

Decompositions of Higher-Order Grammars to First-Order Transducers

Kazuhiro Inaba

Regular Language

Input: a string **t**

Determine if **t** belongs to
a regular language

$O(|t|)$ time

$O(1)$ space

$(0 \mid 1(0(1^*0)^+)^*1)^+$

Context-Free Language

Input: a string t

$$O(|t|^{\omega})_{\text{time}}$$

= the order of matrix multiplication

$$O((\log |t|)^2)_{\text{space}}$$

E	::=	T		T + T		T - T
T	::=	F		F * F		F / F
F	::=	(E)		\emptyset		1N
N	::=	\emptyset		1		$\emptyset N$ 1N

Macro Language [Fischer68, Aho68, Rounds73]

- Nonterminals of CFG
has type :: **string**.
- MG can have **string->string**
or **(string, string)->string**,
etc.

NP-complete

$O(|t|)$ space

```
S() ::= T(, , )
T(x, y, z)
    ::= T(ax, by, cz)
    | xyz
```

Higher-Order Tree Grammars

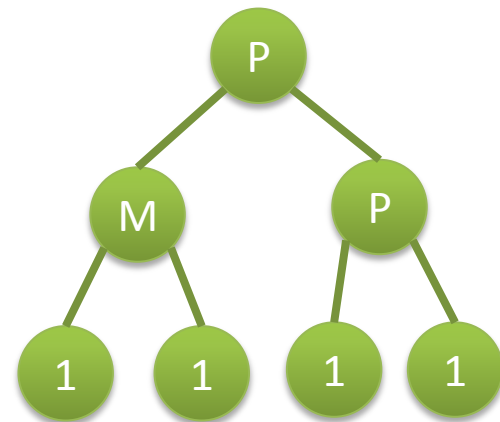
- $((\text{tree} \rightarrow \text{tree}) \rightarrow \text{tree}) \rightarrow \text{tree}$
- etc.

????? time

????? space

```

S          ::= TWICE(X)
TWICE(f)   ::= f(f(1))
X(e)       ::= Plus(e, e)
              | Minus(e, e)
  
```



Goal of Today's Talk (1)

For “**safe**” subset of higher order grammars,
it is still:

NP_{time}

O(|t|)_{space}

Goal of Today's Talk (2)

Conjecture:

For **any** higher order grammars, it is still:

NP_{time}

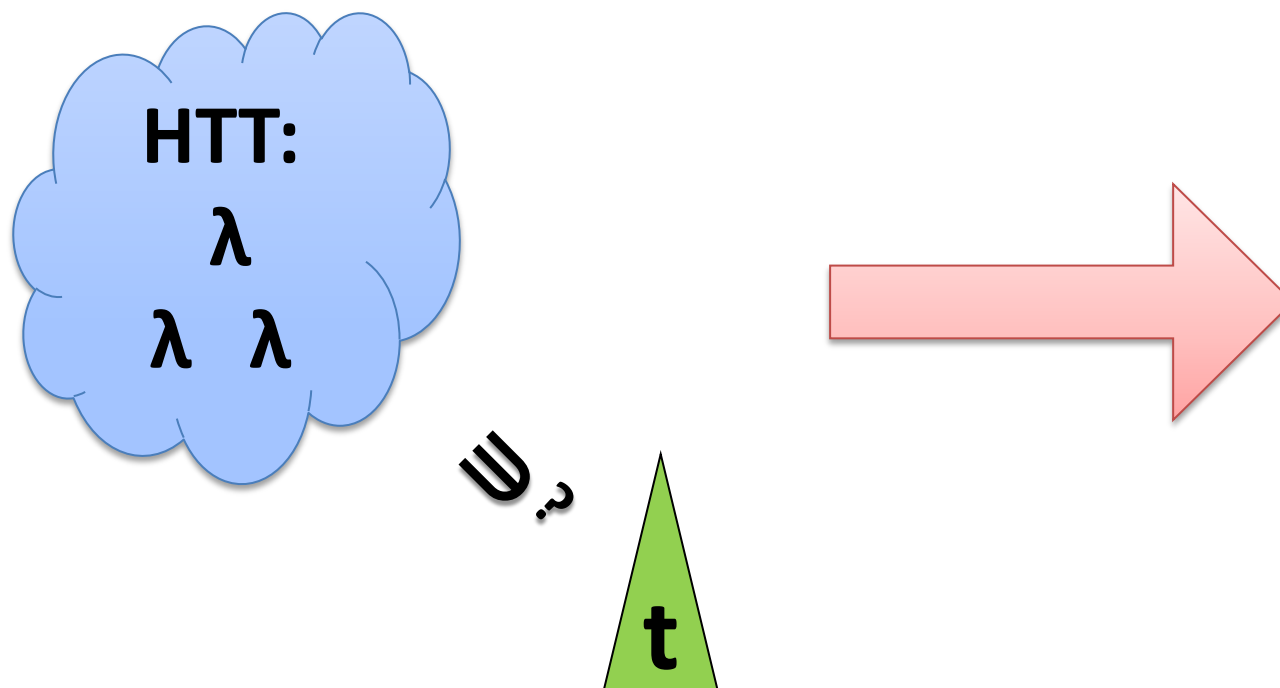
O(|t|)_{space}

Caution:

- In this talk, we concentrate on the complexity with respect to the size of the input tree **t**.
- Regard grammar **G** as fixed.
- “**O**(**|t|**) space” means “**O**(**|t|** **f**(**|G|**)) space”.
 - Indeed, $f(x) \geq n\text{-EXP}$ where n is the highest order.

Overview of Our Approach

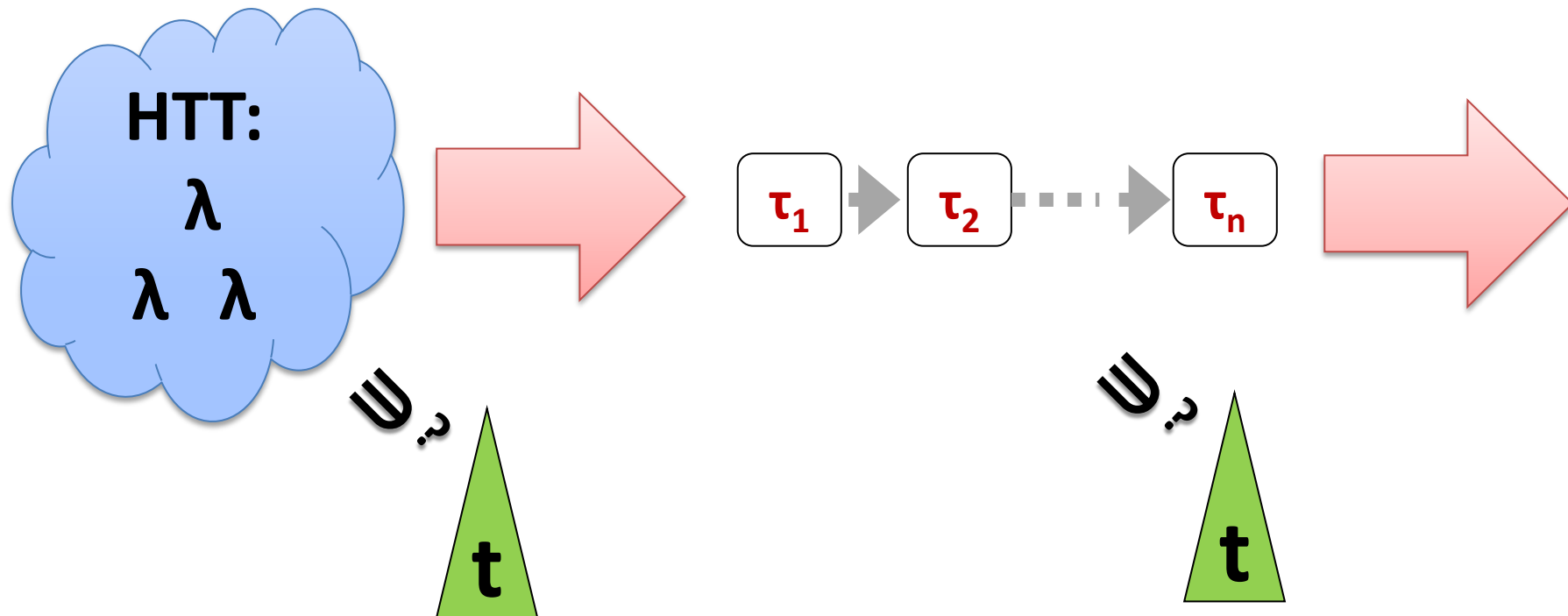
- 1) Introduce HTT: Higher-order Tree Transducers
 - A slight generalization of higher-order grammars.
 - Examine the problem: “Can **t** be an output of a HTT?”



Overview of Our Approach

2) First Order Decomposition

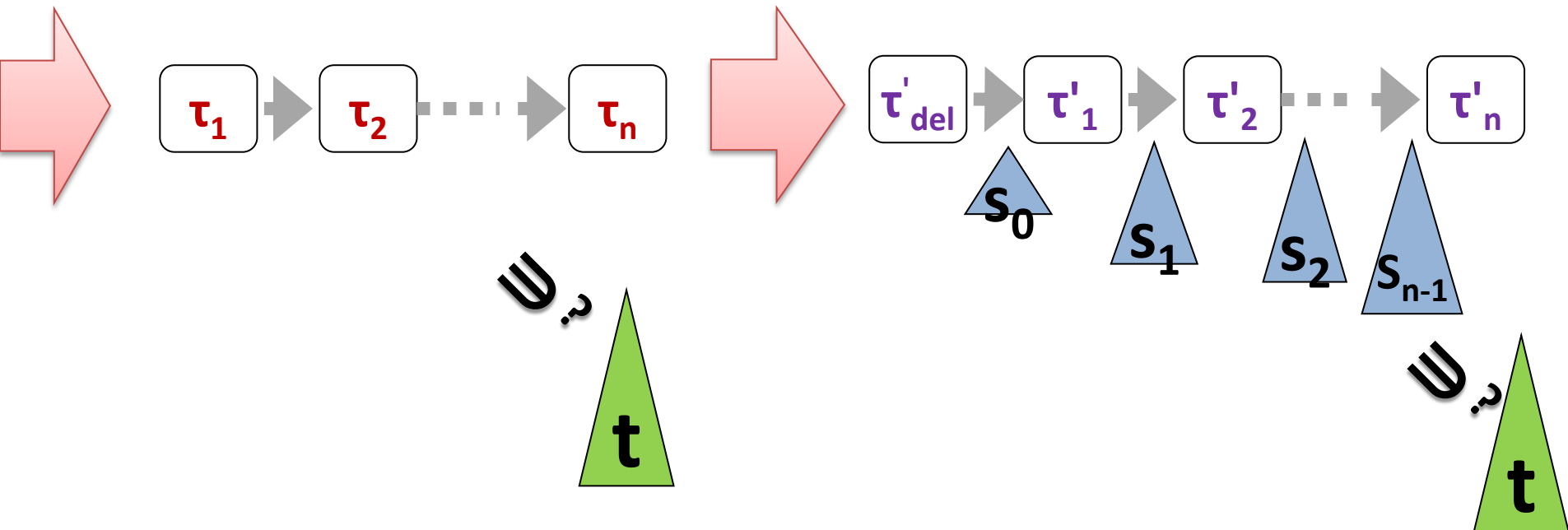
- Show order- n HTT is simulatable by n composition of first order HTTs.



