Expressive Power of Safe HORS

Examined Through Decomposition of
Higher Order Programs to Garbage Free 1^{st} Order Form

Kazuhiro Inaba
Joint work with Sebastian Maneth

at *Shonan Meeting* on
Automated Techniques for Higher-Order Program Verification
2011

Background

- HORS (Higher Order Recursion Scheme) is very powerful and expressive.
- **n-EXPTIME hard** problems!

Computational Complexity w.r.t. Grammar Size and Data Size

- MSO on words/trees:
  - Emptiness checking is **non elementary (HYPEREXP)** for the size of the formula.
  - The class of languages it represents is **regular**.
    - **O(n)** time, **O(1)** space membership wrt the word length

"MSO on words is a verrrrrrrry concise representation for relatively simple languages."

How about HORS?

- HORS:
  - Emptiness, Model Checking, Containment by Regular Languages, ... are **n-EXPTIME hard**.
  - What is known about the languages it describes?
    - The class of languages it represents is **????**.
    - **????** time, **????** space membership wrt the word length.

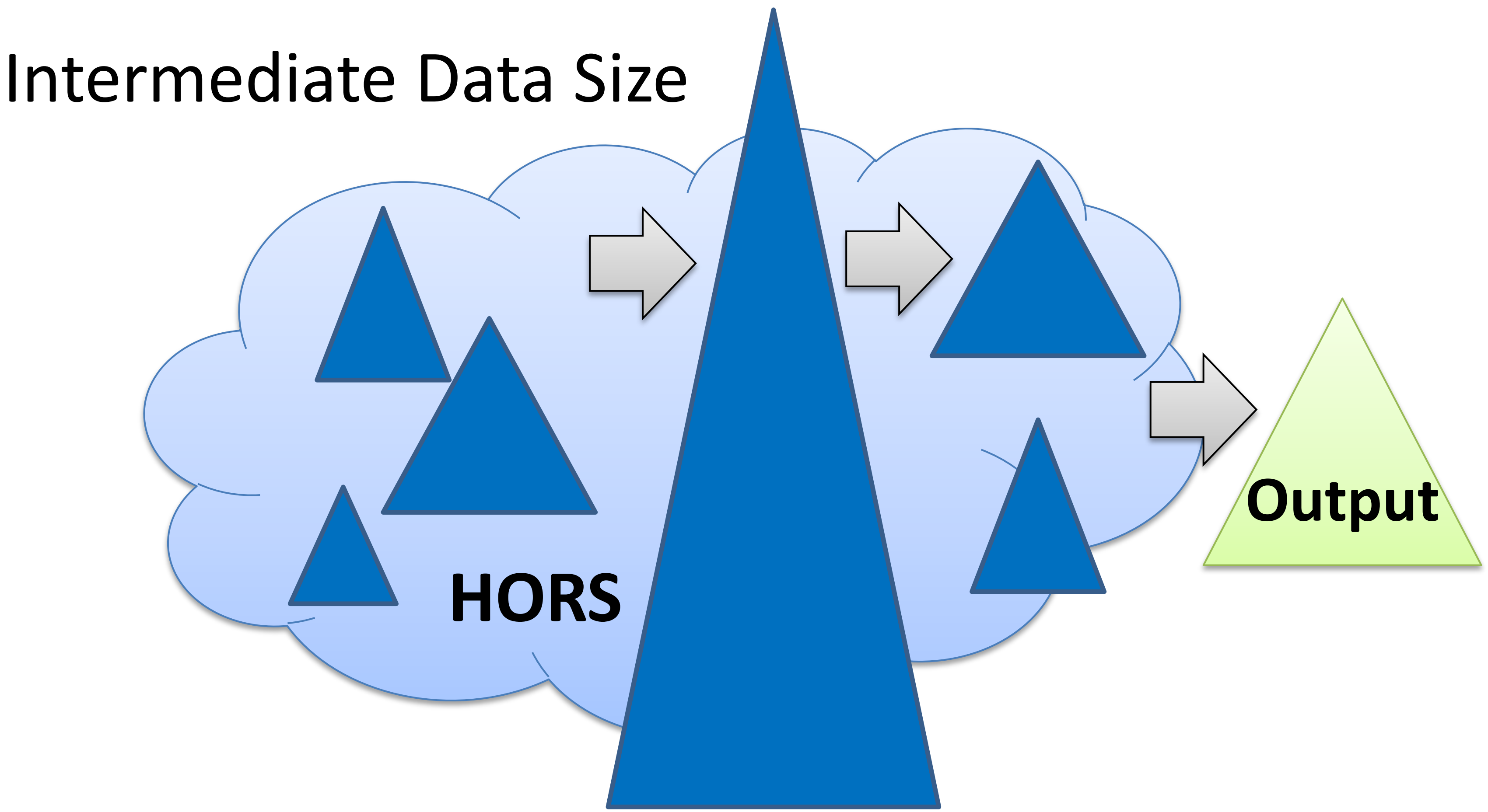
[Greibach 70]

> Aho and Ullman [3] have shown that the indexed languages can be characterized by AFAs whose data structure is a pushdown store of pushdown stores, with an added duplicate order which replicates the topmost store. They call these degree 2 pushdown stores and show that this idea can be extended to degree n, for any n, and that all these families have decidable emptiness problems and are contained in the context-sensitive languages.
>
> 3. A. V. Aho and J. Ullman, private communication.

Today's talk verifies the statement
(even for wider class of languages).

[Gr70] S. A. Greibach, "Full AFLs and Nested Iterated Substitution", *Inf. Ctrl. 16*

