maneth 08]

$f(T) \in \text{DCS}$ if $f \in \text{Safe-HTT}$ and $T \in \text{REG}$

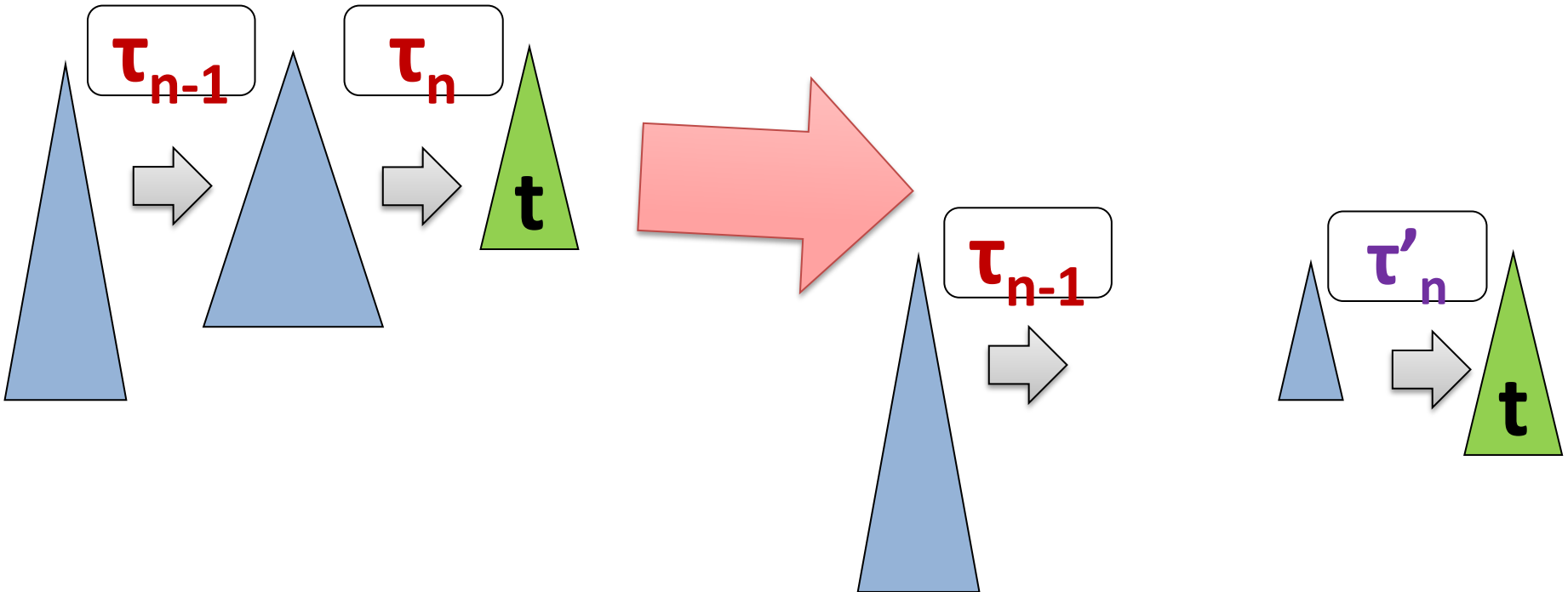
- DCS = Deterministic-Context-Sensitive
  = DLINSPACE membership
- Proof: in a next few slides...
- Corollary : Safe Higher-order languages (aka. OI-Hierarchy) are context-sensitive.