Overview of Our Approach

3) “Garbage Free Form”

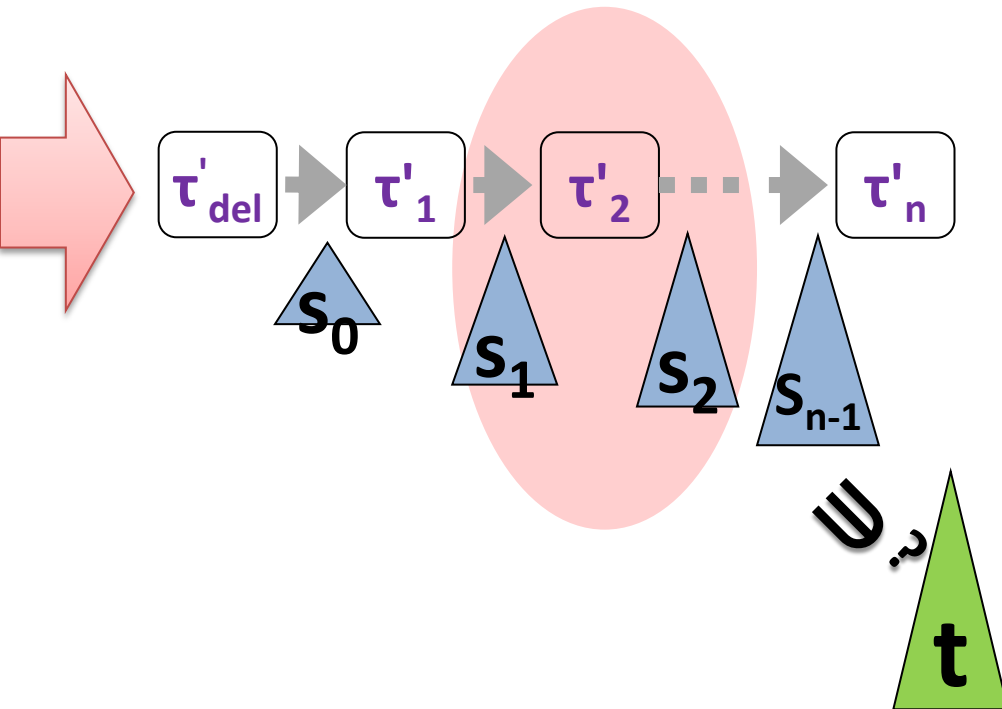
- Show that we can transform the HTTs so that all intermediate trees are smaller than t .



Overview of Our Approach

4) Subproblem: translation membership

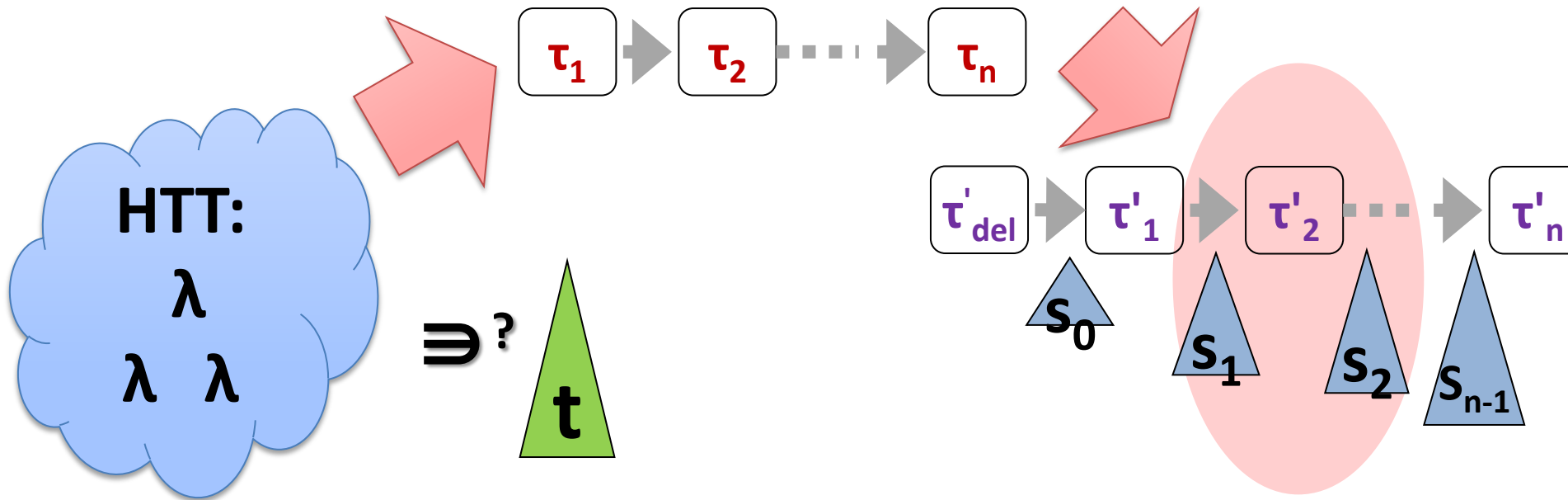
- Given trees **s1**, **s2** and 1-HTT **τ** , can we determine “ **$\tau(s1) \ni s2$** ?” in NP / $O(|s1| + |s2|)$ space?



Overview of Our Approach

5) Wrap up! Given HTT G and a tree t ,

- Convert G to garbage-free 1st order composition
- “Guess” (by NP / $O(n)$ space) all intermediate trees s_k .
- Check each translation membership.



HTT [Engelfriet&Vogler 88]

Higher-order “single-input” “safe” tree transducer

$\text{Mult} :: \text{Tree} \rightarrow \text{Tree}$
 $\text{Mult}(\text{Pair}(x_1, x_2)) \rightarrow \text{Iter}(x_1)(\text{Add}(x_2))(\text{Z})$

$\text{Iter} :: \text{Tree} \rightarrow (\text{Tree} \rightarrow \text{Tree}) \rightarrow \text{Tree} \rightarrow \text{Tree}$
 $\text{Iter}(\text{S}(x))(f)(y) \rightarrow \text{Iter}(x)(f)(f(y))$
 $\text{Iter}(\text{Z})(f)(y) \rightarrow y$

$\text{Add} :: \text{Tree} \rightarrow \text{Tree} \rightarrow \text{Tree}$
 $\text{Add}(\text{S}(x))(y) \rightarrow \text{Add}(x)(\text{S}(y))$
 $\text{Add}(\text{Z})(y) \rightarrow y$

HTT

- Set of mutually recursive functions
 - Defined in terms of **induction on a single input tree**
 - Input trees are always consumed, not newly constructed
 - Output trees are always created, but not destructed
 - Rest of the parameters are **ordered by the order**
 - Multiple parameters of the same order is ok but in uncurried form

Inductive Input Param Order-1 Param(s) Order-0 Param(s) Result

```

Iter :: Tree → (Tree → Tree) → Tree → Tree
Iter(S(x))(f)(y)    → Iter(x)(f)(f(y))
Iter(Z)(f)(y)       → y
  
```

HTT

Nondeterminism (// and \perp)

```

Subseq :: Tree → Tree
Subseq(Cons(x,xs)) → Cons(x, Subseq(xs))
                  // Subseq(xs)
Subseq(Nil)       → Nil
Subseq(Other)     →  $\perp$ 

```

In this talk, evaluation strategy is unrestricted (= call-by-name).
But call-by-value can also be dealt with.

HTT

- Notation: **n-HTT**
 - is the class of **Tree** \rightarrow **Tree** functions representable by HTT of order $\leq n$.
 - {Subseq} is 0-HTT, {Mult, Iter, Add} \in 2-HTT

Subseq :: Tree \rightarrow Tree

Mult :: Tree \rightarrow Tree

Iter :: Tree \rightarrow (Tree \rightarrow Tree) \rightarrow Tree \rightarrow Tree

Add :: Tree \rightarrow Tree \rightarrow Tree

Order-n to Order-1

THEOREM [EV88] [EV86]

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

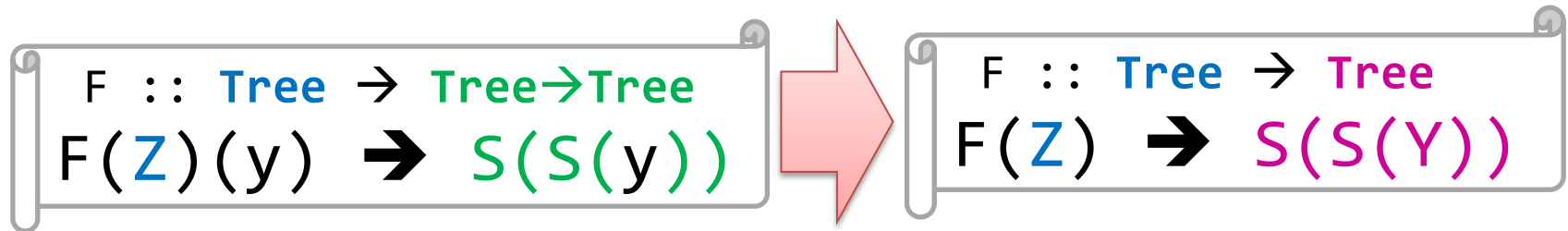
n -th order tree transducer is representable
by a n -fold composition of 1st-order tree
transducers.