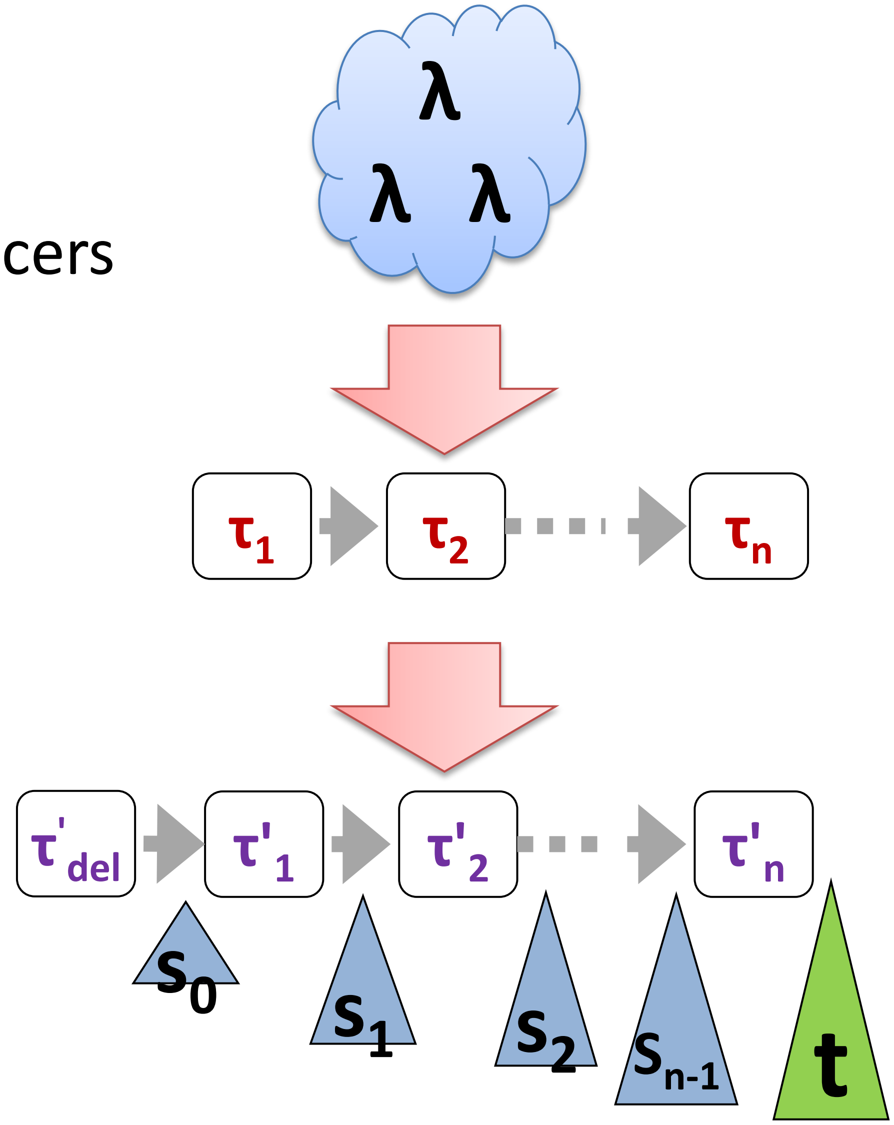
Our Approach



If they are at most of size M at any point, O(M) space & O(2^M) time.

Outline of This Talk

- Target Language
  - Higher-order Tree Transducers
- 1st-order Decomposition
  - Sketch of the construction
- Garbage Free Form
  - Derived consequences
  - Sketch of the construction



HTT [Engelfriet&Vogler 88]

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

```
Mult :: Tree → Tree
Mult(Pair(x1,x2))   ➔  Iter(x1)(Add(x2))(Z)

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

Add :: Tree → Tree → Tree
Add(S(x))(y)   ➔  Add(x)(S(y))
Add(Z)(y)      ➔  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 ⊥)

```
 Subseq :: Tree → Tree
Subseq(Cons(x,xs)) → Cons(x, Subseq(xs))
                   // Subseq(xs)
Subseq(Nil)      → Nil
Subseq(Other)    → ⊥
```

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`→`Tree` functions representable by HTTs of order ≦ n.
  - {Subseq} is 0-HTT, {Mult, Iter, Add}∈2-HTT

```
Subseq :: Tree → Tree
```

```
Mult :: Tree → Tree
Iter :: Tree → (Tree → Tree) → Tree → Tree
Add  :: Tree → Tree → Tree
```

Order-n to Order-1

THEOREM [EV88] [EV86]

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

n-th order tree transducer is representable by a n-fold composition of 1st-order tree transducers. ("= or ⊊ ?" is left open, as far as I know.)

[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-HTT = 1-HTT ∘ (n-1)-HTT

Idea:

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

```
F :: Tree → Tree→Tree
F(Z)(y) → S(S(y))
```

⇒

```
F :: Tree → Tree
F(Z) → S(S(Y))
```

Represent 1st-order application symbolically, too.

```
… → F(x)(Z)
```

⇒

```
… → @(F(x), Z)
```

Proof: n-HTT = 1-HTT ∘ (n-1)-HTT

Represent 1st-order things symbolically.

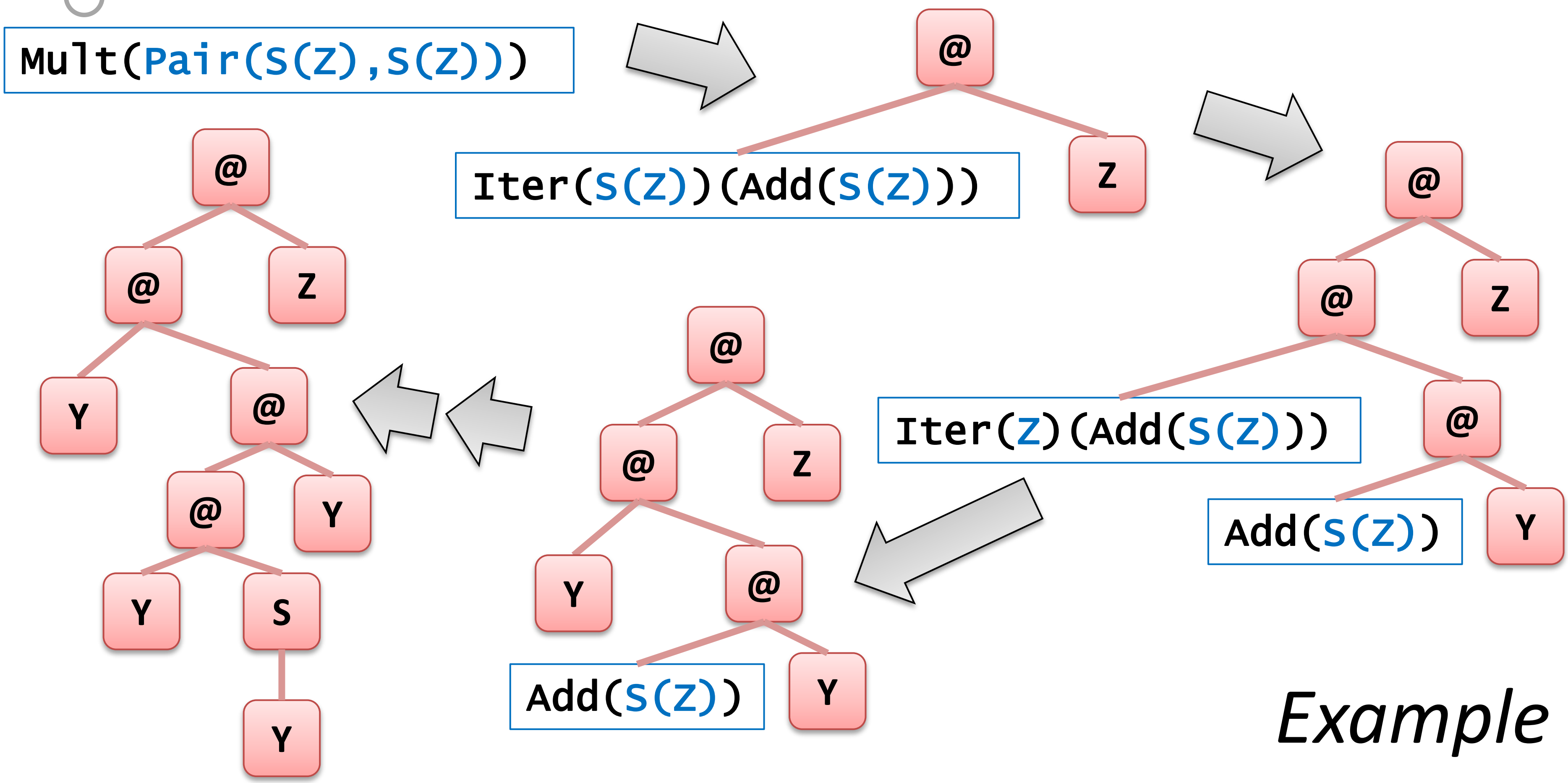
```
 F :: Tree → Tree
F(Z) ➔ S(S(Y))
```