Proof: Out(1-HTT ^{n}) $\in$ DLINSPACE
The Key Idea



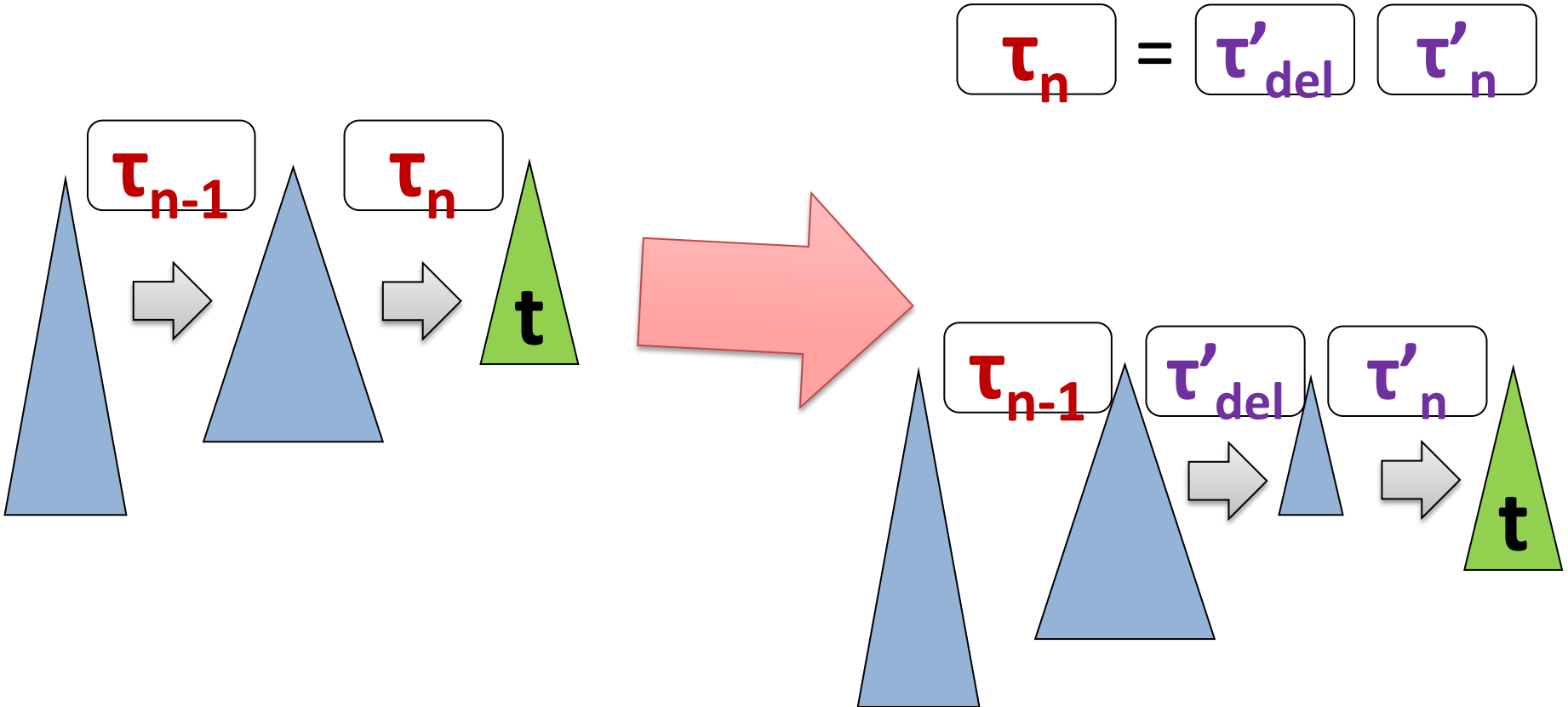
How to Construct the “Garbage-Free” Form

Make each 1-HTT “productive”



