Tree transducers of linear size-to-height increase

(and the additive conjunction of linear logic)

Luc Dartois
Université Marie et Louis Pasteur
France

Lê Thành Dũng (Tito) Nguyễn
CNRS & Aix-Marseille Université
France

Charles Peyrat
Université Paris-Est Créteil
France

Abstract

We investigate a natural generalization to trees of Hennie machines, a known automaton model for regular string functions. Tree-to-tree Hennie machines are tree-walking tree transducers with the ability to rewrite the node labels of their input tree, subject to a bounded visit restriction. Interestingly, they do not merely compute regular tree functions (i.e. MSO transductions), but a larger class of functions with linear size-to-height increase (LSHI).

We prove that this class sits between LSHI macro tree transducers (MTTs) and MSO set interpretations. To argue for its robustness, we show that it is closed under precomposition (resp. postcomposition) by MTTs of linear size (resp. height) increase. As a consequence, it contains the entire composition hierarchy of MTTs of linear height increase; we also prove that this composition hierarchy is strict.

Finally, we give an alternative characterization of this function class based on a λ-calculus with linear types. The key difference with similar characterizations of MSO transductions is the use of additive tuples in the encoding of output trees. Our equivalence proof, using game semantics / geometry of interaction, is heavily inspired by an analogous result on higher-order recursion schemes.

Keywords

macro tree transducers, λ-calculus, MSO logic, game semantics

1 Introduction

Macro tree transducers (MTTs) are an automata model for tree transformations first introduced by Engelfriet and Vogler in the 1980s [21, 22]. There have been many developments concerning MTTs since then; one starting point of this paper is the recent:

Theorem 1.1 (Gallot, Maneth, Nakano & Peyrat [28]). *Given a* macro tree transducer *computing a function f on ranked trees, one can decide whether it has:*

linear height increase (LHI): $\mathrm{height}(f(t)) = O(\mathrm{height}(t))$
linear size-to-height increase (LSHI): $\mathrm{height}(f(t)) = O(|t|)$

Compare this to an important result from a quarter century ago:

Theorem 1.2 (Engelfriet & Maneth [20]). *Given an* MTT *computing a function f on ranked trees, one can decide whether it is of* linear size increase (LSI). *Furthermore, if it is, then one can translate it to an* Monadic Second-Order transduction *that defines f. (Conversely, every* MSO *transduction can be translated into an* MTT.*)*

This seems more informative than the statement of the analogous Theorem 1.1: in addition to decidability, it gives us an equivalence between LSI-MTTs and a completely different formalism for describing functions from trees to trees, based on MSO logic. This equivalence is a consequence of the strategy for decidability: it comes from boundedness properties on the runs of (suitably normalized) MTTs that characterize the LSI property.[1]

Similar boundedness properties arise in the decision procedure for Theorem 1.1. Hence the question: what consequences can we draw concerning MTTs of linear (size-to-)height increase? We show that they can be translated to MSO *set interpretations* (MSO-SI), an extension of MSO transductions from the 2000s [10] recently studied[2] by Lhote et al. [24, 27]. More than that: to prove this comparison in expressive power, we find it enlightening to consider an intermediate model of independent interest:

$$\text{LSHI-MTT} \subset \textbf{tree-to-tree Hennie machines} \subset \text{MSO-SI}$$

In fact, this new machine model is our actual main topic here. Recall that a *Hennie machine* [17, Section 7] is a Turing machine:

- with a single read/write input tape and a write-only left-to-right output tape;
- whose tape head must stay within the original bounds of the input string (that is, the only usable memory is the space initially occupied by the input);[3]
- visiting each cell of the tape *a bounded number of times*.[4]

In other words, it is a bounded-visit two-way transducer extended with the ability to use the positions of input letters as mutable memory cells. In this informal description, replacing the two-way string transducer by its usual generalization to trees, namely the *tree-walking tree transducer* (see e.g. [11, Chapter 8]), results in the notion of *tree-to-tree Hennie machine* (THM).

Claim 1.3. THMs have linear size-to-height increase.

Proof idea. The bounded visit property forces THMs to terminate in linear time. In the tree-to-tree case, the branches are produced in parallel by forking processes, each taking linear time. We give a more rigorous proof in Section 3.2. □

Just as recent works on polyregular functions [4, 37] seem to answer "what string-to-string functions of polynomial size increase can be computed by 'reasonable' transducers", we envision our

[1] Reducing various quantitative decision problems to boundedness questions is a general idea with wide-ranging applicability in automata theory, see e.g. [1, 9]. Note also that using a quite different approach, Gallot, Lhote and Nguyễn [27] gave alternative proofs of Theorem 1.2 and of a generalization of the LSHI half of Theorem 1.1.

[2] Let us also mention the work of Bojańczyk et al. [5, 6] on string-to-string MSO *interpretations*, an intermediate case between transductions and set interpretations. They are one of the equivalent characterizations of *polyregular functions*.

[3] Actually, this condition does not appear in Hennie's original work [34, Section II.C] on language recognition by bounded-visit Turing machines, nor in its subsequent extension by Rajlich [48] to string outputs. However the name "Hennie machine" seems be consistently used to refer to a machine model that satisfies this restriction on the tape head, starting with its introduction by Engelfriet and Hoogeboom [17, §7] (for later examples, cf. [31–33]).

[4] Guillon, Pighizzini, Prigioniero and Průša [31–33] define Hennie machines (as language acceptors) by a linear-time restriction instead of a bounded-visit restriction, but these are effectively equivalent, see [32, Lemma 1] or [34, proof of Theorem 3].

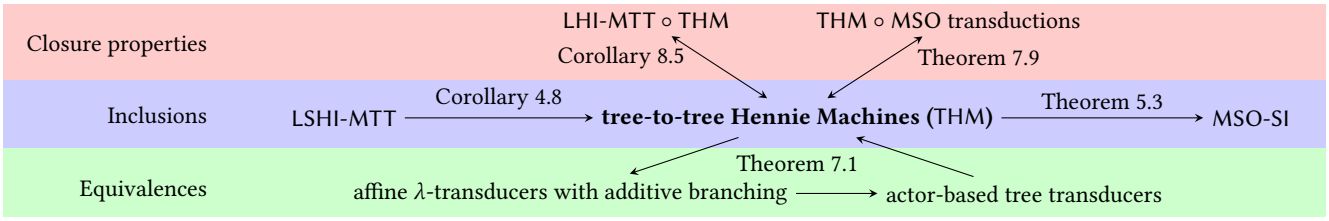


Figure 1: Summary of the main results. An arrow indicates inclusion of function classes.

study of tree-to-tree Hennie machines as a contribution towards understanding "what LSHI functions can be computed by 'reasonable' transducers". The class of functions computed by THMs does not seem to coincide with any previously defined formalism.[5] Yet we argue that it is robust, by exhibiting closure properties and alternative characterizations. These results are summarized in Figure 1; the rest of this introduction discusses them further.

Closure properties. We show that the functions computed by THM

(1) are *regularity reflecting*, i.e. the inverse image of a regular tree language is effectively regular (Corollary 8.1);
(2) are closed under precomposition by MSO tree transductions (i.e. macro tree transducers of linear size increase);
(3) are closed under postcomposition by LHI-MTTs.

As a consequence, THMs subsume the composition hierarchy of LHI-MTTs, whose strictness we also show (Corollary 9.1).