```
… ➔ @(F(x), Z)
```

Then a 1-HTT performs the actual "application".

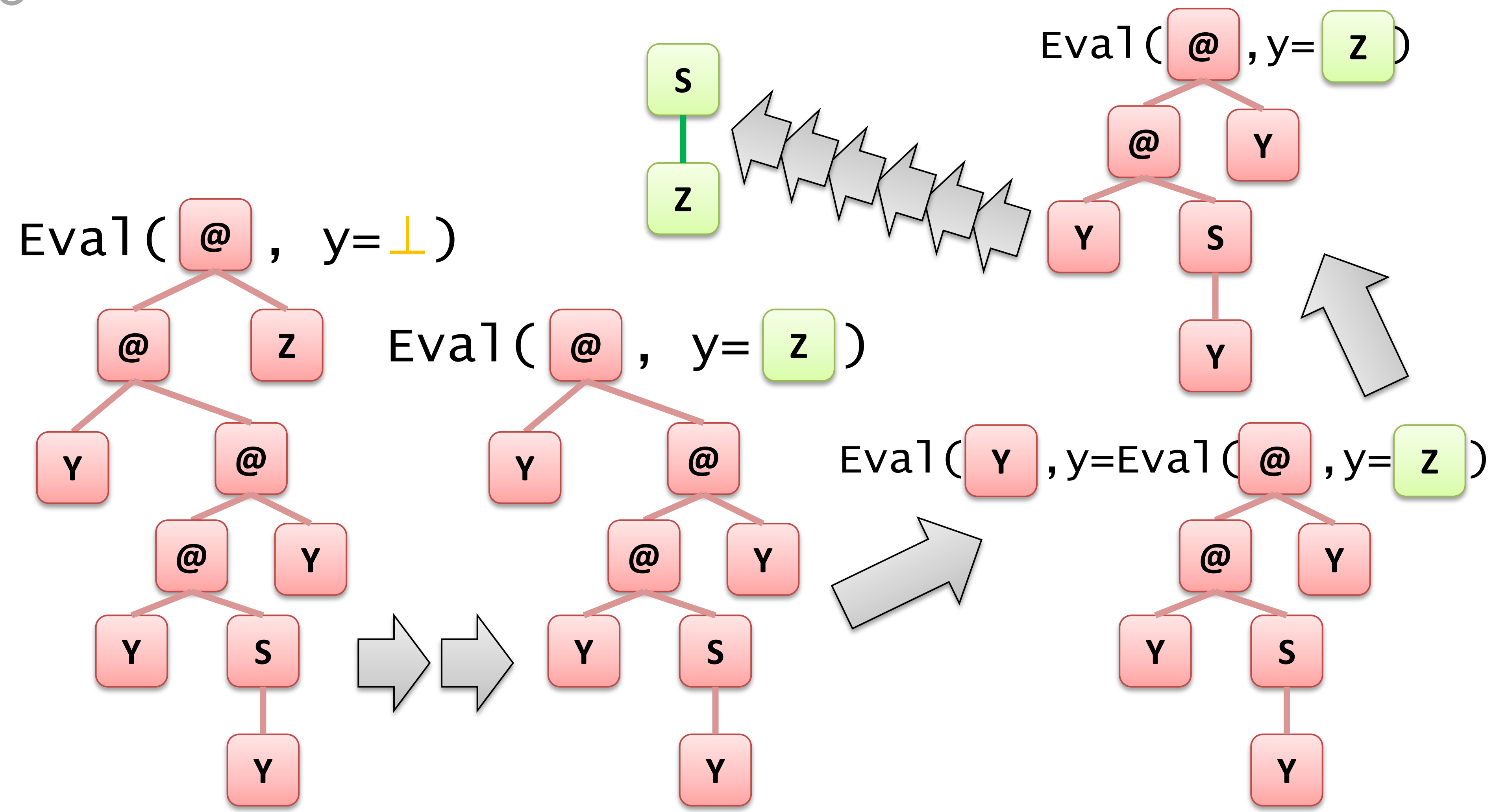
```
Eval(@(f, b))(y) ➔ Eval(f, Eval(b)(y))
Eval(Y)(y)       ➔ y
Eval(S(x))(y)    ➔ S(Eval(x)(y))
Eval(Z)(y)       ➔ Z
```

```
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
```



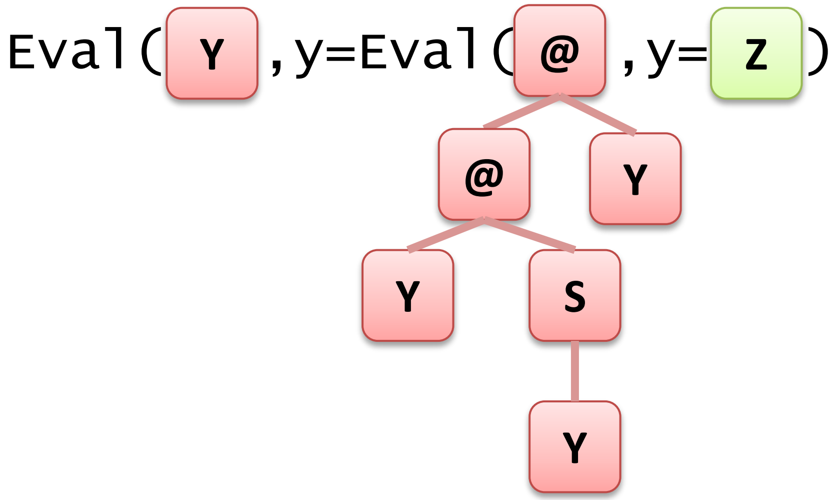
Example