[EV86] J. Engelfriet & H. Vogler, “Pushdown Machines for Macro Tree Transducers”, *TCS* 42
[EV88] —, “High Level Tree Transducers and Iterated Pushdown Tree Transducers”, *Acta Inf.* 26

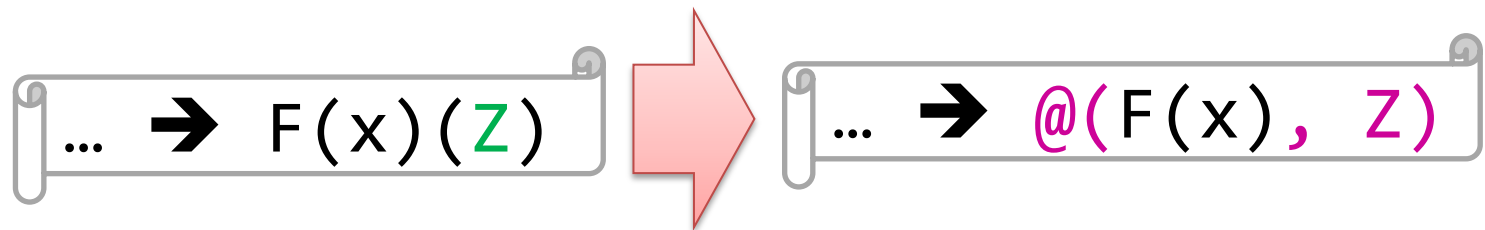
Proof: $n\text{-HTT} = 1\text{-HTT} \circ (n-1)\text{-HTT}$

Idea:

Represent 1st-order term $\text{Tree} \rightarrow \text{Tree}$ by a Tree .



Represent 1st-order application symbolically, too.



Proof: $n\text{-HTT} = 1\text{-HTT} \circ (n-1)\text{-HTT}$

Represent 1st-order things symbolically.

$F :: \text{Tree} \rightarrow \text{Tree}$
 $F(\text{Z}) \rightarrow S(S(Y))$

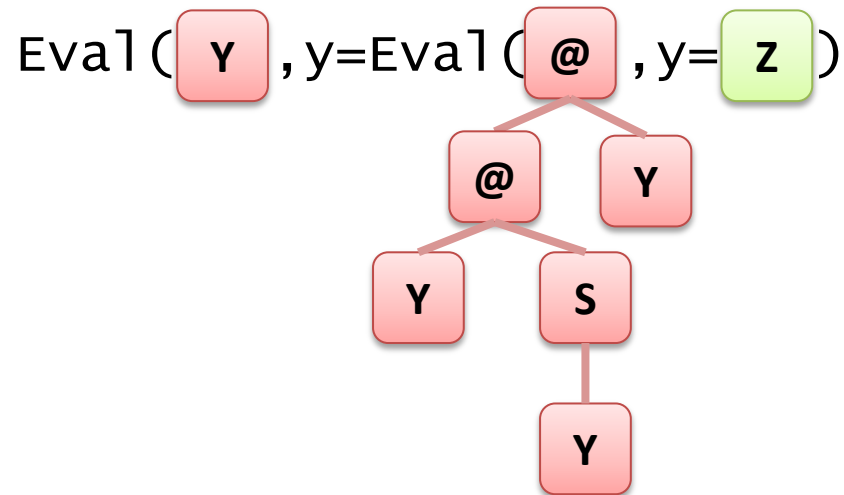
$\dots \rightarrow @(F(x), Z)$

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

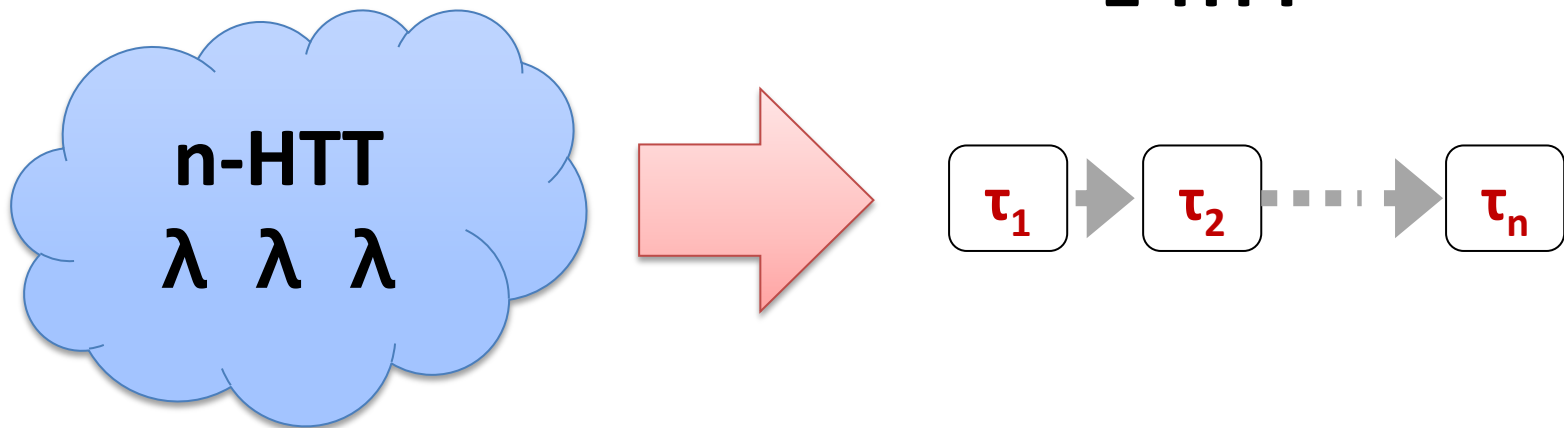
$\text{Eval}(@(f, b))(y) \rightarrow \text{Eval}(f, \text{Eval}(b)(y))$
 $\text{Eval}(Y)(y) \rightarrow y$
 $\text{Eval}(S(x))(y) \rightarrow S(\text{Eval}(x)(y))$
 $\text{Eval}(Z)(y) \rightarrow Z$

Why That Easy

- Relies on the **ordered-by-order** condition.
 - No variable renaming is required! [Blum&Ong 09]

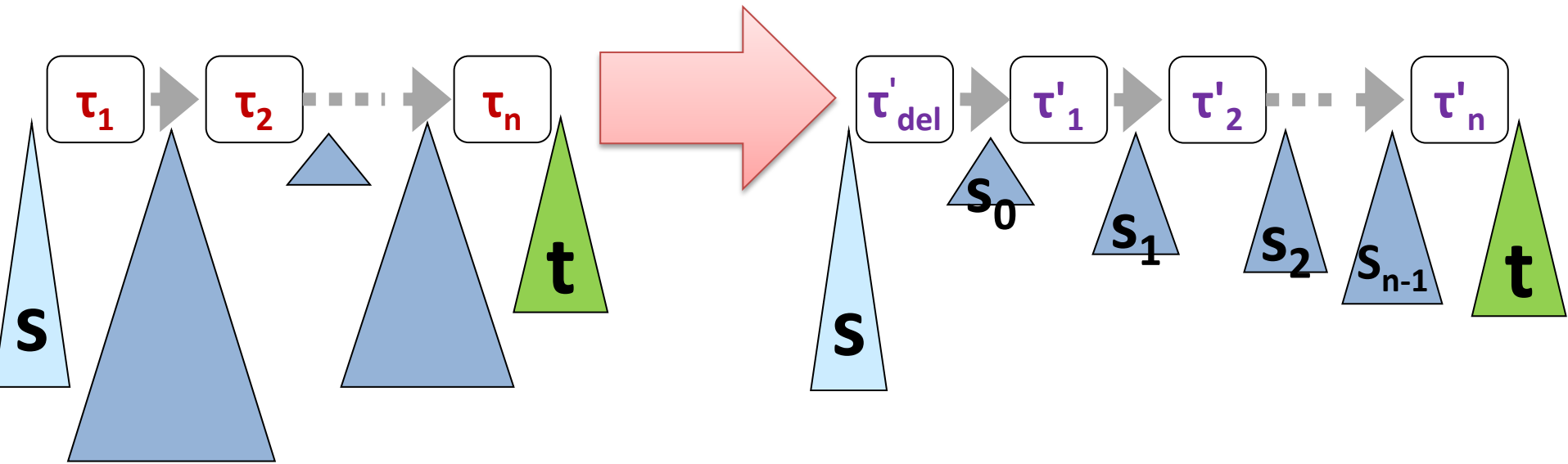


Decomposed.



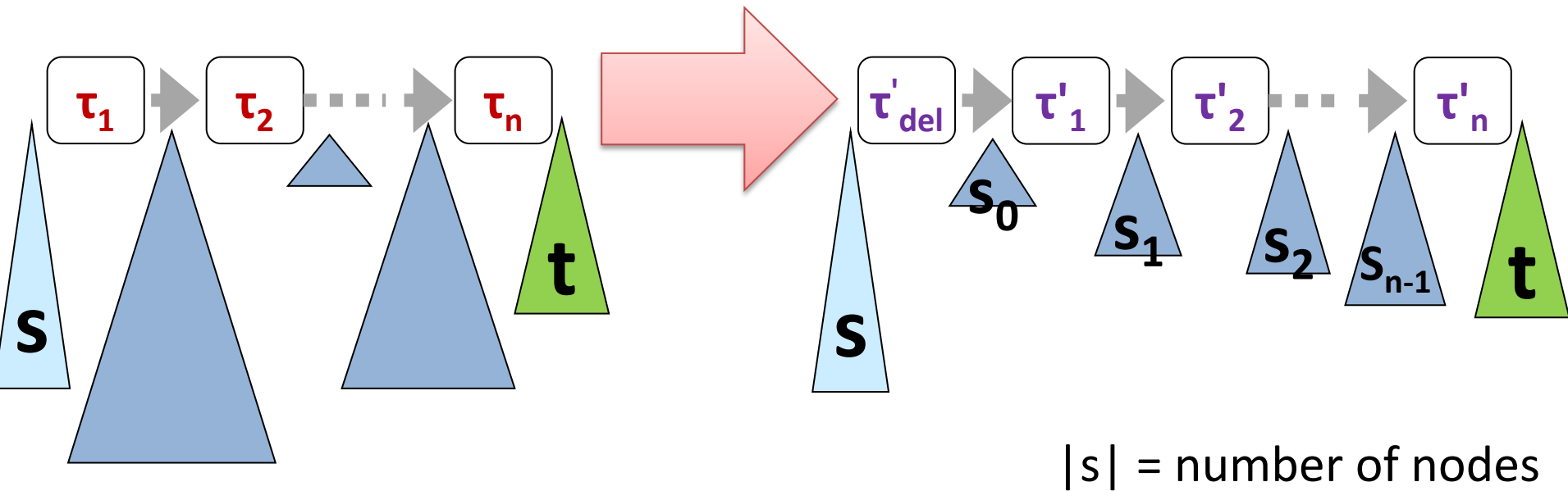
Next, Make Intermediate Trees Small.