The proof of the third item relies on characteristics of LHI-MTTs uncovered in Gallot et al.'s proof of Theorem 1.1, as discussed at the beginning. We relate these characteristics to a "narrow-visit" condition on THMs. Narrow-visit implies LHI, and a large part of the proof consists in showing: [narrow-visit THM] ∘ THM = THM (Theorem 8.4).

For the first two items above, it is convenient to use our alternative characterization, due to its good compositional properties.

λ-calculus characterization. We prove that THMs are expressively equivalent to *affine*[6] *λ-transducers with additive branching*.

This is a formalism in the style of linear higher-order tree transducers [26, 46] or "implicit automata" [44, 45, 47]. Compared to these works, we extend the type system of these affine λ-transducers with the *additive conjunction* '&' of linear logic, and use it[7] in the encoding of output trees to group the children of a node. Intuitively, this means that the affineness restrictions apply separately for each output branch (just like the bounded-visit restriction of our Hennie machines) rather than globally.

This is a natural idea in the context of the previous work of Clairambault and Murawski [8]. They prove that two formalisms generating infinite trees are expressively equivalent: one based on affine λ-calculus and using additive branching, the other based on automata whose infinite memory is subject to a bounded-visit condition. Our own proof is a direct adaptation of theirs.

- To compile λ-transducers into THMs, we reuse a form of finite-memory geometry of interaction[8] developed in [8]. This involves a detour through yet another model of computation, based on communicating finite-state processes that we call "actors".[9]
- The converse translation is more technically elementary, but of broader significance in the grand scheme of things: it gives us regularity reflection. In a paper closely related to our work [2], some colleagues[10] leverage this result to prove that string-to-string MSO-SI are regularity-reflecting — with the side effect of settling an old open problem on automatic structures, cf. [24, §1]. One initial impetus for our work, actually, was to provide this piece of the proof of [2]. The use of "yield-Hennie machines" in [2] led us to introduce and study tree-to-tree Hennie machines more generally, with an independent motivation coming from the line of work on MTTs described previously.

Remark 1.4. Without the affineness constraint, simply typed λ-transducers subsume virtually all transducer models (for both strings and trees) from the literature, see e.g. [44, §1.4.1].

Regularity reflection for λ-transducers (cf. [29, §5.3.1] or [44, Corollary 1.4.1][11]) is a straightforward application of the naive set-theoretic semantics of the simply typed λ-calculus: if we interpret the base type as a finite set of states, then all higher-order types are also interpreted as finite sets, and λ-terms are interpreted as elements of these sets. This technically simple yet conceptually deep idea underlies several connections between automata theory and λ-calculus, from the use of finitary denotational semantics in higher-order model checking [30, 51] to the theory of regular languages of λ-terms [42, 43, 49]. Let us note that Clairambault and Murawski's aforementioned [8] comes from a line of work motivated by higher-order model checking [7, 41].

Plan of the paper. In Section 2 we fix some notations and recall some generalities on tree automata/transducers. In Section 3 we introduce unrestricted tree-to-tree Hennie machines, the bounded-visit restriction (defining THMs) and the narrow-visit restriction, as well as some features extending THMs. The three sections that follow are independent; we translate:

[5] Though in the tree-to-string case, it collapses to MSO transductions (Corollary 5.4). In the string-to-string case this was one of Engelfriet and Hoogeboom's main results on Hennie machines [17, Theorem 26].
[6] In the λ-calculus, a linear (resp. affine) function uses its argument exactly (resp. at most) once. Affineness is closer to the "single use restrictions" in automata theory.
[7] An idea originating in the work of Clairambault, Grellois and Murawski [7].
[8] The name "Geometry of Interaction" refers to a family of semantics (both operational and denotational) of linear logic and λ-calculi, closely related to game semantics.
[9] They were originally called "transducers" [8, §3.2], but we avoid this name here to avoid confusion. The name "actor" alludes both to the geometry of inter*action* and to the more general "Actor model" of message-passing computation [12, 35, 39].
[10] Anonymous for the sake of double-blind review.
[11] Corollary of Hillebrand and Kanellakis's [36, Theorem 3.4], which is the origin of Nguyễn and Pradic's aforementioned "implicit automata" research programme.

- LSHI-MTTs to THMs in §4;
- THMs to MSO set interpretations in §5;
- actor-based tree transducers to THMs in §6.

In §7 we use actors to relate THMs and λ-transducers, and deduce the closure by precomposition by MSO transductions. In §8 we look at postcomposition properties: we prove regularity reflection (via λ-transducers) and postcomposition by LHI-MTT. Finally, in §9 we study the LHI-MTT composition hierarchy.

2 Notations & preliminaries

A *ranked set* is a set Σ endowed with a function $\mathrm{rank}\colon \Sigma \to \mathbb{N}$. A finite ranked set is called a *ranked alphabet*. We use the notations $\Sigma^{(k)} = \{\sigma \in \Sigma \mid \mathrm{rank}(\sigma) = k\}$ and $\mathsf{R}_\Sigma = \max(\mathrm{rank}(\Sigma))$. We use $[k]$ as a shorthand for the set $\{1, \dots, k\}$.

A *tree* t over a ranked set Σ is defined by:

- a nonempty prefix-closed domain $\mathrm{Dom}(t) \subset [\mathsf{R}_\Sigma]^*$, whose elements are called the *nodes* of t;
- a map $\mathrm{lab}_t\colon \mathrm{Dom}(t) \to \Sigma$, the *labeling* of t, such that

$$\forall u \in \mathrm{Dom}(t), \quad \{i \in \mathbb{N} \mid ui \in \mathrm{Dom}(t)\} = [\mathrm{rank}(\mathrm{lab}_t(u))]$$

Thus, we also refer to $\mathrm{rank}(\mathrm{lab}_t(u))$ as the *rank of the node* u.

We write T_Σ for the set of trees over Σ, and when $\Sigma \cap Y = \varnothing$,

$$\mathsf{T}_\Sigma(Y) = \mathsf{T}_{\Sigma \cup Y} \quad \text{with } \mathrm{rank}(y) = 0 \ \forall y \in Y$$

In other words, the elements of $\mathsf{T}_\Sigma(Y)$ are like trees over Σ except that some leaves may have labels from Y.

The axioms entail that $\mathrm{Dom}(t)$ contains the empty sequence ε; it is called the *root* of t. The *leaves* are the rank 0 nodes, or equivalently, the maximal nodes for the prefix ordering. This ordering is also called the *ancestor* relation on nodes, and its converse is the *descendant* relation. For $u \in \mathrm{Dom}(t) \setminus \{\varepsilon\}$, we write $u{\uparrow}$ for the *parent* of u, i.e. the unique node for which $\exists i \in [\mathsf{R}_\Sigma] : u = (u{\uparrow})i$.

We write $\mathsf{D}_k = \{{\uparrow}\} \cup [k]$ for the set of directions we can move in from a node of arity k in a tree, where $\uparrow$ represents the upward direction towards the parent. According to the previous notations, if $u \neq \varepsilon$ is a rank k node of a tree t and $d \in \mathsf{D}_k$, then ud defines a node in t (one of the neighbors in u).

A *partial branch* b of t is the downward closure of a node s — its *endpoint* — in $\mathrm{Dom}(t)$ with respect to the prefix order. A branch, or *total branch* for disambiguation purposes, is a partial branch whose endpoint is a leaf.

The *size* $|t|$ of a tree t is the cardinality of its domain. Its *height* $\mathrm{height}(t)$ is the maximum size of its branches minus one.