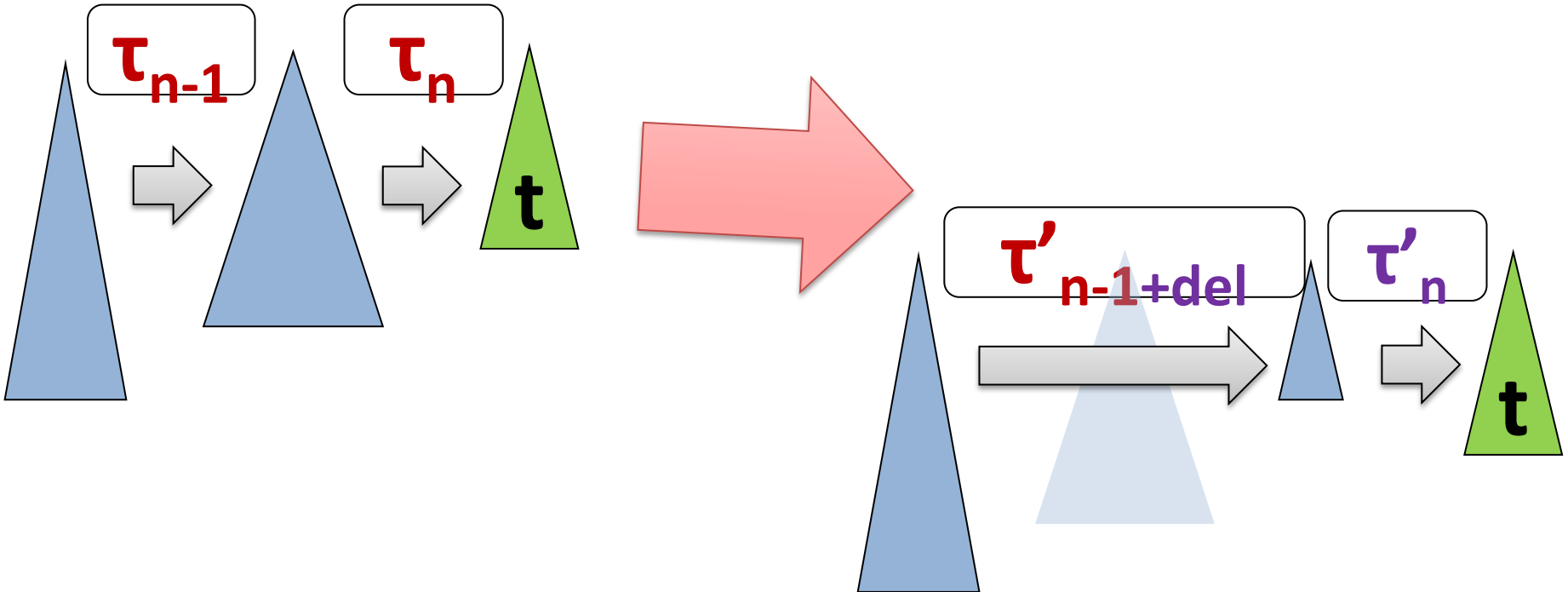
How to Construct the “Garbage-Free” Form