1-HTTⁿ



THEOREM [I. & Maneth 08] [I. 09]^(+ improvement)

$\forall \tau_1, \dots, \tau_n \in \mathbf{1-HTT}, \exists \tau'_{\text{del}} \in \mathbf{0-LHTT}, \tau'_1, \dots, \tau'_n \in \mathbf{1-HTT},$
 for any $(\tau_n \circ \dots \circ \tau_1)(s) \ni t,$
 there exist $\tau'_{\text{del}}(s) \ni s_0, \tau'_i(s_i) \ni s_{i+1}, |s_i| \leq |s_{i+1}|, s_n = t.$

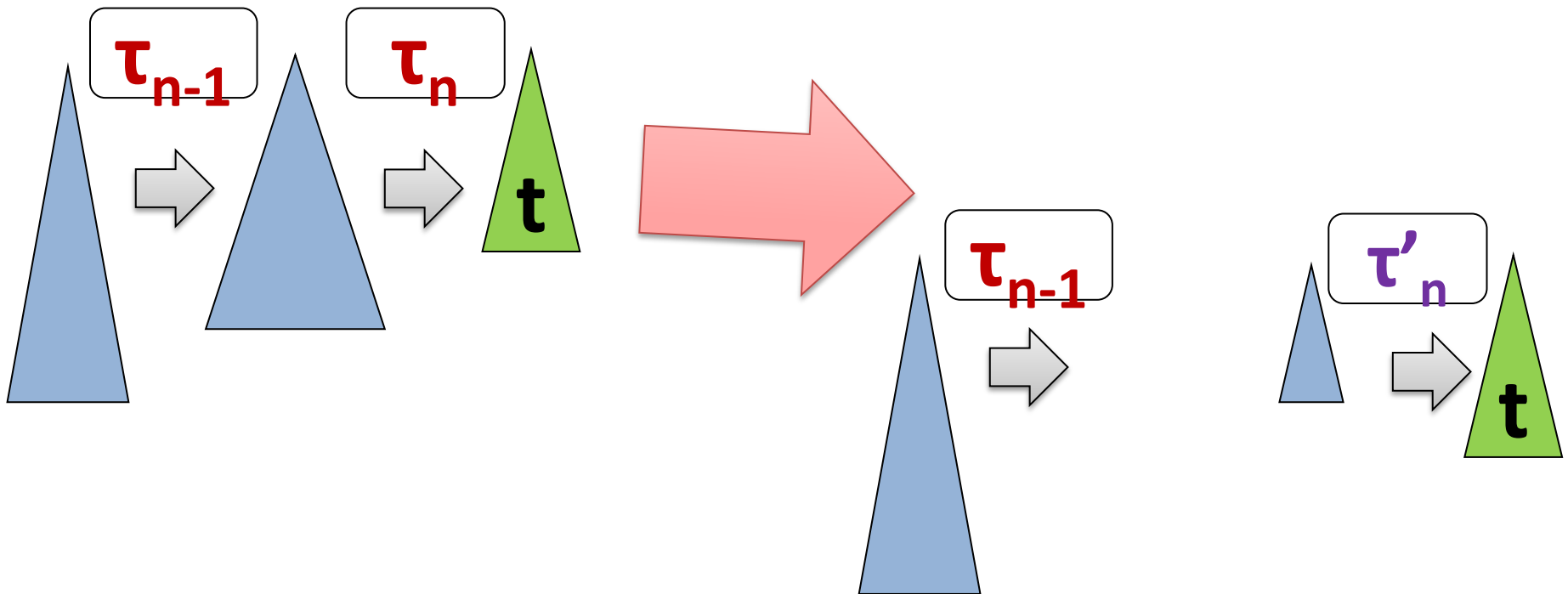


[IM08] K. Inaba & S. Maneth, “The complexity of tree transducer output languages”, *FSTTCS*

[Inaba09] K. Inaba, “Complexity and Expressiveness of Models of XML Transformations”, *Dissertation*

How to Construct the “Garbage-Free” Form

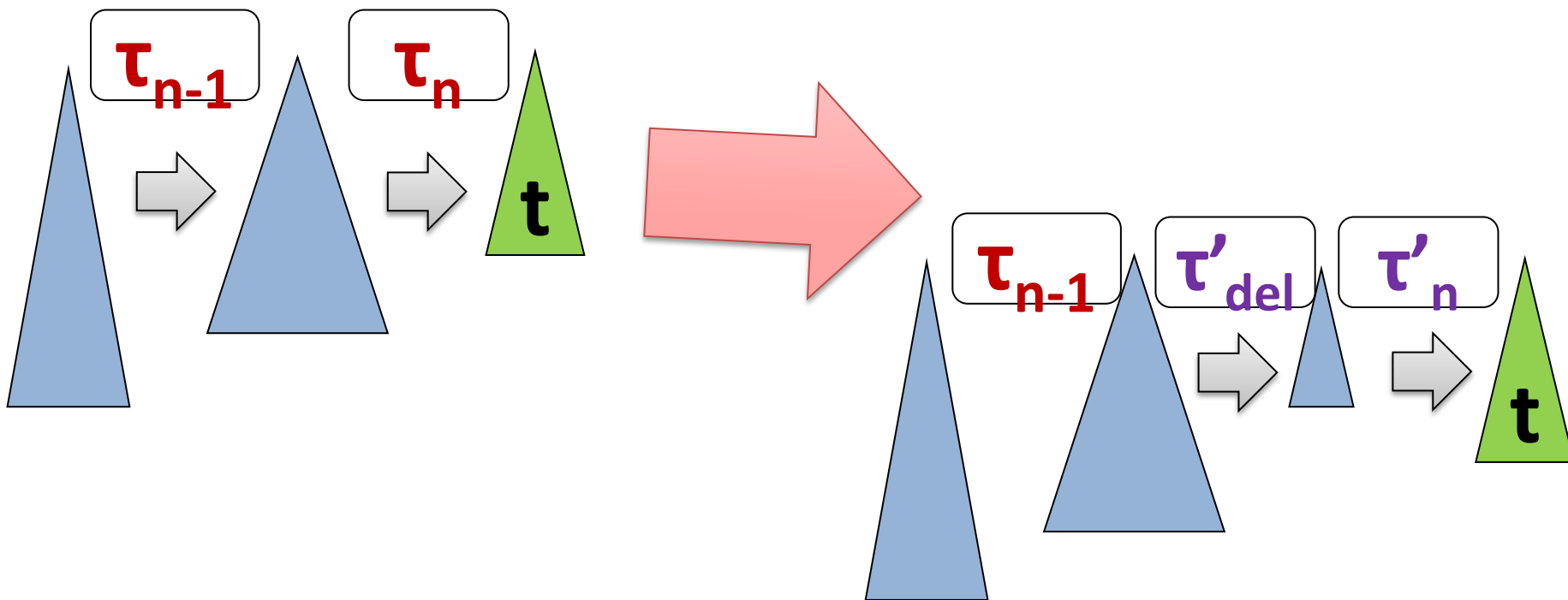
Make each 1-HTT “productive”



How to Construct the “Garbage-Free” Form

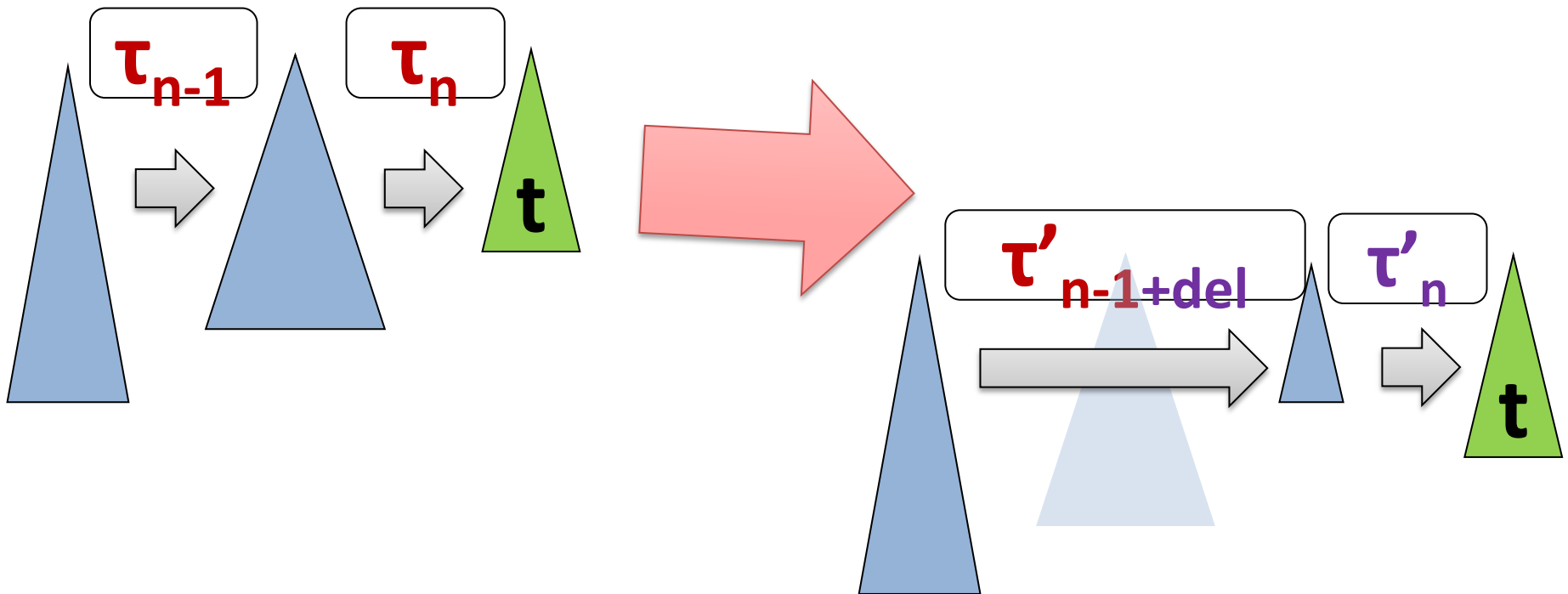
Make each 1-HTT “productive”
by separating its “deleting” part