```
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
```



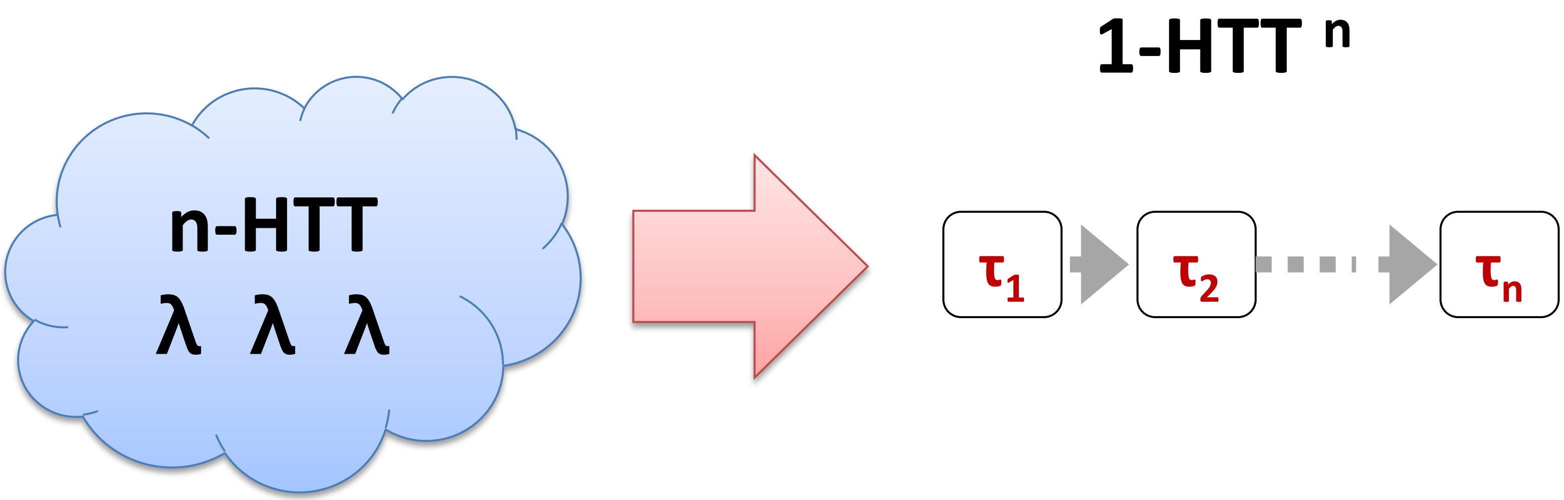
Why That Easy

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

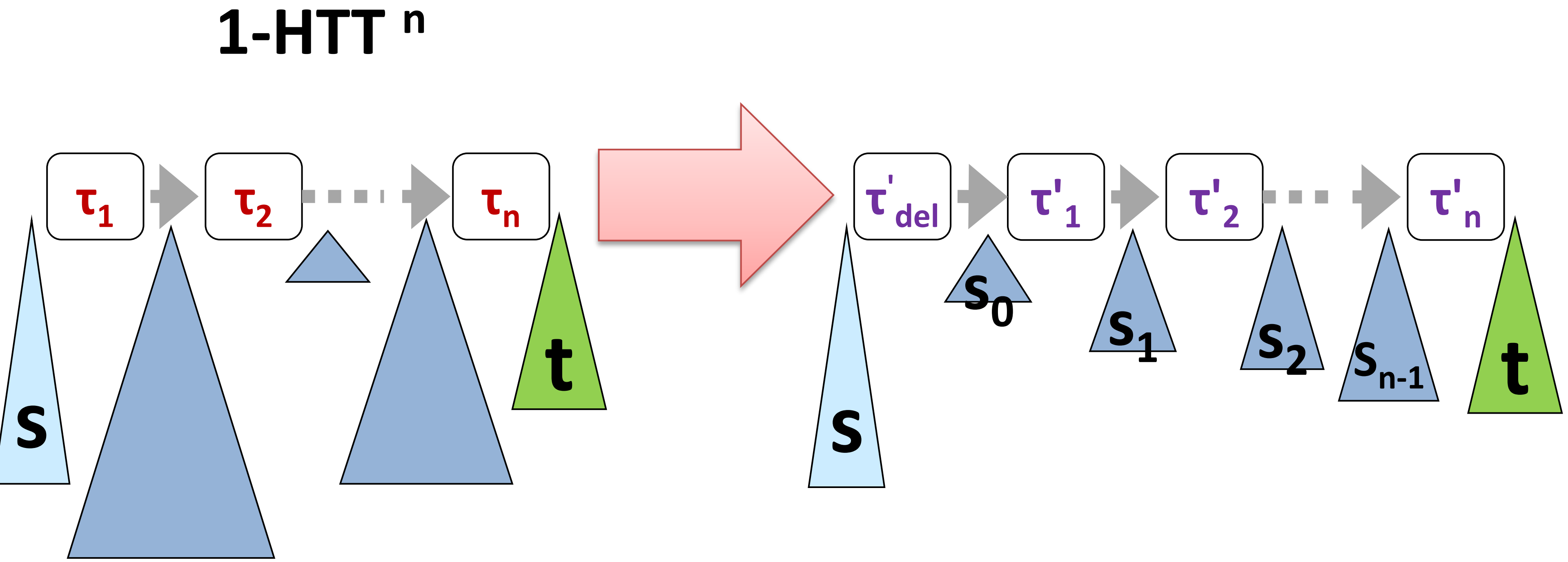


[BO09] W. Blum and C.-H. L. Ong, "The Safe Lambda Calculus", *LMCS 5*

Now, Decomposed.



Next, Make Intermediate Trees Small.

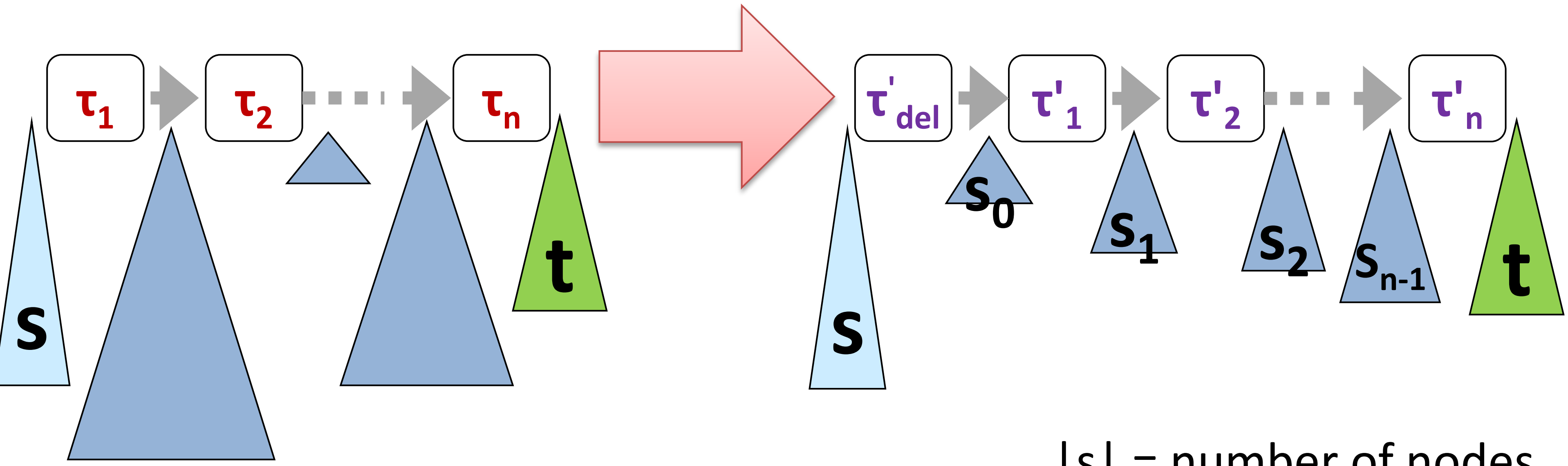


THEOREM [I. & Maneth 08] [I. 09](+ improvement)

$\forall \tau_1, \ldots, \tau_n \in$ 1-HTT, $\exists \tau'_{del} \in$ 0-LHTT, $\tau'_1, \ldots, \tau'_n \in$ 1-HTT,

for any $(\tau_n \circ \ldots \circ \tau_1)(s) \ni t$,

there exist $\tau'_{del}(s) \ni s_0$, $\tau'_i(s_i) \ni s_{i+1}$, $|s_i| \leqq |s_{i+1}|$, $s_n = t$.



|s| = number of nodes

[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*

Consequences : Range Membership

Membership problem for
the class **Range(1-HTT ^{n})** of languages is

- in DLINSPACE
- in NP

That is, given **$(\tau_n \circ \ldots \circ \tau_1)$** and **$t$**, we can determine