For a node s in a tree t, the *subtree of* t *rooted at* s is defined by:

$$\mathrm{Dom}(t|_s) = \{u \in [\mathsf{R}_\Sigma]^* \mid su \in \mathrm{Dom}(t)\}$$
$$\mathrm{lab}_{t|_s}(u) = \mathrm{lab}_t(su)$$

The *substitution of* s *by* t' *in* t is obtained from t by replacing the subtree rooted at s by t' — formally:

$$\mathrm{Dom}(t[s \leftarrow t']) = (\mathrm{Dom}(t) \setminus \{s\} \cdot [\mathsf{R}_\Sigma]^*) \cup \{s\} \cdot \mathrm{Dom}(r)$$
$$\mathrm{lab}_{t[s \leftarrow t']}(u) = \begin{cases} \mathrm{lab}_{t'}(v) & \text{if } u = sv \text{ for some } v \in \mathrm{Dom}(r) \\ \mathrm{lab}_t(u) & \text{otherwise} \end{cases}$$

In this paper, it is often (but not always) the case that in such a substitution, s is a leaf; we then speak of a *first-order substitution*.

For two families $\vec{\sigma} = (\sigma_i)$ of distinct letters and $\vec{t'} = (t'_i)$ of trees with the same indices, we denote by $t[\vec{\sigma} \leftarrow \vec{t'}]$ or $t[\sigma_i \leftarrow t'_i \ \forall i]$ the simultaneous substitution of all nodes of t with label σ_i of t by t'_i (recall that this replaces the entire subtrees rooted at these nodes). In the case of a singleton family we write $t[\sigma \leftarrow t']$.

For $\sigma \in \Sigma$ of rank k and $t_1, \dots, t_k \in \mathsf{T}_\Sigma$, we write $\sigma(t_1, \dots, t_k)$ for the unique tree whose root has label σ and whose subtree rooted at the node i is t_i for all $i \in [k]$. Every tree can be uniquely decomposed this way; since $|t_i| < |\sigma(t_1, \dots, t_k)|$ this allows reasoning by *structural induction*.

2.1 A running example

Let $\Sigma^{\mathsf{exa}} = \{a, b\}$ with $\mathrm{rank}(a) = 2$ and $\mathrm{rank}(b) = 0$. We define $\mathrm{pbt}\colon \mathbb{N} \to \mathsf{T}_{\Sigma^{\mathsf{exa}}}$ (for "perfect binary tree") by:

$$\mathrm{pbt}(0) = b \qquad \mathrm{pbt}(n+1) = a(\mathrm{pbt}(n), \mathrm{pbt}(n)) \text{ for } n \in \mathbb{N}$$

and our running example $\mathsf{exa}\colon \mathsf{T}_{\Sigma^{\mathsf{exa}}} \to \mathsf{T}_{\Sigma^{\mathsf{exa}}}$ by

$$\mathsf{exa}(t) = \mathrm{pbt}(|\,\mathrm{Dom}(t) \cap 1^*2^*|)$$

In other words, $\mathrm{height}(\mathsf{exa}(t))$ counts the number of nodes of t that are reached from the root by paths that cannot take a "turn" from going right to going left. This function has:

exponential size increase: on inputs $a(b, \dots a(b, b) \cdots)$;
linear size-to-height increase: $\mathrm{height}(\mathsf{exa}(t)) \leqslant |t|$;
quadratic height increase: the number of words in 1^*2^* of length at most n is $(n+1)(n+2)/2$.

2.2 Classical tree automata

A *deterministic bottom-up tree automaton* over Σ consists of:

- a finite set Q of states;
- for each $\sigma \in \Sigma$, a transition function $\delta[\sigma]\colon Q^{\mathrm{rank}(\sigma)} \to Q$.

We may identify $\delta[\sigma]$ with an element of Q when $\mathrm{rank}(\sigma) = 0$. As usual, this can be extended to $\delta[t] \in Q$ for every tree $t \in \mathsf{T}_\Sigma$ by structural induction: $\delta[\sigma(t_1, \dots, t_k)] = \delta[\sigma](\delta[t_1], \dots, \delta[t_k])$.

We shall be concerned with two extensions of deterministic bottom-up tree automata with additional data:

- A *bottom-up recognizer* comes with a subset $F \subset Q$ of *accepting states*. It accepts the language $\{t \mid \delta[t] \in F\}$. The languages that can be defined this way form the well-known class of *regular tree languages*.
- A *bottom-up relabeler* comes with a rank-preserving map $\rho\colon \Sigma \times Q \to \Gamma$. It defines a shape-preserving tree-to-tree function, called a *bottom-up relabeling*: for $k = \mathrm{rank}(\sigma)$,

$$[\![\mathcal{R}]\!]\colon \mathsf{T}_\Sigma \to \mathsf{T}_\Gamma \qquad \text{(where } \mathcal{R} \text{ is the relabeler)}$$
$$\sigma(t_1, \dots, t_k) \mapsto \rho(\sigma, \delta[\sigma(t_1, \dots, t_k)])([\![\mathcal{R}]\!](t_1), \dots, [\![\mathcal{R}]\!](t_k))$$

2.3 Tree-generating machines

We recall from [46, §3] a lightweight formalism for describing various tree transducer models. A *tree-generating machine* with output alphabet Γ is made of:

- a set Conf of *configurations*;
- an *initial configuration* $C_{init} \in \mathrm{Conf}$;
- a (partial) *computation-step function* $\mathrm{Conf} \rightharpoonup \mathsf{T}_\Gamma(\mathrm{Conf})$.

$\mathsf{T}_\Gamma(\mathrm{Conf})$ should be seen as a set of global states of a computation:

- the nodes with labels in Γ are part of the eventual output tree, which is being produced in a top-down fashion;
- the leaves with labels in Conf correspond to processes running in parallel, each in charge of producing one of the remaining subtrees of the output.

By taking rewrite rules that substitute a leaf by the image of its label by the computation-step function, we get a confluent rewriting system on $\mathsf{T}_\Gamma(\mathrm{Conf})$. A *run* is a rewriting sequence starting from C_{init}. By confluence, there is at most one tree in T_Γ that can be reached by the runs (indeed, such trees are normal forms). If it exists, we call it the *output* of the machine.

To define the *semantics* $[\![\mathcal{T}]\!] \colon \mathsf{T}_\Sigma \rightharpoonup \mathsf{T}_\Gamma$ *of a tree transducer* $\mathcal{T}$, our methodology is to specify how to build a tree-generating machine from $\mathcal{T}$ and a given input tree $t \in \mathsf{T}_\Sigma$, depending on the transducer model. The output of this tree-generating machine is then used as the definition of $[\![\mathcal{T}]\!](t)$.

2.4 A branchwise semantics

It is often useful to reason on the trajectory of a single process producing one of the output branches, rather than the entire family of processes producing the output tree in parallel. This is why we propose an alternative presentation of the semantics of a tree-generating machine, which does not appear in [46].