Make each 1-HTT “productive”
by separating its “deleting” part

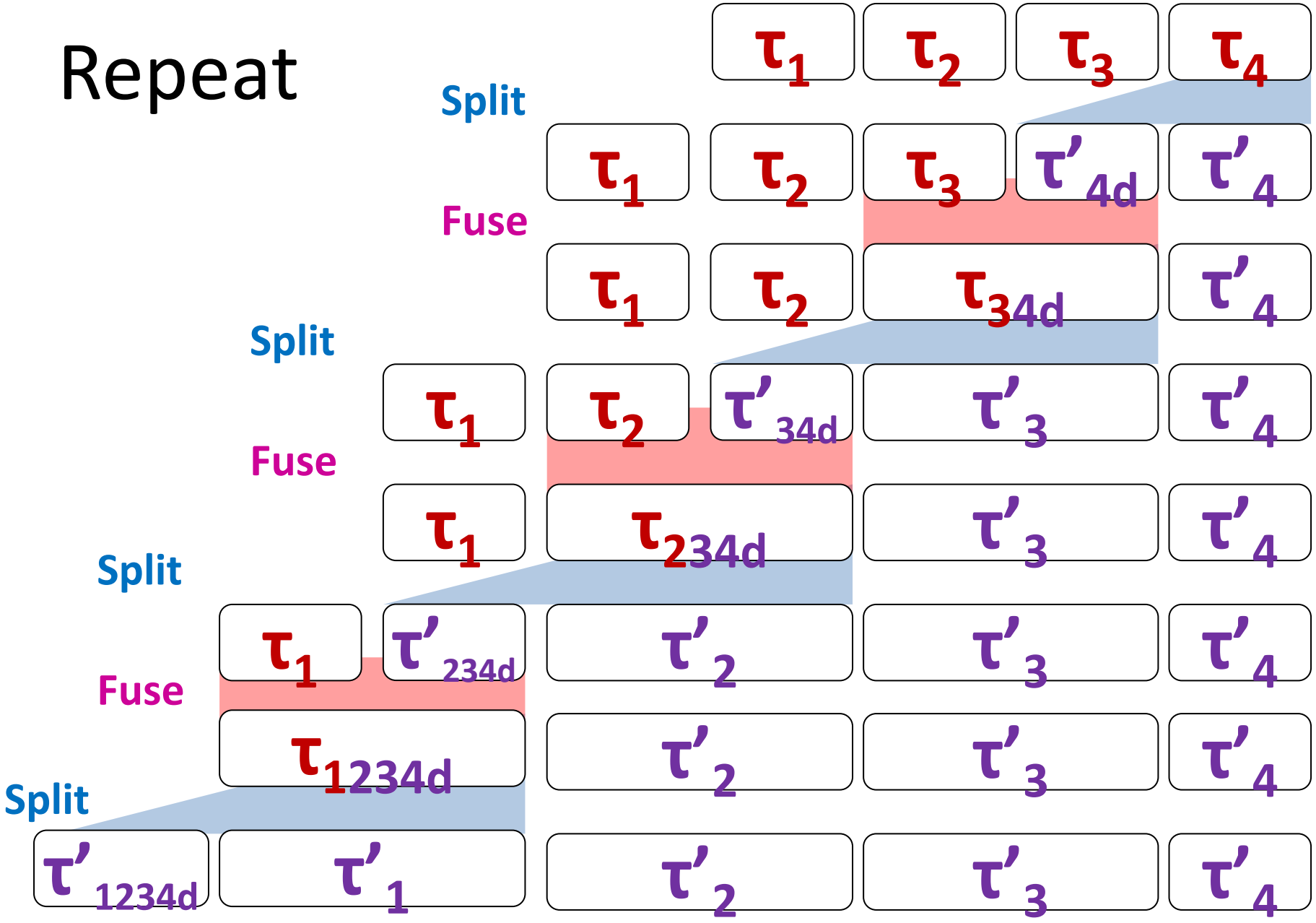


How to Construct the “Garbage-Free” Form

Make each 1-HTT “productive”
by separating its “deleting” part,
and fuse the deleter to the left [En75,77][EnVo85][EnMa02]

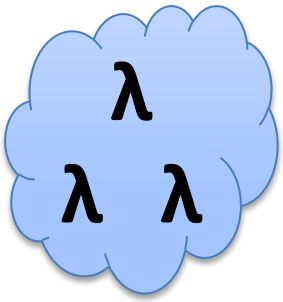


Repeat

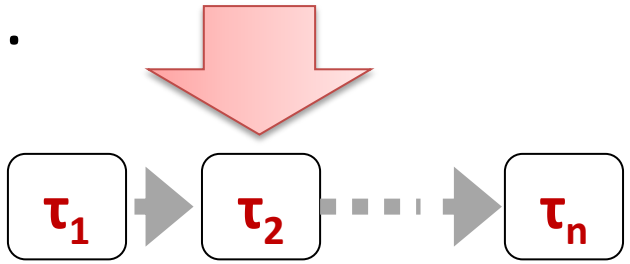

Split
Fuse
Split
Fuse
Split
Fuse
Split
τ1 τ2 τ3 τ4
τ1 τ2 τ3 τ'4d τ'4
τ1 τ2 τ34d τ'4
τ1 τ2 τ'34d τ'3 τ'4
τ1 τ234d τ'3 τ'4
τ1 τ'234d τ'2 τ'3 τ'4
τ1234d τ'2 τ'3 τ'4
τ'1234d τ'1 τ'2 τ'3 τ'4

Summary: Out(Safe-n-HTT) is context sensitive.