“$\exists$ s. $(\tau_n \circ \ldots \circ \tau_1)(s) \ni t$”

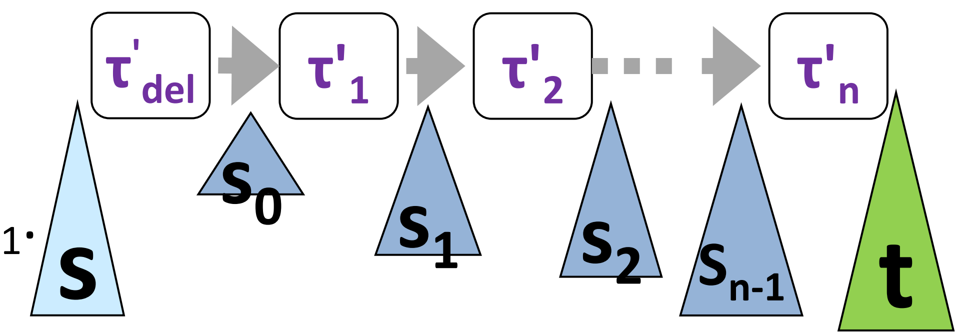
in $O(\, f(|\tau_1|+\ldots+|\tau_n|) \cdot |t| \,)$ space and
in $O(\, g(|\tau_1|+\ldots+|\tau_n|) \cdot \mathrm{poly}(|t|) \,)$ nondeterministic time.

Consequences : Range Membership

Membership problem for
the class **Range(1-HTT ^{n})** of languages is
- in DLINSPACE
- in NP

PROOF

Guess (in NP) or
exhaustively try (in DLINSPACE)
all the intermediate trees: s_0 ... s_{n-1}.



Then check **Range(τ'$_{del}$)∋s$_0$** and **τ'$_i$(s$_i$)∋s$_{i+1}$**,
both turn out to be feasible in DLINSPACE ∩ NP.

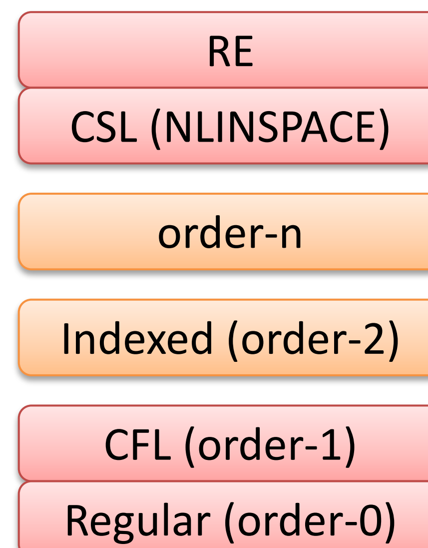
Consequences : Range Membership

Membership problem for
the class **Range(1-HTT ^{n})** of languages is

- in DLINSPACE
- in NP

COROLLARY

Higher-order *safe* recursion scheme, also known as *OI-hierarchy, HO-PDA language, Maslov hierarchy, generalized indexed language*, etc., is Context-Sensitive.



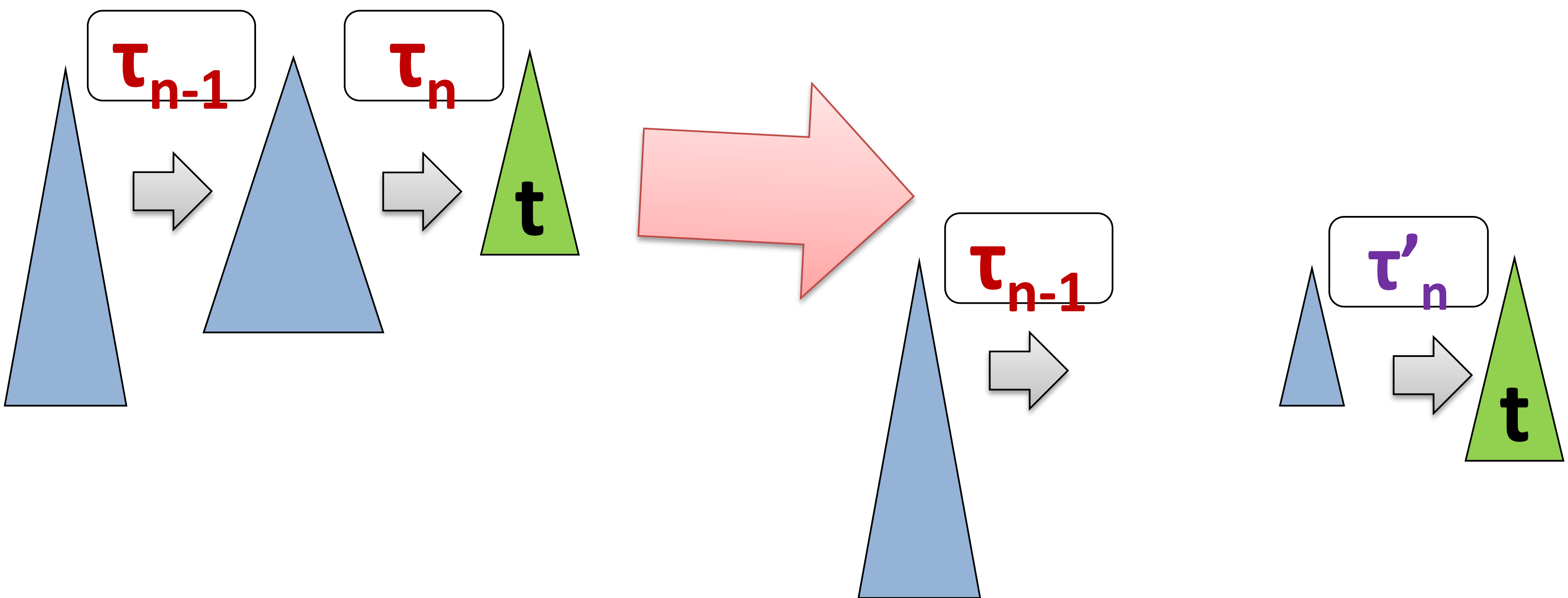