Let Γ be the ranked output alphabet. We define the set of *branch words* of a tree over Γ by structural induction:

$$\mathrm{brawo}(\gamma) = \{\gamma\} \quad \text{when } \mathrm{rank}(\gamma) = 0$$
$$\mathrm{brawo}(\gamma(t_1, \dots, t_k)) = \bigcup_i (\gamma, i) \cdot \mathrm{brawo}(t_i) \text{ when } k \geqslant 1$$

Claim 2.1. For $t \in \mathsf{T}_\Gamma$, $b = (\gamma_1, i_1) \dots (\gamma_n, i_n)\gamma_{n+1} \in \mathrm{brawo}(t)$:

- $\mathrm{lab}_t(i_1 \dots i_\ell) = \gamma_{\ell+1}$ for all $\ell \in \{0, \dots, n\}$;
- mapping b to $\{\varepsilon, i_1, \dots, (i_1 \dots i_n)\}$ defines a size-preserving bijection from $\mathrm{brawo}(t)$ to the total branches of t.

Consequently, $\mathrm{brawo} \colon \mathsf{T}_\Gamma \to \mathcal{P}(\widetilde{\Gamma}^* \Gamma^{(0)})$ is injective, where

$$\widetilde{\Gamma} = \{(\gamma, i) \mid \gamma \in \Gamma,\ i \in \{1, \dots, \mathrm{rank}(\gamma)\}\}$$

We therefore aim to characterize the output of a tree-generating machine (with the notations of the previous subsection) through its branch words. To do so, we consider a labeled transition system over $\mathrm{Conf} \cup \Gamma^{(0)}$ with labels in $\widetilde{\Gamma}^*$.

Definition 2.2. Let $C \in \mathrm{Conf}$, $b \in \widetilde{\Gamma}^*$ and $x \in \mathrm{Conf} \cup \Gamma^{(0)}$.

$$C \xrightarrow{b} x \overset{\text{def}}{\iff} bx \in \mathrm{brawo}(\text{computation-step}(C))$$

As expected, we define $x \xrightarrow{b}{}^* y$ by the existence of some decomposition $b = b_1 \dots b_n$ that labels some *path*

$$x = x_0 \xrightarrow{b_1} \dots \xrightarrow{b_n} x_n = y$$

This is an instance of labeled transition system *with monoidal structure on labels*, see for instance [50, Example 2.3].

Claim 2.3. If the output of the tree-generating machine exists, then its set of branch words is $\{b\gamma \mid C_{init} \xrightarrow{b}{}^* \gamma\}$.

In general, we refer to a path starting from C_{init} (not necessarily ending on an element of $\Gamma^{(0)}$) as a *branch-outputting run*.

3 Tree-to-tree Hennie machines

3.1 Unrestricted THMs: definition & semantics

Definition 3.1. An *unrestricted tree-to-tree Hennie machine* (uTHM) is a tuple $\mathcal{H} = (Q, M, \top, \Sigma, \Gamma, q_{init}, \delta)$ where:

- Q is the finite set of *states* of $\mathcal{H}$;
- M is the finite set of *memory symbols* of $\mathcal{H}$;
- $\top \in M$ is the initial memory value of $\mathcal{H}$;
- Σ (resp. Γ) is the ranked *input* (resp. *output*) alphabet;
- $q_{init} \in Q$ is the initial state of $\mathcal{H}$;
- $\delta \colon Q \times \Sigma \times M \rightharpoonup \mathsf{T}_\Gamma(Q \times M \times \mathsf{D}_{\mathsf{R}_\Sigma})$ is the (partial) *transition function* of $\mathcal{H}$, such that for all $\sigma \in \Sigma$,

$$\delta(Q \times \{\sigma\} \times M) \subseteq \mathsf{T}_\Gamma(Q \times M \times \mathsf{D}_{\mathrm{rank}(\sigma)})$$

The semantics of this uTHM has been informally sketched in the introduction. To describe it rigorously, we use the formalism of tree-generating machines. We start with its *configurations* over an input $t \in \mathsf{T}_\Sigma$: they are tuples (u, q, μ) where

- u is a node of t, the current position;
- $q \in Q$ is the current state;
- $\mu \colon \mathrm{Dom}(t) \to M$ describes the current memory values.

For such a configuration and $(q', m, d) \in Q \times M \times \mathsf{D}_{\mathsf{R}_\Sigma}$, let

$$(u, q, \mu) \triangleright (q', m, d) = (ud, q', \mu') \text{ where } \begin{cases} \mu'(u) = m \\ \mu'(v) = \mu(v) \text{ for } v \neq u \end{cases}$$

— the machine "writes the symbol m in memory at the current position u before moving to the new position ud". The result of this operation is *undefined* if $ud \notin \mathrm{Dom}(t)$.

We extend the definition of $\triangleright$ to $\mathsf{T}_\Gamma(Q \times M \times \mathsf{D}_{\mathsf{R}_\Sigma})$ by application to the leaves; inductively: $C \triangleright \gamma(t_1, \dots, t_k) = \gamma(C \triangleright t_1,\ \dots,\ C \triangleright t_k)$.

Let $\mathrm{Conf}(\mathcal{H}, t)$ be the set of configurations of the uTHM $\mathcal{H}$ on the input tree t. The *computation-step function* is

$$\mathrm{Conf}(\mathcal{H}, t) \rightharpoonup \mathsf{T}_\Gamma(\mathrm{Conf}(\mathcal{H}, t))$$
$$(u, q, \mu) \mapsto (u, q, \mu) \triangleright \delta(q, \mathrm{lab}_t(u), \mu(u))$$

Finally, the *initial configuration* is $C_{init}(t) = (\varepsilon, q_{init}, (v \mapsto \top))$.

Example 3.2. We give a uTHM that computes exa from §2.1.

- There are 2 states, q_0 (initial) and q_1.
- There are 4 memory symbols $\top, \mathsf{L}, \mathsf{R}, \maltese$.
- The transitions are (undefined in the 2 unspecified cases):

$$(q_0, a, \top) \mapsto a((q_1, \mathsf{L}, 2), (q_1, \mathsf{L}, 2)) \qquad (q_0, b, \top) \mapsto a(b, b)$$
$$(q_1, a, \top) \mapsto a((q_1, \mathsf{R}, 2), (q_1, \mathsf{R}, 2)) \qquad (q_1, a, \mathsf{L}) \mapsto (q_0, \maltese, 1)$$
$$(q_1, b, \top) \mapsto a((q_1, \maltese, \uparrow), (q_1, \maltese, \uparrow)) \qquad (q_1, a, \mathsf{R}) \mapsto (q_1, \maltese, \uparrow)$$

On an input tree $t \in \mathsf{T}_{\Sigma^{\mathrm{exa}}}$, all branch-outputting runs ending in b traverse the same configurations in the same order; the only difference is in the labels ($(a, 1)$ vs. $(a, 2)$). Such a run starts by writing L to the root, and going down to the right child in state q_1. It then goes right, writing R in memory on all encountered nodes, until it reaches a leaf. It then starts going up while writing $\maltese$ on the visited nodes (including the leaf) until it reaches a node where the memory is L. After having thus visited all nodes in $\mathrm{Dom}(t) \cap 2^*$, it then moves to the left child of the root, which has not yet been visited, and switches back to the initial state q_0. The computation then proceeds as it did before on the subtree rooted at the left child.

Note that the transitions only output a node a when the current memory is $\top$, which ensures that each input node is counted at most once when determining the output height.

3.2 The bounded-visit restriction

Remark 3.3. A uTHM computing a string-to-boolean function (i.e. $\max(\mathrm{rank}(\Sigma)) = 1$ and $\Gamma = \{\texttt{true} : 0,\ \texttt{false} : 0\}$) is the same thing as a deterministic *linear bounded automaton*. It is well known that this automaton model can recognize at least all context-free languages [40, Theorem 3] while its nondeterministic variant recognizes precisely the context-sensitive languages [40, Theorem 1].