$$\tau_n = \tau'_{\text{del}} \tau'_n$$

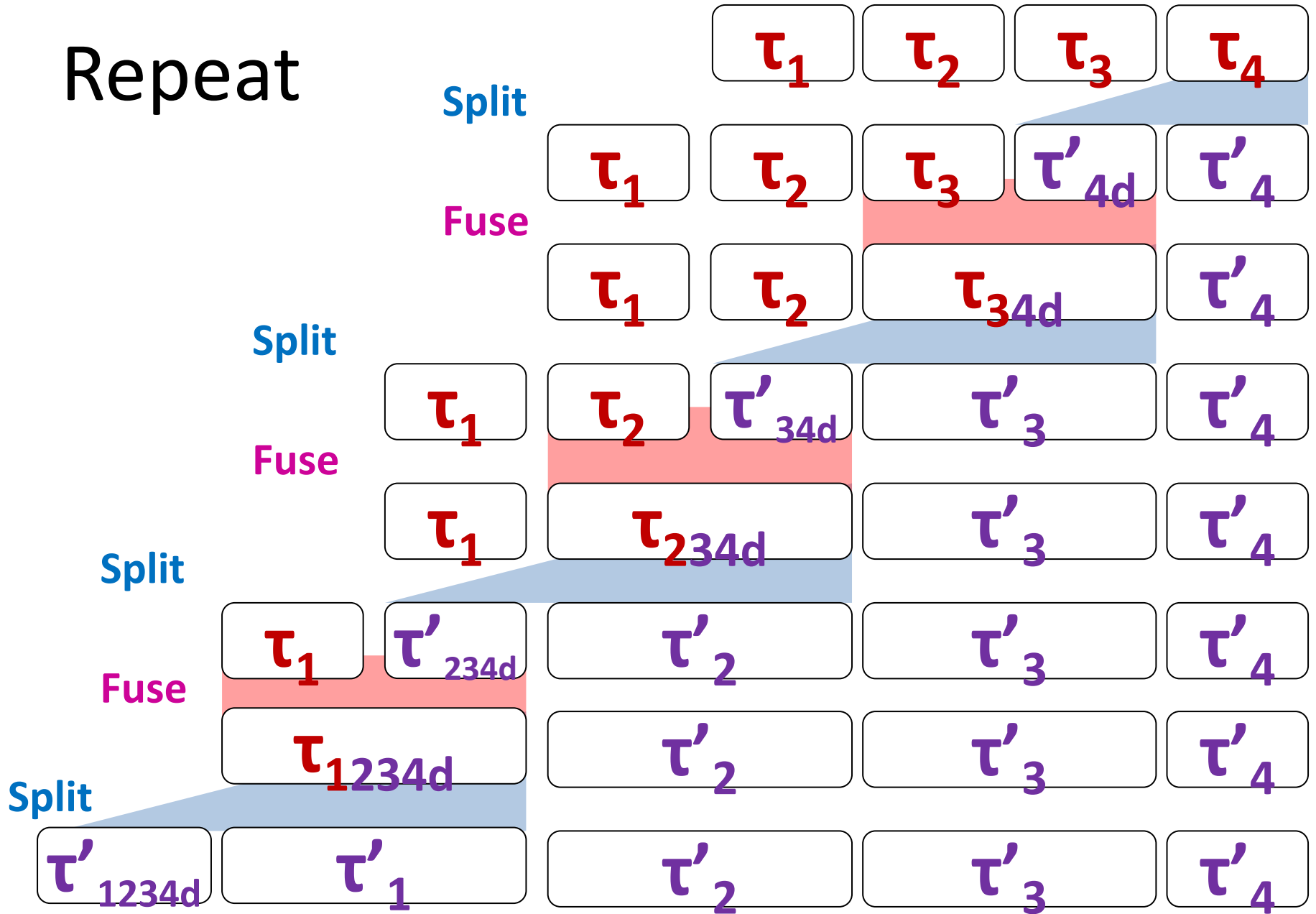


How to Construct the “Garbage-Free” Form

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



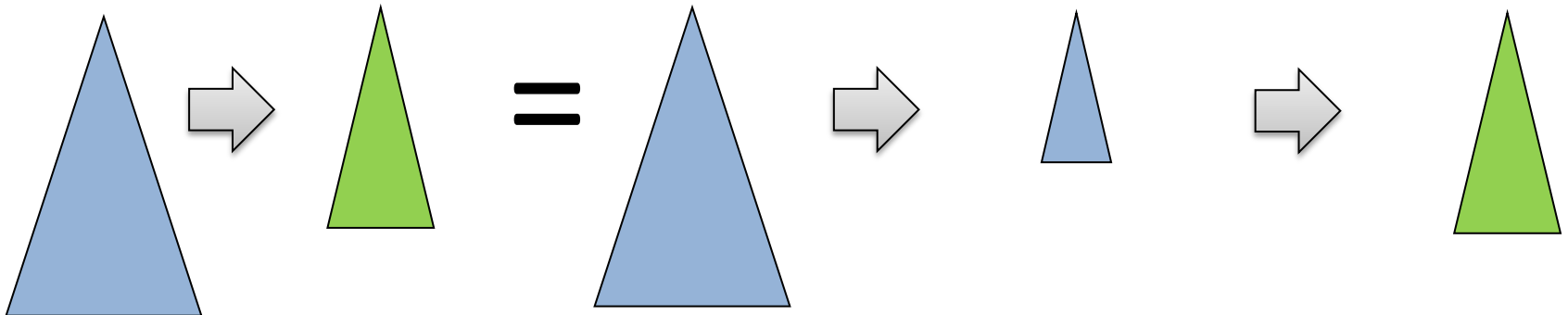
Repeat



Key Part

Separate the “deleting” transformation

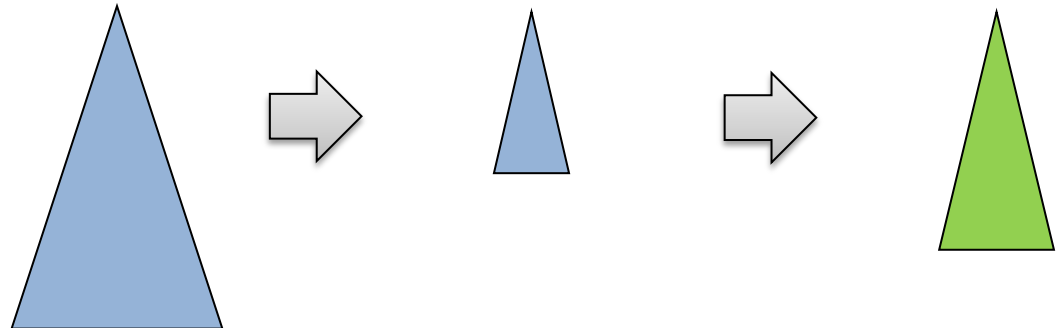
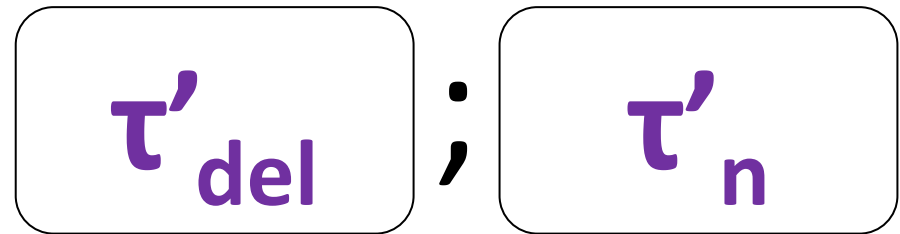
$$\tau_n = \tau'_{\text{del}} ; \tau'_n$$



Key Part

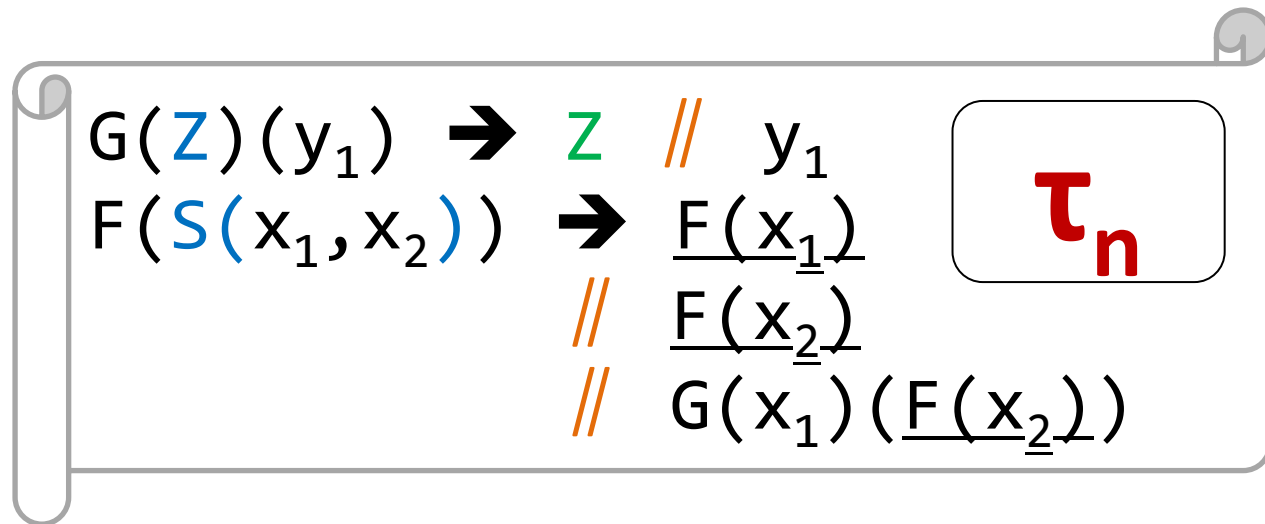
Slogan: **Work on every node**

(τ'_n must generate at least one node for each input node)



Work on Every Node \Rightarrow Visit All Nodes

Deleting HTTs



may not recurse down to a subtree.

Work on Every Node \Rightarrow Visit All Nodes

$$F(\textcolor{blue}{S}(x_1, x_2)) \rightarrow G(x_1)(F(x_2))$$

 τ_n

Nondeterministically delete every subtree!