Consequences : Linear-Size Inverse

For all $\tau_n \circ \ldots \circ \tau_1 \in$ 1-HTT^n , $t \in$ Range($\tau_n \circ \ldots \circ \tau_1$)
there exists s such that
$f(s) \ni t$ and $|s| < h(|\tau_n \circ \ldots \circ \tau_1|) \cdot |t|$

COROLLARY (by our constructive proof)
Right inverse of **1-HTT^n** is computable in DLINSPACE∩NP.

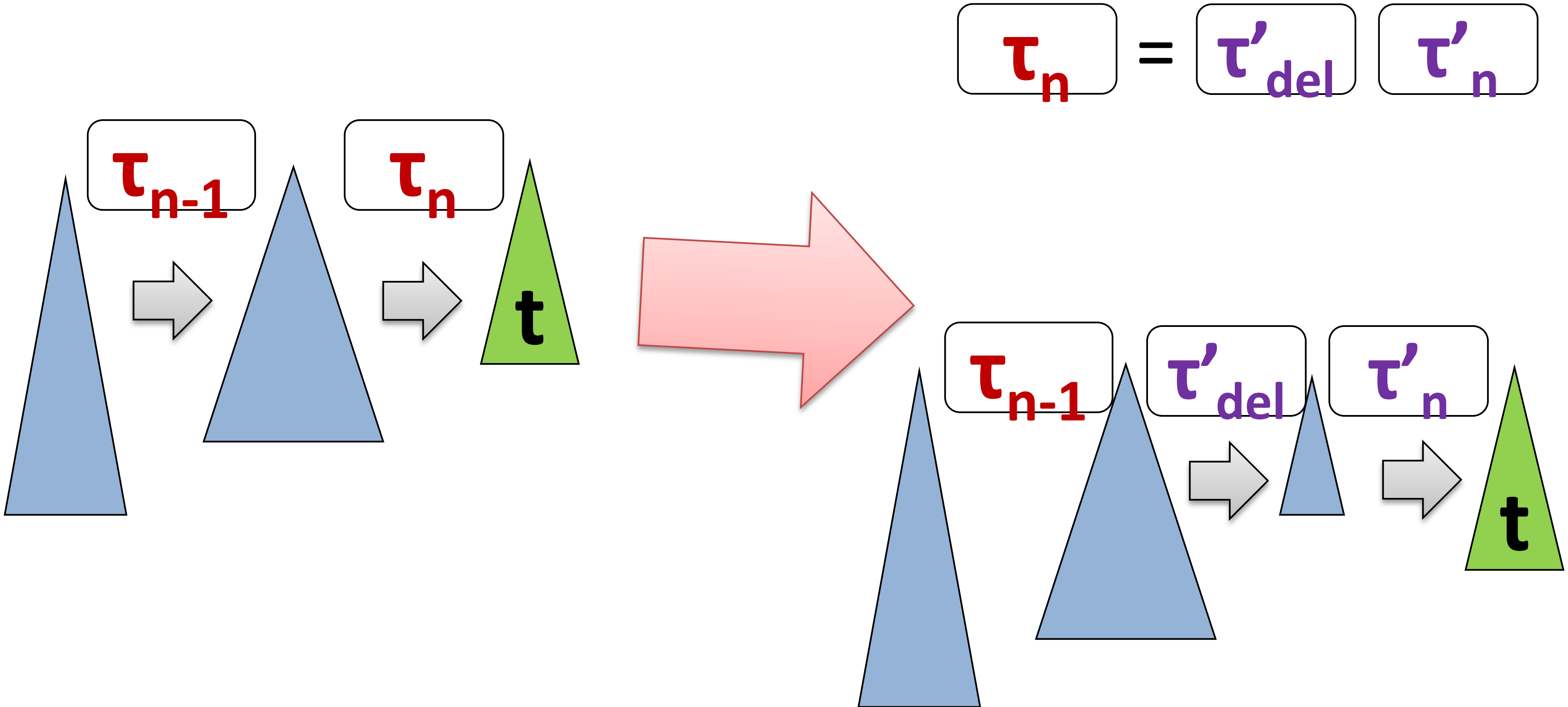
How to Construct the “Garbage-Free” Form

Make each 1-HTT “productive”



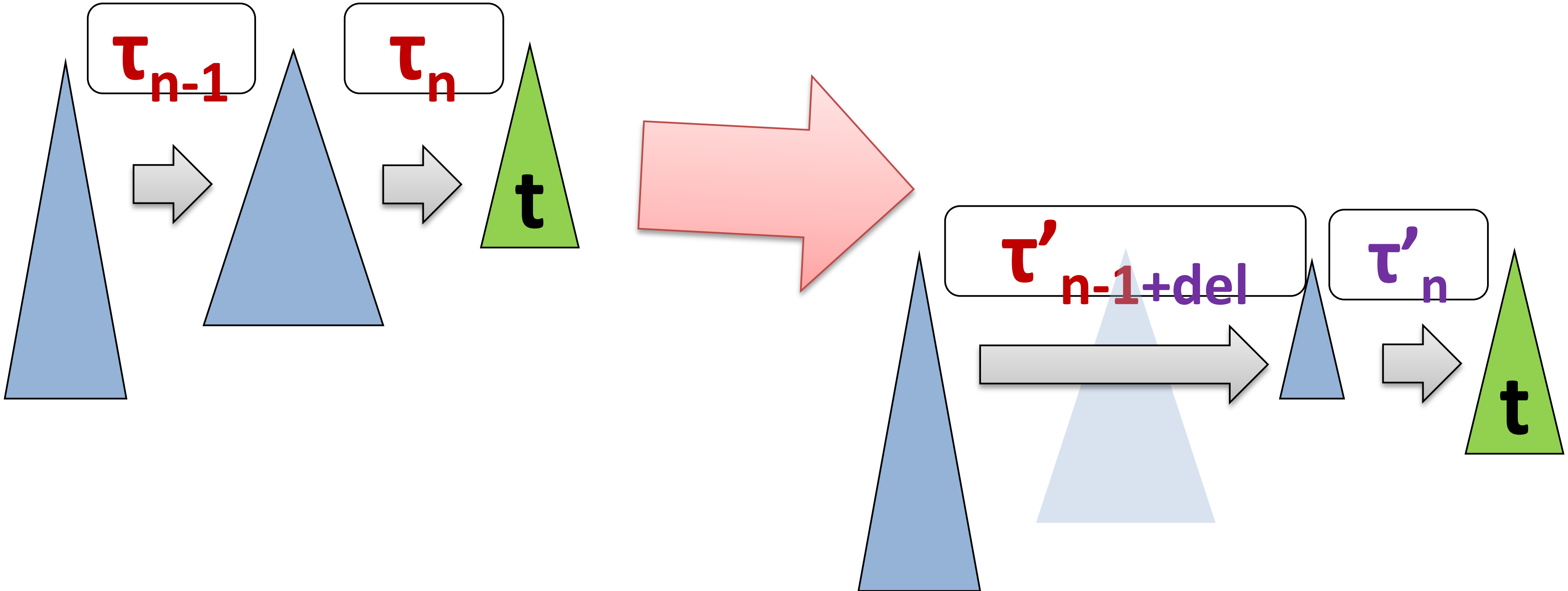
How to Construct the “Garbage-Free” Form

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

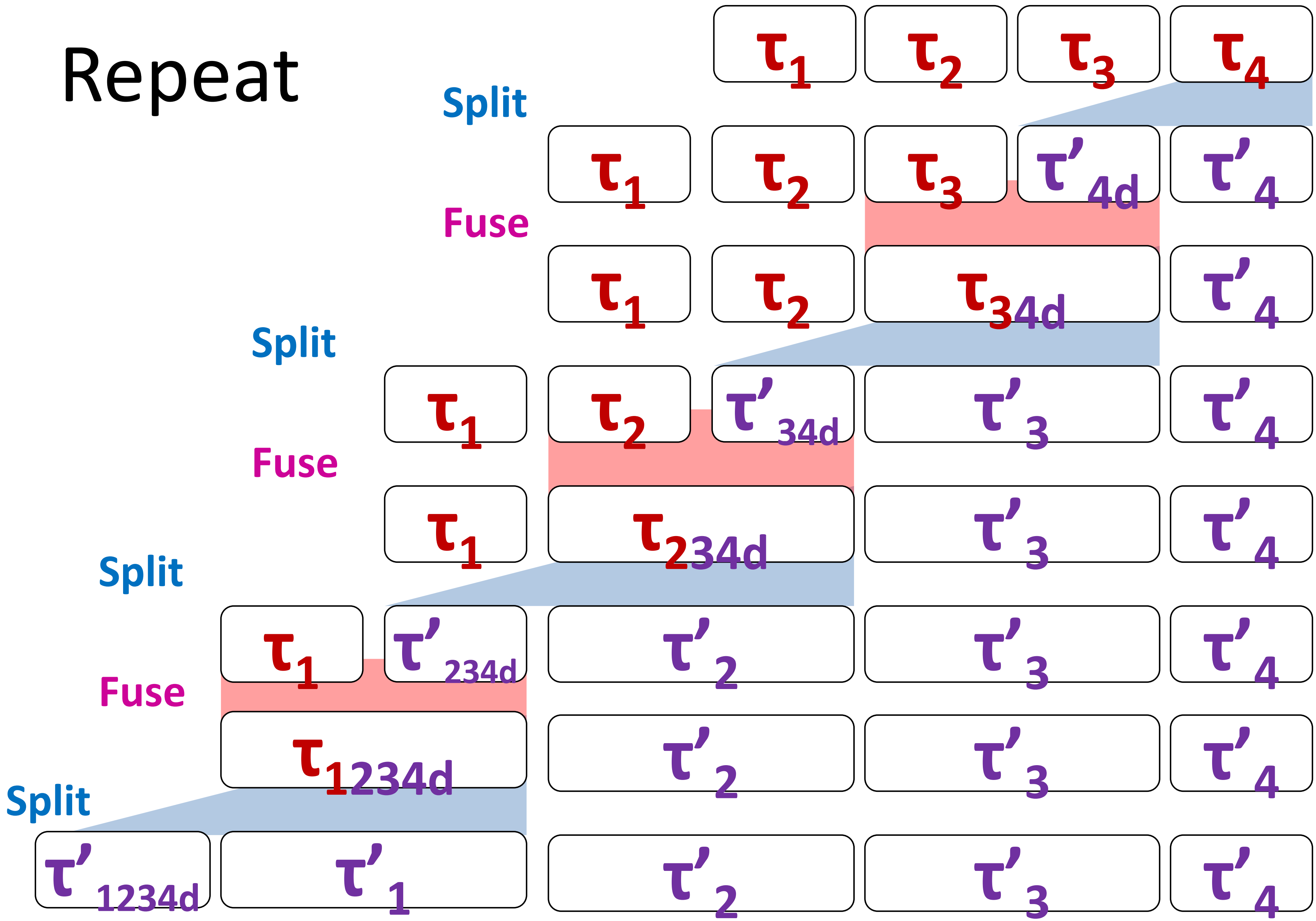


How to Construct the "Garbage-Free" Form

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



Repeat

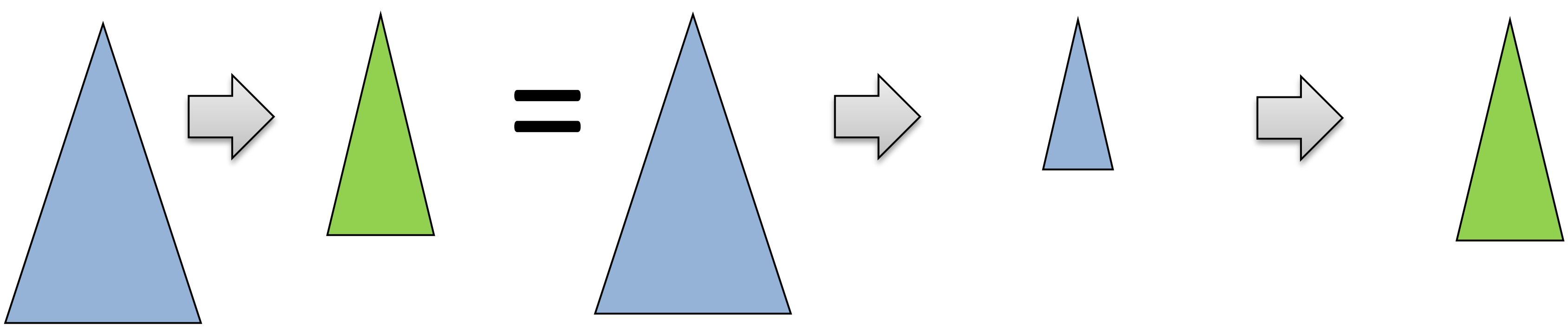


τ1
τ2
τ3
τ4
Split
τ1
τ2
τ3
τ'4d
τ'4
Fuse
τ1
τ2
τ34d
τ'4
Split
τ1
τ2
τ'34d
τ'3
τ'4
Fuse
τ1
τ234d
τ'3
τ'4
Split
τ1
τ'234d
τ'2
τ'3
τ'4
Fuse
τ1234d
τ'2
τ'3
τ'4
Split
τ'1234d
τ'1
τ'2
τ'3
τ'4

Key Part

Separate the “deleting” transformation