In order to stay within the realm of "finite-state computability" we shall therefore impose the bounded-visit restriction. The latter can be conveniently defined by using the branchwise semantics.

Definition 3.4. Let $\mathcal{H}$ be a uTHM with input alphabet Σ.

The *number of visits* of a branch-outputting run of $\mathcal{H}$ at a node u (resp. a set of nodes S) of the input tree is the number of configurations in the path whose current position is u (resp. is in S).

$\mathcal{H}$ is *bounded-visit* on an input language $L \subseteq \mathsf{T}_\Sigma$ whenever there is a uniform bound N such that, for any $t \in L$, any node u of t and any branch-outputting run on the input t, the number of visits at u is at most N. A *tree-to-tree Hennie machine* (THM) is a uTHM that is bounded-visit on all inputs ($L = \mathsf{T}_\Sigma$).

Example 3.2 is a THM: it visits each node at most twice.

We can now establish linear size-to-height increase rigorously.

Proof of Claim 1.3. By definition of height & Claims 2.1 / 2.3,

$$\begin{aligned}\mathrm{height}(\llbracket\mathcal{H}\rrbracket(t)) &= \max\{|b| \mid \exists\gamma : b\gamma \in \mathrm{brawo}(\llbracket\mathcal{H}\rrbracket(t))\}\\ &= \max\{|b_1 \dots b_\ell| \mid \exists\gamma : C_{init}(t) \xrightarrow{b_1} \dots \xrightarrow{b_\ell} \gamma\}\end{aligned}$$

The labeled transition system $\to$ for $\mathcal{H}$ has labels derived from the branch words of the range $\delta(Q \times \Sigma \times M)$ of the transition function of $\mathcal{H}$ (to see this, unfold the definition of the computation-step function). Since this range is finite, $|b_i| = O(1)$ in the above description, so $|b_1 \dots b_\ell| = O(\ell)$. Finally, $\ell = O(|t|)$ because any branch-outputting run of a THM visits each of the $|t|$ nodes $O(1)$ times — this is the bounded-visit restriction. □

Remark 3.5. Unfortunately, it is undecidable whether a uTHM is a THM, already in the string-to-boolean case [32, Theorem 1 + Lemma 1]. However, given an explicit bound, we have a decidability result (Corollary 5.5).

We also define the *number of downwards visits* of a branch-outputting run of a uTHM at an input node u by counting the occurrences of transitions of the form $(u\uparrow, \dots) \to (u, \dots)$ (thus, if u is the root, this number is zero). The following property guarantees that "bounded-downwards-visit" is the same thing as bounded-visit.

Claim 3.6. The number of visits at a node is at most the sum of the numbers of downwards visits at this node and at its children.

Proof. Along a branch-outputting run, the position of a uTHM moves locally, therefore every upwards visit of u from ui must have been preceded by a downwards visit of ui from u. □

3.3 Weight-reducing THMs & totality

The bounded-visit restriction is defined semantically. We present here a syntactic counterpart, taken from [32] for Hennie machine as acceptors of string languages.

Definition 3.7. A uTHM with the set M of memory symbols and the transition function δ is *weight-reducing* if there exists a partial order on M such that

- the initial symbol $\top$ is maximal;
- for every $m \in M$, only memory symbols that are strictly smaller than m appear in $\delta(\dots, \dots, m)$.

For instance, Example 3.2 is weight-reducing ($\top > \mathsf{L}, \mathsf{R} > ✠$). The following result may be compared with [32, Lemma 2].

Proposition 3.8. *Every weight-reducing* uTHM *is a* THM.
Conversely, given a uTHM $\mathcal{H}$ *and* $N \in \mathbb{N}$*, one can compute a weight-reducing* THM $\mathcal{H}'$ *such that* $\llbracket\mathcal{H}'\rrbracket$ *is a* total function *and:*

- *for t in the domain of $\llbracket\mathcal{H}\rrbracket$ s.t. all branch-outputting runs of $\mathcal{H}$ visit at most N times each node, $\llbracket\mathcal{H}'\rrbracket(t) = \llbracket\mathcal{H}\rrbracket(t)$;*
- *on other inputs, $\mathcal{H}'$ returns an output that contains some chosen default leaf symbol.*

In particular, if $\mathcal{H}$ is a THM *whose number of visits is uniformly bounded by N, then $\llbracket\mathcal{H}\rrbracket \subseteq \llbracket\mathcal{H}'\rrbracket$.*

Speaking of totality, the semantics of a THM is naturally a *partial* function, but we focus on the case when the function happens to be total. This restriction is not very significant: as we explain below (Corollary 5.5), the domain of a partial THM is effectively regular.

3.4 The narrow-visit restriction

Recall that a *chain* in a partially ordered set (poset) is a totally ordered subset, while an *antichain* is a subset of pairwise incomparable elements. When we say that a set of nodes in a tree is an antichain, we mean with respect to the ancestor ordering.

Definition 3.9. A uTHM $\mathcal{H}$ is *narrow-visit* on an input language L whenever there is a uniform bound N such that, for any $t \in L$, *any antichain S of t* and any branch-outputting run on the input t, the number of visits at S is at most N.

We rephrase the narrow-visit property by combining a basic observation with a classical duality theorem (which is closely related to other dualities in combinatorial optimization, cf. [25]).

Theorem 3.10 (Dilworth [14]). *A poset is covered by the union of $N < \infty$ chains iff all its antichains have cardinality at most N.*

Claim 3.11. For a set of tree nodes, the following are equivalent:

- it is a chain (for the ancestor ordering);
- it is included in some branch of the tree.

Corollary 3.12. *An* uTHM $\mathcal{H}$ *is narrow-visit on an input language L if and only if the two following conditions hold jointly:*

- *$\mathcal{H}$ is bounded-visit on L;*
- *for every branch-outputting run whose input tree is in L, the set of visited nodes (i.e. those with a nonzero number of visits) is included in the union of $O(1)$ input branches.*

Corollary 3.13. *If a* uTHM *is narrow-visit on all inputs, then it is a* THM *and it has linear height increase.*

Proof. As we have seen when proving Claim 1.3, the output height is at most linear in the length of branch-outputting runs. On an input t, such a run visits $O(1)$ branches, each of which has at most height(t) + 1 nodes, and each node is visited $O(1)$ times. □

3.5 Basic extensions

To conclude Section 3, we present three additional features that can be added to uTHMs and THMs. We show that these extensions are conservative, i.e. do not lead to an increase in expressive power.

Stay-instructions. The positions of two successive configurations of an uTHM are always distinct neighbors. We extend the syntax of transitions to allow staying on the current position; this is often convenient in our constructions of THMs. Stay-instructions are indicated by a new symbol $\circlearrowleft$, and relax the requirement on the transition function δ to be

$$\delta(Q \times \{\sigma\} \times M) \subseteq \mathsf{T}_\Gamma(Q \times M \times (\{\circlearrowleft\} \cup \mathsf{D}_{\mathrm{rank}(\sigma)}))$$

The semantics is extended by taking $u\circlearrowleft = u$ for any input node u.

Lemma 3.14. *Every* uTHM *with stay-instructions can be effectively translated to an equivalent* uTHM*. This translation preserves the bounded-visit and narrow-visit restrictions on any input language.*

Bottom-up initialization. Our second conservativity result can be understood as a "closure under precomposition" property.