 τ'_{del}

$$\text{Del}(\textcolor{blue}{S}(x_1, x_2)) \rightarrow$$

$$\begin{aligned} &\textcolor{violet}{S12}(\text{Del}(x_1), \text{Del}(x_2)) \quad // \quad \textcolor{violet}{S1_}(\text{Del}(x_1)) \\ &// \quad \textcolor{violet}{S_2}(\text{Del}(x_2)) \quad // \quad \textcolor{violet}{S_}() \end{aligned}$$

At least one choice
of nondeterminism
“deletes correctly”.

$$F(\textcolor{violet}{S12}(x_1, x_2)) \rightarrow G(x_1)(F(x_2))$$

$$F(\textcolor{violet}{S1_}(x_1)) \rightarrow G(x_1)(\textcolor{brown}{\perp})$$

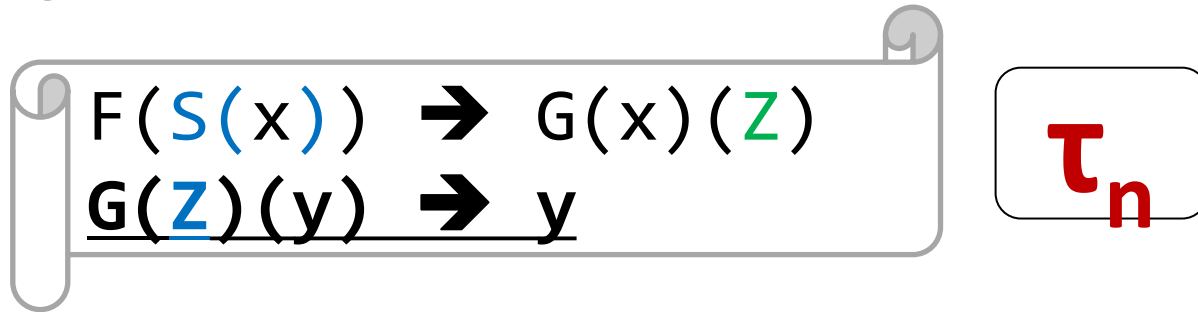
$$F(\textcolor{violet}{S_2}(x_2)) \rightarrow \textcolor{brown}{\perp}$$

$$F(\textcolor{violet}{S_}()) \rightarrow \textcolor{brown}{\perp}$$

 τ'_n

Work on Every Node \Rightarrow Work on Leaf

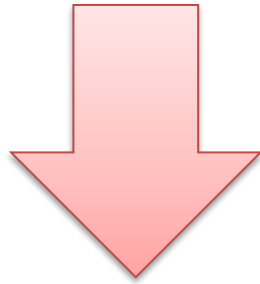
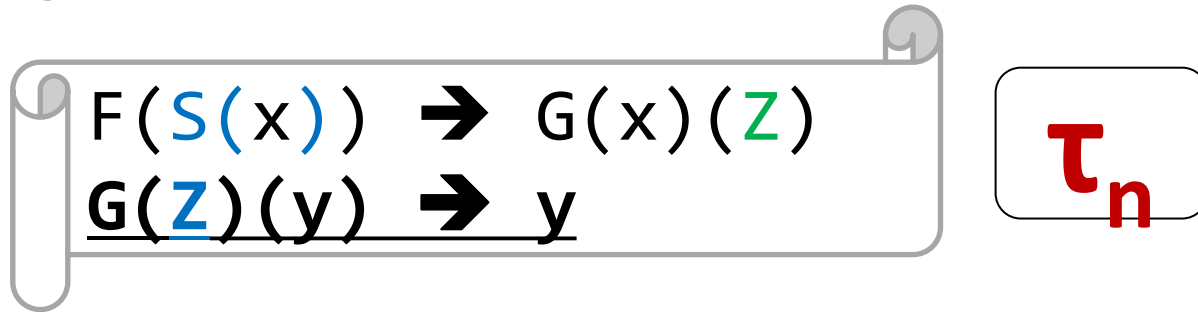
Erasing HTTs



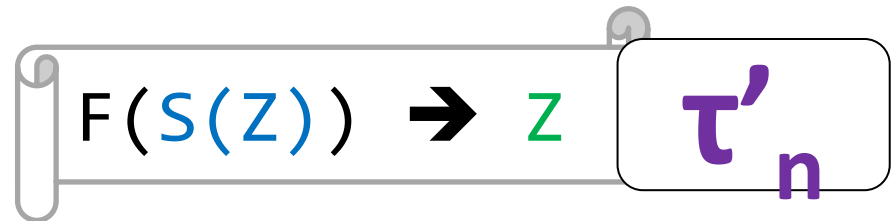
may be idle at leaves.

Work on Every Node \Rightarrow Work on Leaf

Erasing HTTs



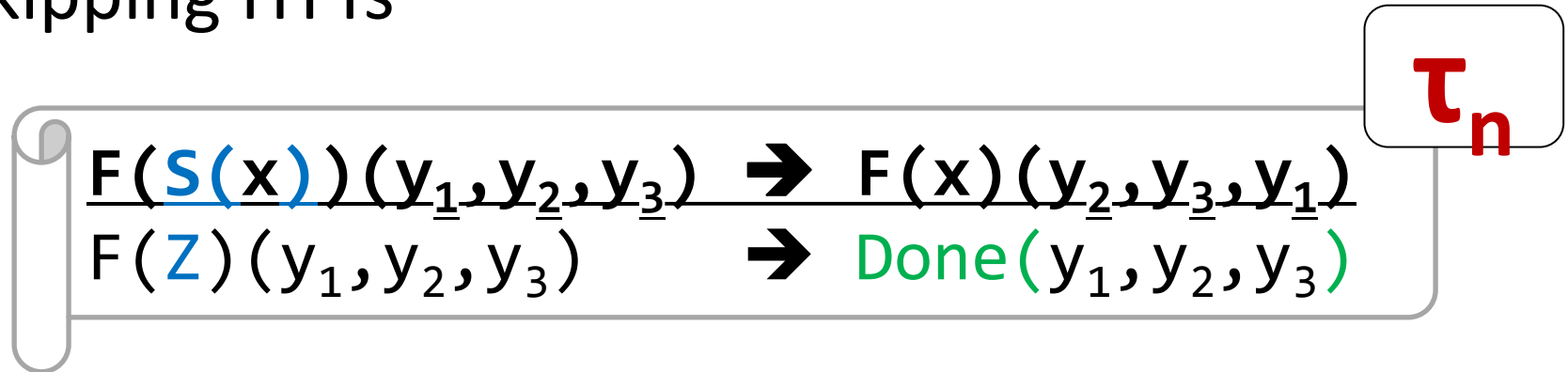
Inline Expansion



Work on Every Node

\Rightarrow Work on Monadic Nodes

Skipping HTTs



are good at juggling.

Work on Every Node

\Rightarrow Work on Monadic Nodes

Skipping HTTs

$\cancel{F(S(x))(y_1, y_2, y_3)} \rightarrow \cancel{F(x)(y_2, y_3, y_1)}$
 $F(Z)(y_1, y_2, y_3) \rightarrow \text{Done}(y_1, y_2, y_3)$

τ_n

Nondeterministic deletion again.

Remember how arguments would've been shuffled.

$F(Z123)(y_1, y_2, y_3) \rightarrow \text{Done}(y_1, y_2, y_3)$
 $F(Z231)(y_1, y_2, y_3) \rightarrow \text{Done}(y_2, y_3, y_1)$
 $F(Z312)(y_1, y_2, y_3) \rightarrow \text{Done}(y_3, y_1, y_2)$

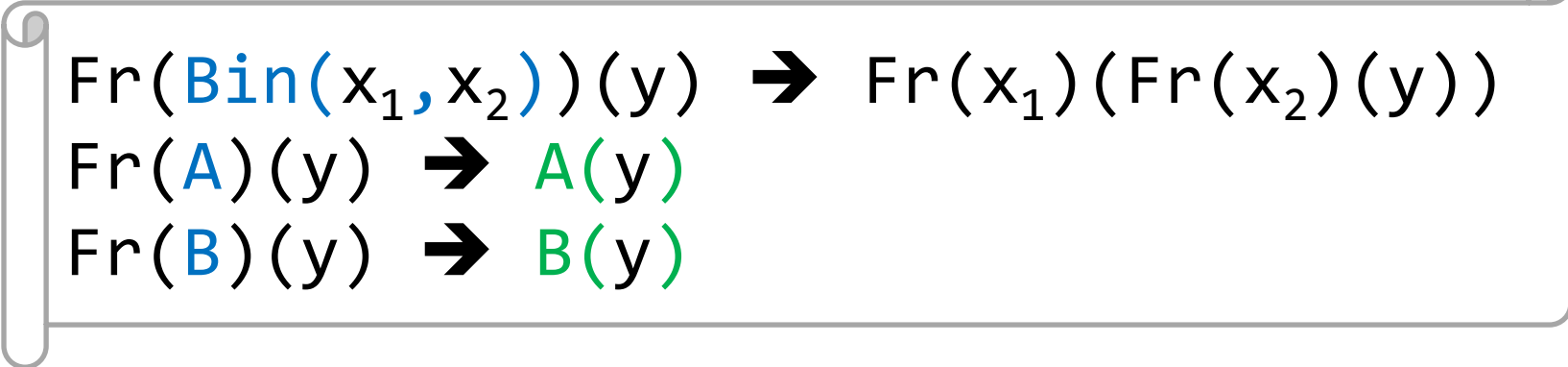
τ'_n

Simple Arithmetic

- **Input size** = **#leaf** + **#monadic** + **#others**
 - For each leaf on the input, generate ≥ 1 node.
 - For each monadic node, generate ≥ 1 node.
 - Thus, **#leaf** + **#monadic** \leq **Output size**.
- For any tree, **#others** < **#leaf** \leq **Output size**.
- Add: **#leaf** + **#monadic** + **#others** \leq **Output size***2
- So, **Input size** < **Output Size** * 2

Work on Nodes with Rank-2,3,...