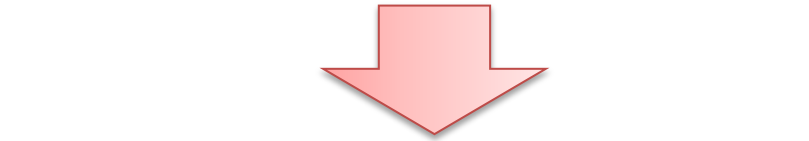
1. Decompose n-HTT to $(1\text{-HTT})^n$.



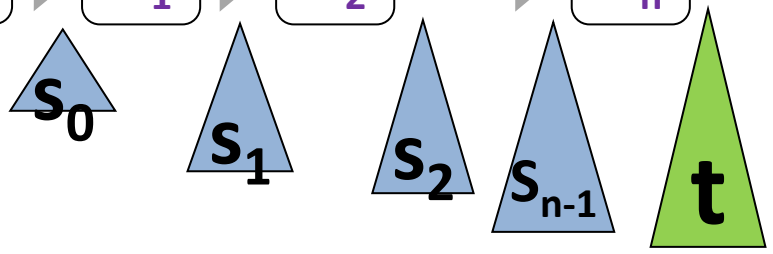
2. Split each 1-HTT to (LT; 1-HTT).
   = deleting and productive part



3. Fuse deleting part ahead.
   1-HTT ; LT ⊆ 1-HTT



4. Now all intermediate trees must be small.
   Try them all in DLINSPACE.



"Safe" :: $D_{i+1} = \{D_i^k \to D_i\}$	"Unsafe" :: $D \to D$

Grammars

- MSO model … decidable. […
- Hierarchy is …
- Equivalent to "iterated pushdown automata" [Da 82]
  (= (stack of)* stacks)
- **Context-sensitive.**
  [Maneth 02][I.&Maneth 08]

Transducers [EV88]

- **n-DHTT = (1-DHTT) ^{n}**
- **n-NHTT $\subseteq$ (1-NHTT) ^{n}**

Open Questions

… king is
…, Kobayashi 09]
…. [Kartzow&Parys …

- Equivalent to "*collapsible* pushdown automata" [Hague&Murawski&Ong&Serre 08]
- **Is unsafe higher-order languages context-sensitive?**

- **Does unsafe higher-order transducers have first-order decompositions?**

Idea: Stack-TT

- 1-HTT has difficulty in implementing ***capture-avoiding substitution***.
- How about extending them with a stack