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

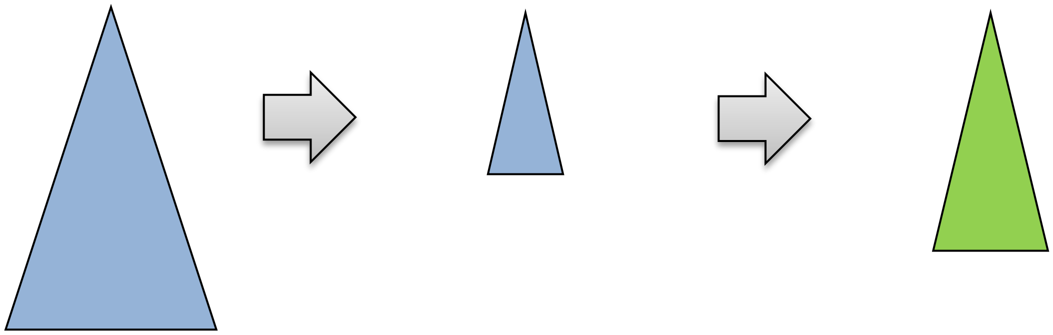


Key Part

Slogan: **Work on every node**

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

τ'_{del} ; τ'_n



Work on Every Node ⇒ Visit All Nodes

Deleting HTTs

τ_n

$$G(Z)(y_1) \rightarrow Z \;//\; y_1$$

$$F(S(x_1,x_2)) \rightarrow \underline{F(x_1)} \;//\; \underline{F(x_2)} \;//\; G(x_1)(\underline{F(x_2)})$$

may not recurse down to a subtree.

Work on Every Node ⇒ Visit All Nodes

$F(S(x_1,x_2)) \rightarrow G(x_1)(F(x_2))$ τ_n

Nondeterministically delete every subtree!

τ'_{del}

$Del(S(x_1,x_2)) \rightarrow$
$S12(Del(x_1),Del(x_2)) \mathbin{/\!/} S1\_(Del(x_1))$
$\mathbin{/\!/} S\_2(Del(x_2)) \mathbin{/\!/} S\_\_()$

$F(S12(x_1,x_2)) \rightarrow G(x_1)(F(x_2))$
$F(S1\_(x_1)) \rightarrow G(x_1)(\bot)$
$F(S\_2(x_2)) \rightarrow \bot$
$F(S\_\_()) \rightarrow \bot$

τ'_n

At least one choice of nodeterminism “deletes correctly”.

Work on Every Node ⇒ Work on Leaf

Erasing HTTs