Lemma 3.15. *Given a* uTHM $\mathcal{H}$ *and a bottom-up relabeler* $\mathcal{R}$ *(cf. §2), whose respective input and output alphabets match, one can effectively construct a* uTHM $\mathcal{H}'$ *such that* $[\![\mathcal{H}']\!] = [\![\mathcal{H}]\!] \circ [\![\mathcal{R}]\!]$*.*
If $\mathcal{H}$ *is bounded-visit on* $[\![\mathcal{R}]\!](L)$*, then* $\mathcal{H}'$ *is bounded-visit on* L*.*

The idea of the proof is to perform a post-order tree traversal that computes and records in memory, at each node, the state reached there by the deterministic bottom-up automaton contained in $\mathcal{R}$.

Definition 3.16. A uTHM *with bottom-up initialization* is:

- the data of a uTHM — we reuse the notations of Def. 3.1;
- a deterministic bottom-up tree automaton (Q', δ');
- a map $\mathtt{init}\colon Q' \to M$.

Its semantics is the same as for uTHM except that for an input tree $t \in \mathsf{T}_\Sigma$, the initial memory value at node u is $\mathtt{init}(\delta'[t|_u])$.

We then define in the same way as Definition 3.4 the bounded-visit condition. A THM *with bottom-up initialization* is a uTHM with bottom-up initialization that happens to be bounded-visit.

Proposition 3.17. *Every* uTHM *with bottom-up initialization can be effectively translated to an equivalent* uTHM*.*
The translation preserves the bounded-visit condition on any input language, but not the narrow-visit condition.

Proof. For any uTHM $\mathcal{H}''$ with bottom-up initialization, let:

- $\mathcal{R}$ be the bottom-up relabeler using (Q', δ') from the above definition and $\rho\colon (\sigma, q') \mapsto (\sigma, \mathtt{init}(q'))$;
- $\mathcal{H}$ be a uTHM simulating $\mathcal{H}''$ while reading from the input letters the initial memory values when it encounters a node for the first time — thus, when $\mathcal{H}''$ is bounded-visit on all inputs, $\mathcal{H}$ is bounded-visit on the range of $[\![\mathcal{R}]\!]$.

Then $[\![\mathcal{H}'']\!] = [\![\mathcal{H}]\!] \circ [\![\mathcal{R}]\!]$. Therefore, we can apply Lemma 3.15. □

Regular lookaround. Finally, we consider the extension of tree-to-tree Hennie machines with the ability to evaluate regular predicates on the current configuration to decide the next transition.

Definition 3.18 (Encodings of configurations). Consider a uTHM $\mathcal{H}$ with the notations of Definition 3.1.
For $t \in \mathsf{T}_\Sigma$, we encode $(u, q, \mu) \in \mathrm{Conf}(\mathcal{H}, t)$ as the tree over $\Sigma \times M \times (Q \cup \{\mathtt{notHere}\})$, with the ranks inherited from Σ, whose domain is $\mathrm{Dom}(t)$ and whose label at each node v is

$$(\mathrm{lab}_t(v),\ \mu(v),\ [q \text{ if } v = u, \text{otherwise } \mathtt{notHere}])$$

Definition 3.19. A uTHM *with regular lookaround* (uTHM$^{\mathsf{R}}$) $\mathcal{H}$ from an input alphabet Σ to an output alphabet Γ consists of:

- the data of a uTHM *without the transition function* as in Definition 3.1, i.e. $(Q, M, \top, \Sigma, \Gamma, q_{init})$;
- a deterministic bottom-up automaton $(Q_{\mathrm{LA}}, \delta_{\mathrm{LA}})$ over the input alphabet $\Sigma \times M \times (Q \cup \{\mathtt{notHere}\})$, which we call the *lookaround automaton*;
- a transition function $\delta\colon Q_{\mathrm{LA}} \to T_\Gamma(Q \times M \times \mathsf{D}_k)$.

The configurations of the machine $\mathcal{H}$ are the same as if it were an ordinary uTHM (the transition function is not involved). The semantics is defined analogously to ordinary uTHMs, except that the computation-step function uses the lookaround automaton:

$$\mathrm{Conf}(\mathcal{H}, t) \rightharpoonup \mathsf{T}_\Gamma(\mathrm{Conf}(\mathcal{H}, t)) \qquad (\text{for } t \in \mathsf{T}_\Sigma)$$
$$C \mapsto C \circledcirc \delta_{\mathrm{LA}}[\text{encoding of } C]$$

We combine bottom-up initialization with dynamic maintenance of local lookaround information to prove:

Theorem 3.20. *Every* uTHM$^{\mathsf{R}}$ *can be effectively translated to an equivalent* uTHM*. The translation preserves the bounded-visit condition on any input language.*

Corollary 3.21. *The classes of functions computed by* uTHM*s and by* THM*s are both closed by precomposition by* MSO relabelings*.*

The aforementioned MSO relabelings are tree transformations that keep the shape of a tree (i.e. its domain) while changing its labels in a regular fashion; see for instance [11, Definition 7.20] for a book reference.

Remark 3.22. For automata models such as e.g. tree-walking tree transducers and macro tree transducers, regular lookaround morally amounts to preprocessing by MSO relabelings. But for THMs, this is not the case because the information given by the regular lookaround depends not only on the input tree and current position, but also on the current memory values at each node.

4 Macro tree transducers

Notations specific to this section. We reserve two special sets of symbols, a set $X = \{x_i : i \in \mathbb{N} \setminus \{0\}\}$ of *variables* and another set $Y = \{y_i : i \in \mathbb{N} \setminus \{0\}\}$ of *parameters*. We also write $X_k = \{x_i : i \leqslant k\}$ and $Y_k = \{y_i : i \leqslant k\}$ for $k \in \mathbb{N}$.
A tree in $\mathsf{T}_\Sigma(Y)$, whose leaves may be parameters, is called a *tree context* over Σ. A *k-ary tree context* is a tree in $\mathsf{T}_\Sigma(Y_k)$.
For Q a ranked set and Z an arbitrary set, $\langle Q, Z\rangle$ is the ranked set of pairs $\langle q, z\rangle$ for $q \in Q$, $z \in Z$, with $\mathrm{rank}(\langle q, z\rangle) = \mathrm{rank}(q)$.

4.1 MTTs: definition and semantics

Definition 4.1. A (total deterministic) MTT $\mathcal{M}$ from an input alphabet Σ to an output alphabet Γ (both ranked) consists of:

- a finite *ranked* set Q of states & an initial state $q_{init} \in Q^{((0))}$;
- for each state $q \in Q$ and each input letter $\sigma \in \Sigma$, a context

$$\mathsf{RHS}_{\mathcal{M}}(q,\sigma) \in \mathsf{T}_{\Gamma \cup \langle Q, X_{\mathrm{rank}(\sigma)}\rangle}(Y_{\mathrm{rank}(q)})$$

The latter is meant to be the right-hand side of a rewrite rule

$$\langle q, \sigma(x_1,\dots,x_{\mathrm{rank}(\sigma)})\rangle(y_1,\dots,y_{\mathrm{rank}(q)}) \to \mathsf{RHS}_{\mathcal{M}}(q,\sigma)$$

Let us illustrate what this means before giving a formal definition:

Example 4.2. Here is an MTT for the function exa from §2.1.