- f (**a** x_1 ... x_n) y_1 ... y_m ys → RHS

  where RHS ::= **d** RHS ... RHS
  | y_i
  | f x_i RHS ... RHS ys

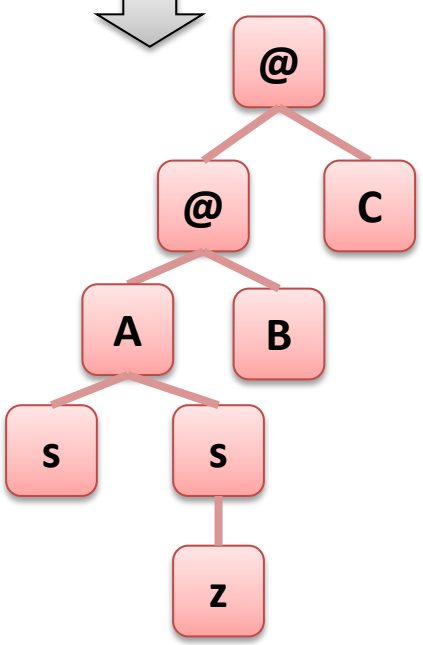
POP m values

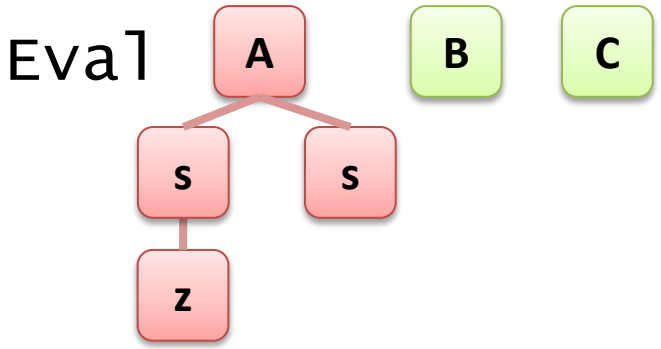
PUSH

Unsafe substitution ➔
De-Bruijn index + Stack-TT

(λx. λy. (A x y)) B C

(λ. λ. (A 0 1)) B C






```
Eval (@ f a) ys
     ➔ Eval f (Eval a ys) ys
Eval (s x) y ys ➔ Eval x ys
Eval z     y ys ➔ y
```

PUSH!

POP!

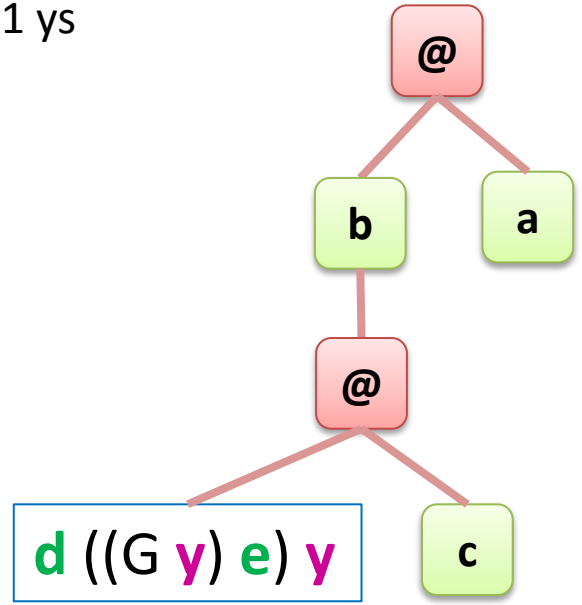
Even if it's unsafe...

```
S     → F a
F y   → b (H (G y) c)
G x y → ...
H f y → d (f e) y
```

```
S     → @ F a
F     → b (@ (H (@ G 2)) c)
G     → ..
H f   → d (@ (k1d2 f) e) 0
```

Eval (**k1d2** x) y1 y2 y3 ys
= Eval x y1 ys

```
S     → @ F a
F     → b (@ (H (@ G y)) c)
G     → ..
H f   → d (@ f e) y
```



Pros & Cons

Found an error in the proof during discussion in the seminar... X(

- Good: ~~Theorem: Unsafe-n-HTT ⊆ (Stack-TT)^n~~
- Good: Theorem: (Stack-TT ; LT) ⊆ Stack-TT

- Bad: it is hard to make it garbage-free.
  - There can be a sequence of significant stack operations not generating output.
- Bad: it may be overly powerful.

Theorem: Stack-TT ⊄ Unsafe-n-HTT

Proof: inverse of stack-TT does not preserve regularity, while unsafe-n-HTT does.

Summary

- “Safe” and “unsafe” HTT are different.

- Can we transfer results in “safe” case to “unsafe”?
  - Can we decompose an unsafe-HTT to 1st order machines?
  - Can we show context-sensitivity of the output language?

- See http://www.kmonos.net/pub/tmp/smtt.pdf for the technical development.