```
F(S(x)) → G(x)(Z)
G(Z)(y) → y
```

τ_n

may be idle at leaves.

Work on Every Node ⇒ Work on Leaf

Erasing HTTs

```
F(S(x)) → G(x)(Z)
G(Z)(y) → y
```

τ_n

Inline Expansion

```
F(S(Z)) → Z
```

τ'_n

Work on Every Node
⇒ Work on Monadic Nodes

Skipping HTTs

τ_n

$F(S(x))(y_1, y_2, y_3) \rightarrow F(x)(y_2, y_3, y_1)$

$F(Z)(y_1, y_2, y_3) \rightarrow \mathrm{Done}(y_1, y_2, y_3)$

are good at juggling.

Work on Every Node ⇒ Work on Monadic Nodes

Skipping HTTs

τ_n

~~$F(S(x))(y_1,y_2,y_3) \rightarrow F(x)(y_2,y_3,y_1)$~~

$F(Z)(y_1,y_2,y_3) \rightarrow Done(y_1,y_2,y_3)$

Nondeterministic deletion again.

Remember how argugments would've been shuffled.

$F(Z123)(y_1,y_2,y_3) \rightarrow Done(y_1,y_2,y_3)$

$F(Z231)(y_1,y_2,y_3) \rightarrow Done(y_2,y_3,y_1)$

$F(Z312)(y_1,y_2,y_3) \rightarrow Done(y_3,y_1,y_2)$

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** ≦ **Output size**.

- For any tree, **#others < #leaf** ≦ **Output size**.
- Add: **#leaf + #monadic + #others** ≦ **Output size***2

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

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

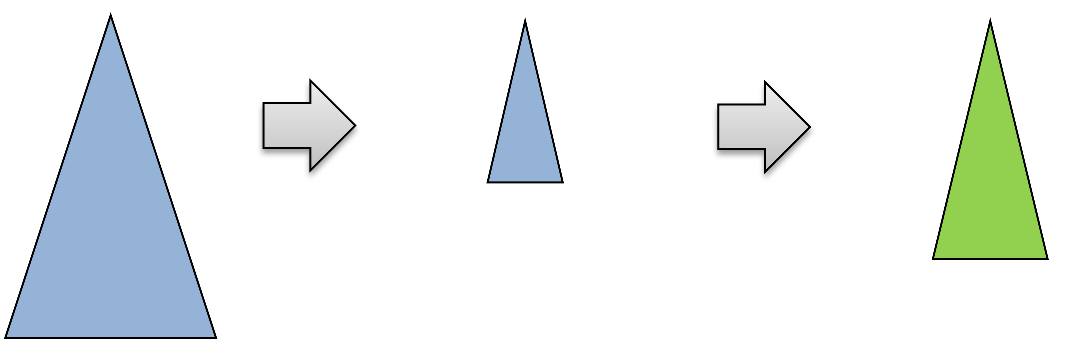
- **Input size < Output Size * 2**

```
Fr(Bin(x1,x2))(y) → Fr(x1)(Fr(x2)(y))
Fr(A)(y) → A(y)
Fr(B)(y) → B(y)
```

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

Done!

τ'_{del} ; τ'_{n}



Summary

- Order-n HTT → (Order-1 HTT)^n
- Garbage Free Form
  - L(Safe-HORS) is context-sensitive.
- Future Direction
  - Extend it to Unsafe HTT
  - *Or, use it for proving safe $\subsetneq$ unsafe*