$$\begin{aligned}
\langle q_0, a(x_1,x_2)\rangle &\to \langle q_1,x_1\rangle(a(\langle q_0,x_2\rangle,\langle q_0,x_2\rangle))\\
\langle q_0, b\rangle &\to a(b,b) \quad \text{(i.e. pbt(1) as defined in §2.1)}\\
\langle q_1, a(x_1,x_2)\rangle(y_1) &\to \langle q_1,x_1\rangle(a(y_1,y_1))\\
\langle q_1, b\rangle(y_1) &\to a(y_1,y_1)
\end{aligned}$$

$$\begin{aligned}
\text{We have } \langle q_0, a(b,b)\rangle &\to \langle q_1,b\rangle(a(\langle q_0,b\rangle,\langle q_0,b\rangle))\\
&\to \langle q_1,b\rangle(a(\mathrm{pbt}(1),\langle q_0,b\rangle))\\
&\to \langle q_1,b\rangle(\mathrm{pbt}(2)) \to \mathrm{pbt}(3) = \mathsf{exa}(a(b,b)).
\end{aligned}$$

In the literature, the semantics of MTTs allow the rewrite rules to apply to any subtree rooted at some node with label of the form $\langle q,t\rangle$. Let us call such trees *redexes*:

$$\mathrm{Redex}(\mathcal{M}) = \{t \in \mathsf{T}_{\Gamma\cup\langle Q,\mathsf{T}_\Sigma\rangle} \mid \mathrm{lab}_t(\varepsilon) \in \langle Q,\mathsf{T}_\Sigma\rangle\}$$

There is also a standard notion [21, Definition 3.4] of *outside-in derivation*[12] that only rewrites *outermost redexes*: those that are not contained in some other redex, i.e. whose root does not have a $\langle Q,\mathsf{T}_\Sigma\rangle$-labeled strict ancestor. The example above is not outside-in: the only steps rewriting outmost redexes are the first and the last.

We now capture outside-in derivations — which suffice to interpret total deterministic MTTs as total functions [21, §4.1] — in the framework of tree-generating machines. This serves as our reference definition of the semantics of MTTs in this paper, avoiding the need to manipulate "second-order substitutions" in full generality.

The tree-generating machine for the MTT $\mathcal{M}$ on an input $t \in \mathsf{T}_\Sigma$ has the set of configurations $\mathrm{Redex}(\mathcal{M})$ and the initial configuration $\langle q_{init}, t\rangle$. Its computation-step function is

$$\mathrm{Redex}(\mathcal{M}) \to \mathsf{T}_{\Gamma\cup\langle Q,\mathsf{T}_\Sigma\rangle} \cong \mathsf{T}_\Gamma(\mathrm{Redex}(\mathcal{M}))$$

$$\langle q,\sigma(\vec{t})\rangle(\vec{t}') \mapsto \begin{cases}\text{replace every label } \langle q',x_i\rangle \text{ by } \langle q',t_i\rangle\\ \text{in } \mathsf{RHS}_{\mathcal{M}}(q,\sigma)[\vec{y} \leftarrow \vec{t}']\end{cases}$$

using the fact that:

Claim 4.3. The simultaneous substitution

$$[r \text{ seen as a leaf} \leftarrow r \text{ seen as a tree} \quad \forall r \in \mathrm{Redex}(\mathcal{M})]$$

defines a bijection $\mathsf{T}_\Gamma(\mathrm{Redex}(\mathcal{M})) \xrightarrow{\sim} \mathsf{T}_{\Gamma\cup\langle Q,\mathsf{T}_\Sigma\rangle}$ whose inverse turns outermost redexes into leaves.

[12] Or "OI derivation", corresponding to call-by-name evaluation in the λ-calculus, while inside-out (IO) derivations correspond to call-by-value.

4.2 Syntactic criteria for linear (size-to-)height

We now recall the results of Gallot, Maneth, Nakano and Peyrat [28] on L(S)HI-MTTs. Originally, they are stated in terms of MTTs with *regular lookahead*. As usual, regular lookahead amounts to preprocessing by a bottom-up relabeling; we propose a rephrasing that takes this point of view.

Definition 4.4. Let $\mathcal{M}$ be an MTT with the notations above. Let us fix a rank 0 symbol $\square$. We define the MTT $\mathcal{M}^{\blacksquare}$ with:

- input alphabet $\Sigma \cup \{\square\}$ and output alphabet $\Gamma \cup \langle Q, \{\blacksquare\}\rangle$;
- the same ranked set of states Q as $\mathcal{M}$;
- the rules of $\mathcal{M}$, plus $\langle q,\square\rangle(\vec{y}) \to \langle q,\blacksquare\rangle(\vec{y})$ for $q \in Q$.

For instance, for the MTT $\mathcal{M}_{\mathsf{exa}}$ of Example 4.2,

$$[\![\mathcal{M}^{\blacksquare}_{\mathsf{exa}}]\!](a(\square,b)) = \langle q_1,\blacksquare\rangle(\mathrm{pbt}(2))$$

$\mathcal{M}^{\blacksquare}$ is called the "extension" of $\mathcal{M}$ in [28]. Its inputs are trees in $\mathsf{T}_\Sigma(\{\square\})$; they can be seen as inputs to $\mathcal{M}$ in T_Σ with some "missing subtrees" marked by $\square$. Informally, the output of $\mathcal{M}^{\blacksquare}$ represents what happens if we run $\mathcal{M}$ on such an incomplete input, computing as much as possible while recording the places where $\mathcal{M}$ gets stuck because it wants to explore a missing subtree.

For an input language $L \subseteq \mathsf{T}_\Sigma$, we denote by $L^{\square} \subseteq \mathsf{T}_\Sigma(\{\square\})$ the set of "inputs from L with missing subtrees", that is, the smallest superset of L closed under replacing a subtree by $\square$. The set of "inputs from L with a single missing subtree" is

$$L^{1\square} = \{t \in L^{\square} \mid t \text{ has only one } \square\text{-labeled leaf}\}$$

Definition 4.5. For $t' \in \mathsf{T}_{\Gamma\cup\langle Q,\{\blacksquare\}\rangle}$, let $\mathrm{height}_{\blacksquare}(t')$ be the maximum number of $\langle Q,\{\blacksquare\}\rangle$-labeled nodes on a branch of t'.

An MTT $\mathcal{M}$ is *finite nesting* (resp. *finite yield nesting*) on an input language L when $\mathrm{height}_{\blacksquare} \circ [\![\mathcal{M}^{\blacksquare}]\!]$ is bounded on $L^{1\square}$ (resp. $L^{\square}$).

One can show that $\mathrm{height}_{\blacksquare}(\mathcal{M}^{\blacksquare}_{\mathsf{exa}}(t'))$ is the number of $\square$-labeled nodes in $\mathrm{Dom}(t') \cap 1^*2^*$ for all $t' \in \mathsf{T}_{\Sigma^{\mathsf{exa}}}(\{\square\})$. Therefore, $\mathcal{M}_{\mathsf{exa}}$ is finite nesting, but not finite yield nesting — which is consistent with the (size-to-)height increase properties stated in §2.1, since:

Theorem 4.6 (rephrasing of [28, §4]). *Given an* MTT $\mathcal{M}$*, one can compute an* MTT $\mathcal{M}'$ *and a bottom-up relabeler* $\mathcal{R}$ *such that:*

- $[\![\mathcal{M}]\!] = [\![\mathcal{M}']\!] \circ [\![\mathcal{R}]\!]$;
- $\mathcal{M}$ *is LSHI iff* $\mathcal{M}'$ *is finite nesting on the range of* $\mathcal{R}$*;*
- $\mathcal{M}$ *is LHI iff* $\mathcal{M}'$ *is finite yield nesting on the range of* $\mathcal{R}$*.*

4.3 From MTTs to (u)THMs

The main result of this section is:

Theorem 4.7. *Any (total deterministic)* MTT $\mathcal{M}$ *can be effectively translated to a* uTHM $\mathcal{H}$ *such that:*

- $[\![\mathcal{M}]\!] = [\![\mathcal{H}]\!]$;
- *for any input tree* t*, any antichain* S *of nodes of* t*, and any branch-outputting run of* $\mathcal{H}$*, the number of downwards visits at* S *is at most* $\mathrm{height}_{\blacksquare} \circ [\![\mathcal{M}^{\blacksquare}]\!](t[s \leftarrow \square \ \ \forall s \in S])$.