- **Input size** < **Output Size** * 2



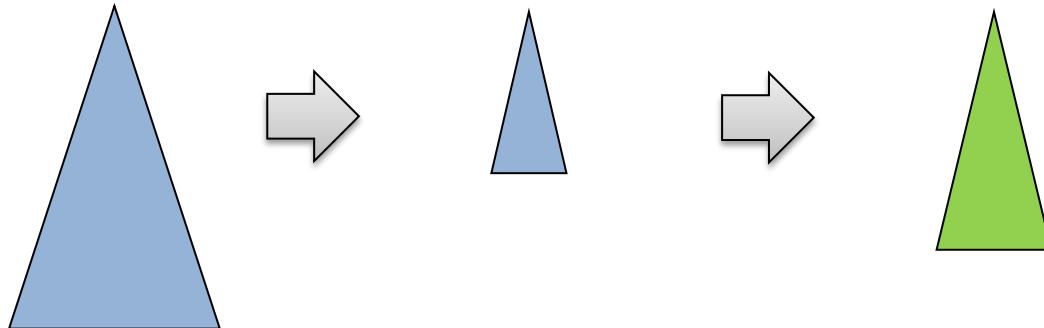
$\text{Fr}(\text{Bin}(x_1, x_2))(y) \rightarrow \text{Fr}(x_1)(\text{Fr}(x_2)(y))$
 $\text{Fr}(A)(y) \rightarrow A(y)$
 $\text{Fr}(B)(y) \rightarrow B(y)$

This bound is sufficient for deriving the results,
but we can improve this to **Input size** \leq **Output Size**,
by *deterministic deletion of leaves + inline expansion*.

Done!

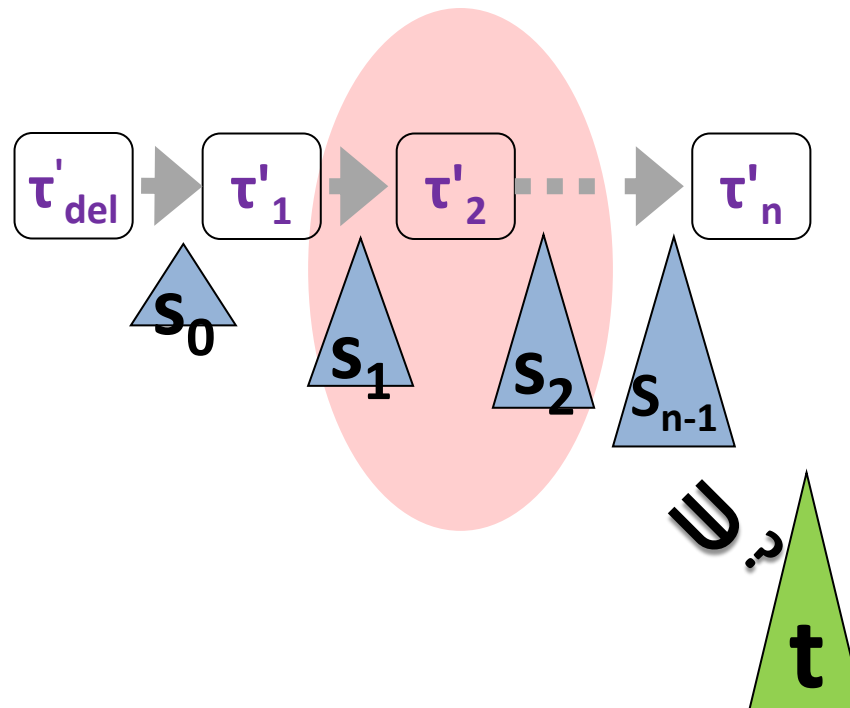
Intermediate trees are small!

$$\tau'_{\text{del}} ; \tau'_n$$



Next.

“Translation membership problem”

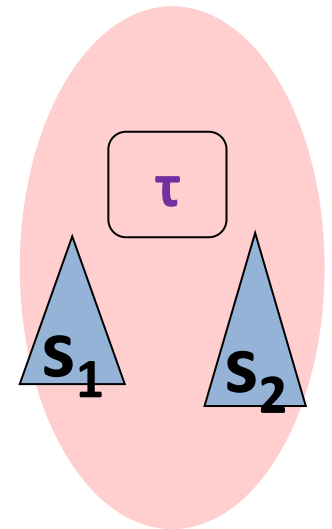


Translation Membership

Given trees **s1**, **s2** and **τ**, can we determine “ $\tau(s1) \ni s2$?” in NP / $O(|s1| + |s2|)$ space?

From the construction, **τ** is always

- 1st order HTT
- Non-deleting/erasing/skipping.
- **Path-linear**: recursive call to the same child node will not nest.
 - OK: **Node**(f(x), g(x)) BAD: f(**x**, g(**x**))
 - ➔ $\text{height}(s2) \in O(|s1|)$



Example

τ

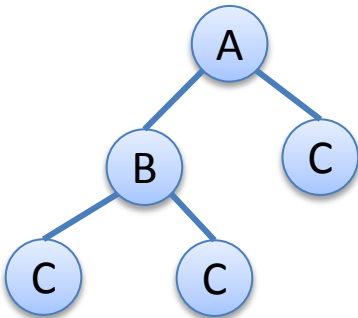
$S(x) \rightarrow F(x)(\Delta)$

$F(A(x_1, x_2))(y) \rightarrow F(x_1)(\alpha(F(x_2, y)))$

$F(B(x_1, x_2))(y) \rightarrow F(x_2)(\beta(F(x_2, y)))$

$F(C)(y) \rightarrow \Gamma(y)$

$s1$



$s2$



Basic Idea:
Just compute $\tau(s1)$ and compare.

τ

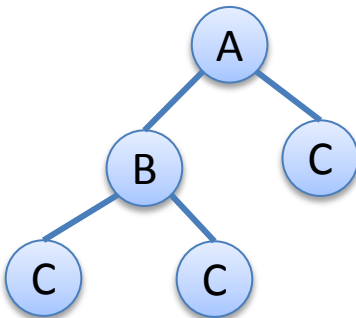
$S(x) \rightarrow F(x)(\Delta)$

$F(A(x1, x2))(y) \rightarrow F(x1)(\alpha(F(x2, y)))$

$F(B(x1, x2))(y) \rightarrow F(x2)(\beta(F(x2, y)))$

$F(C)(y) \rightarrow \Gamma(y)$

$s1$



$\tau(s1)$



$s2$



Key Points

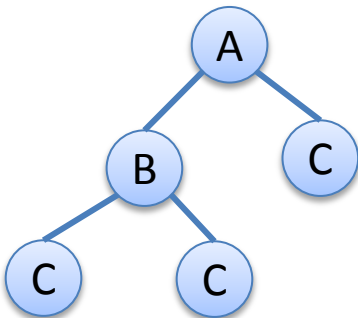
- $\tau(s1)$ may be very big
 - ➔ Compute $\tau(s1)$ incrementally. If it becomes larger than $s2$, return false immediately.
- τ may be nondeterministic
 - ➔ For NP algorithm, use the nondeterminism.
“non-deleting” property ensures polynomial choices.
 - ➔ For linear space algorithm, do back-track search.
The “call-stack” is linear-size bounded (next page),
so it can be done in linear space.

Each node corresponds to
a “call-stack”

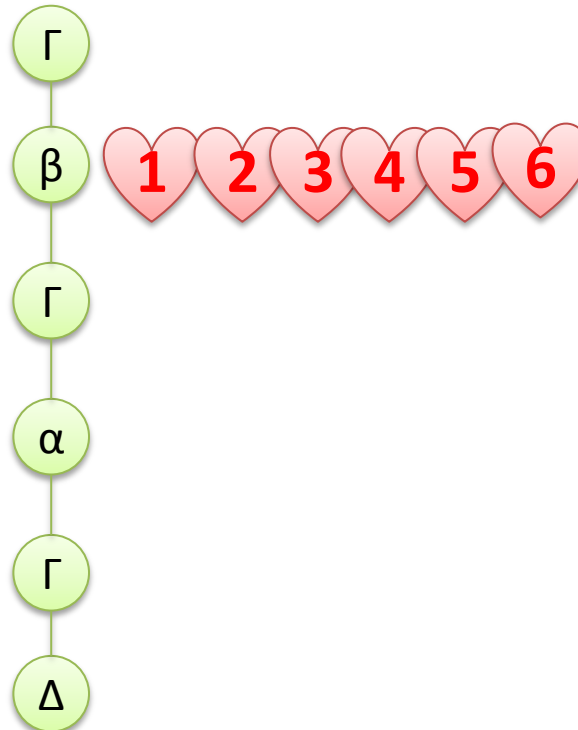
τ

$S(x) \rightarrow \text{♥1 } F(x)(\Delta)$
 $F(A(x1, x2))(y) \rightarrow \text{♥2 } F(x1)(\alpha(F(x2, y)))$
 $F(B(x1, x2))(y) \rightarrow \text{♥3 } F(x2)(\text{♥6 } \beta(F(x2, y)))$
 $F(C)(y) \rightarrow \text{♥4 } \Gamma(\text{♥5 } y)$

$s1$



$\tau(s2)$

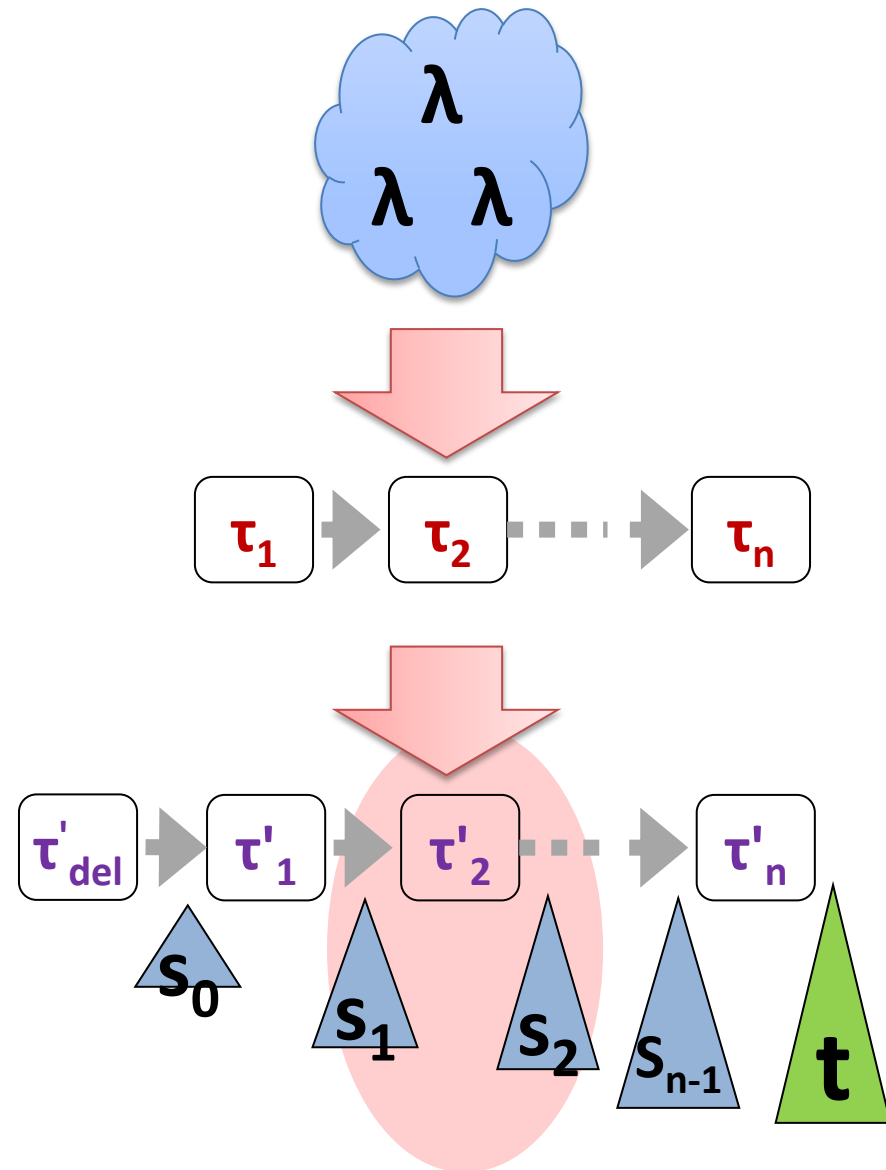


$s2$



Summary

- Safe n-HTT \rightarrow (1-HTT)ⁿ
- Split input-deletion
- Split erasing
- Split skipping
- Fuse deleter
- Translation membership



What about UNSAFE HTT?

- UNSAFE n-HTT \rightarrow (1-**stack**-HTT)ⁿ
 - **???: Now variable names matter. Use De Bruijn index.**
- Split input-deletion
 - **OK:** same technique works (nondet deletion)
- Split erasing
 - **OK:** same technique works (inline expansion)
- Split skipping
 - **????????????????????**
- Fuse deleter
 - **OK:** same technique works (product construction)
- Translation membership
 - **OK:** same technique works (call-stack size argument)

Stack-HTT

- Parameters are now passed as a stack.

$F :: \text{Tree} \rightarrow \text{Stack}\langle \text{Tree} \rangle \rightarrow \text{Tree}$

$F(\text{ADD}(x))(\dots, y1, y2)$

$\rightarrow F(x)(\dots, \text{PLUS}(y1, y2))$

$F(\text{SUB}(x))(\dots, y1, y2)$

$\rightarrow F(x)(\dots, \text{MINUS}(y1, y2))$

$F(\text{ONE}(x))(\dots) \rightarrow F(x)(\dots, 1)$

$F(\text{EOP})(f)(\dots, y) \rightarrow y$

POP!

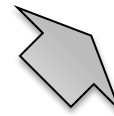
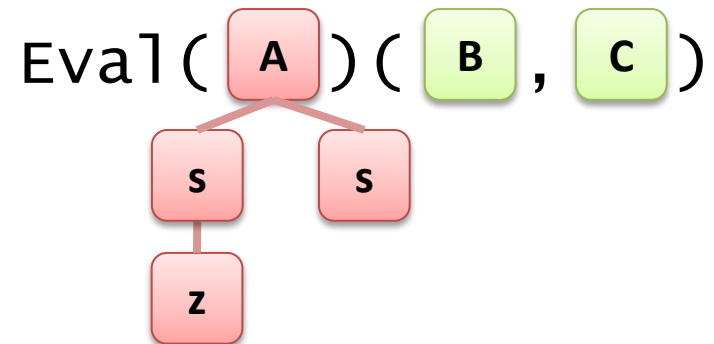
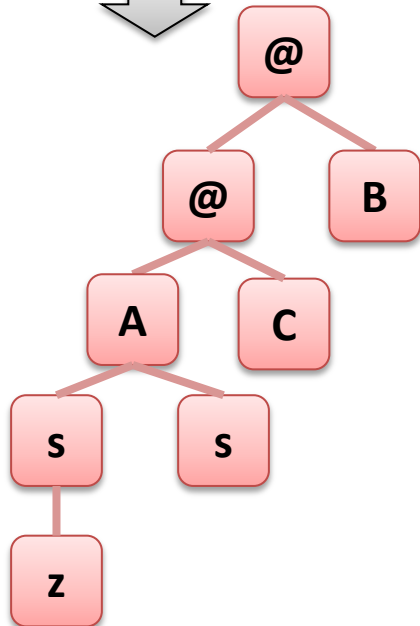
PUSH!

Unsafe substitution \rightarrow Stack HTT

$(\lambda x. \lambda y. A(x, y))(B)(C)$



$(\lambda. \lambda. A(1, 0))(B)(C)$



```
Eval(@(f, a))(...)
   $\rightarrow$  Eval(f)(..., Eval(a)(...))
Eval(s(x))(..., y)
   $\rightarrow$  Eval(x)(...)
Eval(z)(..., y)  $\rightarrow$  y
```

n-Unsafe-HTT \Rightarrow Substitution

$F(\sigma(x...))(y_0)...(y_n)(z) \rightarrow \text{rhs}$ where $z :: \text{tree}$



$F(\sigma(x...))(y_0)...(y_n) \rightarrow [[\text{rhs}]]_0$

- $[[e]]_{k+1} = s([e]_k)$ if $e :: \text{tree}$
 - $[[y_i]]_0 = s(y_i)$ if $y_i :: \text{tree}$
 - $[[z]] = z$
 - $[[x(...)]_k] = x([...]_{k+1})$ if $x(...) :: \text{tree} \rightarrow \text{tree}$
 - ...
 - $[[e_1(e_2)]_k] = @([e_1]_{k+1}, [e_2]_k)$ if $e_1 :: \text{tree} \rightarrow \text{tree}$
 - $[[e_1(e_2)]_k] = [[e_1]]_k ([e_2]_k)$ otherwise
- ...something like this (the above presentation may not be correct) should work, I hope!*

Example

$I :: \text{tree}$

$I \rightarrow S(\textcolor{teal}{A})$

$I \rightarrow @(\textcolor{red}{S}, \textcolor{teal}{A})$

$S :: \text{tree} \rightarrow \text{tree}$

$S(x) \rightarrow F(G(F(x))(\textcolor{teal}{B}))(\textcolor{teal}{C})$

$S \rightarrow$

$@(F(\textcolor{red}{s}(@(\textcolor{red}{G}(F(\textcolor{red}{s}(\textcolor{red}{s}(z))))), \textcolor{teal}{B}))), \textcolor{teal}{C})$

$F :: \text{tree} \rightarrow \text{tree} \rightarrow \text{tree}$

$F(x)(y) = \textcolor{teal}{D}(x, y)$

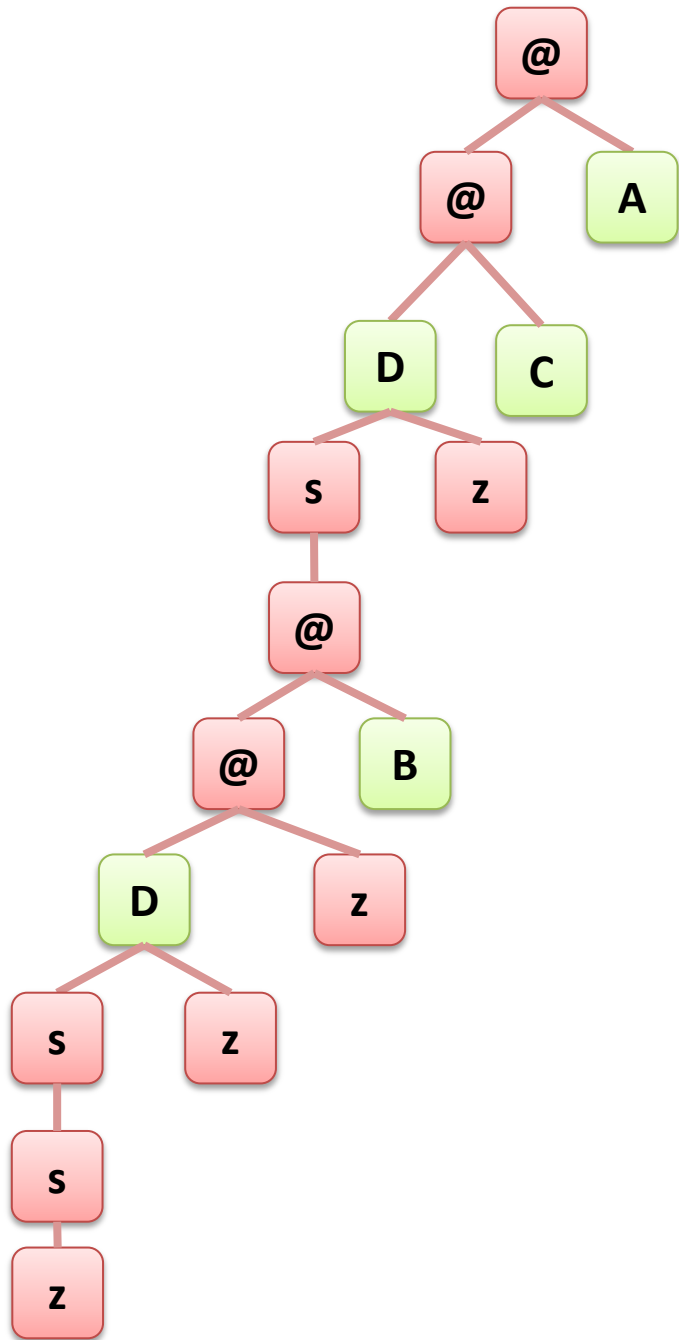
$F(x) \rightarrow \textcolor{teal}{D}(x, \textcolor{red}{z})$

$G :: (\text{tree} \rightarrow \text{tree}) \rightarrow \text{tree} \rightarrow \text{tree}$

$G(u)(x) = u(x)$

$G(u) \rightarrow @(\textcolor{red}{u}, \textcolor{red}{z})$

Example



$$I \rightarrow @(\textcolor{red}{S}, \textcolor{green}{A})$$

$$S \rightarrow @(\textcolor{red}{F}(\textcolor{red}{s}(@(\textcolor{green}{G}(\textcolor{red}{F}(\textcolor{red}{s}(\textcolor{red}{s}(\textcolor{red}{z})))), \textcolor{green}{B}))), \textcolor{green}{C})$$

$$F(x) \rightarrow \textcolor{green}{D}(x, \textcolor{red}{z})$$

$$G(u) \rightarrow @(\textcolor{red}{u}, \textcolor{red}{z})$$