The second item implies that, for any input language L*,*

$$\mathcal{M} \text{ finite nesting on } L \implies \mathcal{H} \text{ bounded-visit on } L$$
$$\mathcal{M} \text{ finite yield nesting on } L \implies \mathcal{H} \text{ narrow-visit on } L$$

To give some intuition for the proof, one may see the rules of an MTT as mutually recursive definitions of $[\![q]\!] : \mathsf{T}_\Sigma \times \mathsf{T}_\Gamma^{\mathrm{rank}(q)} \to \mathsf{T}_\Gamma$ for each state q. This recursion can be compiled to a call stack. In fact, MTTs are expressively equivalent to pushdown tree transducers [22, Theorem 5.16],[13] though we do not use this result and prefer to give a direct construction from MTTs in order to control the number of visits. To represent the stack in memory, our uTHM records at each node the currently active recursive call, if any, whose argument in T_Σ is the subtree rooted at that node. There can be at most one such call per node, so we can use a finite set of memory symbols. As for $\mathrm{height}_{\blacksquare} \circ [\![\mathcal{M}^{\blacksquare}]\!](t[s \leftarrow \square \ \forall s \in S])$, it overapproximates the total number of calls, in a branch-outputting run, whose arguments in T_Σ are subtrees rooted at nodes in S.

From Theorems 4.6 and 4.7, combined with Lemma 3.15 on pre-composing THMs by bottom-up relabelings, we get:

Corollary 4.8. LSHI-MTT $\subset$ THM.

Corollary 4.9. *Given an* LHI-MTT $\mathcal{M}$*, one can compute an* THM $\mathcal{H}$ *and a bottom-up relabeler* $\mathcal{R}$ *such that* $[\![\mathcal{M}]\!] = [\![\mathcal{H}]\!] \circ [\![\mathcal{R}]\!]$ *and* $\mathcal{H}$ *is* narrow-visit on the range of $[\![\mathcal{R}]\!]$.

This last property is involved in the proof of Corollary 8.5.

5 Inclusion in MSO set interpretations

We assume basic knowledge of monadic second-order logic. For a ranked alphabet Σ, the formulas of $\mathrm{MSO}(\Sigma)$ use the atomic predicates $\sigma(x)$, $x \leqslant_{anc} y$ and $\mathrm{child}_i(x, y)$ to express respectively that x has label σ, x is an ancestor or y, and y is the i-th child of x.

Definition 5.1 ([10]). An MSO *set interpretation* (on ranked trees), or MSO-SI for short, consists of:

- a natural number c;
- input and output ranked alphabets Σ and Γ;
- the following $\mathrm{MSO}(\Sigma)$ formulas, where $\vec{X} = X_1, \dots, X_c$ and $\vec{Y} = Y_1, \dots, Y_c$ are sequences of second-order variables:

$$\underset{(\varphi_\gamma(\vec{X}))_{\gamma\in\Gamma}}{\text{label}} \qquad \underset{\varphi_{anc}(\vec{X},\vec{Y})}{\text{ancestor}} \qquad \underset{(\varphi_i(\vec{X},\vec{Y}))_{i\leqslant \mathsf{R}_\Gamma}}{\text{child}}$$

An MSO-SI $\mathcal{I}$ on an input tree t generates a relational structure $\mathcal{I}(t)$ whose universe is the set of assignments

$$\left\{\nu : \vec{X} \to \mathcal{P}(\mathrm{Dom}(t)) \mid \exists \gamma \in \Gamma : (t, \nu) \models \varphi_\gamma(\vec{X})\right\}$$

and whose relations are $\gamma(\nu) \iff (t, \nu) \models \varphi_\gamma(\vec{X})$ and

$$\nu \leqslant_{anc} \nu' \iff (t, \nu, \nu') \models \varphi_{anc}(\vec{X}, \vec{Y})$$

$$\mathrm{child}_i(\nu, \nu') \iff (t, \nu, \nu') \models \varphi_i(\vec{X}, \vec{Y}) \quad \text{for } i \leqslant \mathsf{R}_\Gamma$$

Then $[\![\mathcal{I}]\!] : \mathsf{T}_\Sigma \rightharpoonup \mathsf{T}_\Gamma$ is defined on t, if, and only if, $\mathcal{I}(t)$ is isomorphic to a tree, which we take as the value of $[\![\mathcal{I}]\!](t)$.

Example 5.2. We give an MSO-SI $\mathcal{I}$ defining our running example from §2.1. We assume that all formulas are restricted to nodes in 1^*2^*, which is MSO definable by guarding all quantifications $\exists x$ (resp. $\exists X$) with $\psi_{1^*2^*}(x) = \exists z \ \mathrm{child}_1^*(\varepsilon, z) \wedge \mathrm{child}_2^*(z, x)$ (resp $\forall x \in X \ \psi_{1^*2^*}(x)$) where child_i^* is the transitive closure of the child_i relation and ε is the constant marking the root. Intuitively, $\mathcal{I}$ relies on a (right-depth) ordering $\leqslant$ of the nodes of the input in $\mathrm{Dom}(t) \cap 1^*2^*$. We define the successor in this order by

$$S_\leqslant(x, y) \equiv \mathrm{child}_2(x, y) \vee \big((\forall z \neg \mathrm{child}_2(x, z)) \wedge \exists z \ \mathrm{child}_2^*(z, x) \wedge \forall z' \neg \mathrm{child}_2(z', z) \wedge \mathrm{child}_1(z, y)\big)$$

Then the output nodes of depth i are defined by partitions into X_1, X_2 of the i-smallest elements of $\leqslant$.

Let us consider the formula $\psi_{dom}(X_1, X_2)$ stating that X_1 and X_2 are disjoint and $X_1 \cup X_2$ is a set downward closed by $\leqslant$, and $max_\leqslant(X_1, X_2, x)$ which holds true if x is the $\leqslant$-largest element in $X_1 \cup X_2$. Then we define T by $c = 2$, $\varphi_a(X_1, X_2)$ is ψ_{dom} and $X_1 \cup X_2$ does not cover all $\leqslant$, while $\varphi_b(X_1, X_2)$ is $\psi_{dom}(X_1, X_2)$ and $X_1 \cup X_2$ does cover all $\leqslant$. The ancestor formula is

$$\varphi_{anc}(\vec{X}, \vec{Y}) = \forall z \bigwedge_{i=1,2} z \in X_i \Rightarrow z \in Y_i$$

where $x \leqslant y$ is the transitive closure of $S_\leqslant(x, y)$. And finally $\varphi_i(\vec{X}, \vec{Y})$ is the conjunction of $\varphi_{anc}(\vec{X}, \vec{Y})$ and

$$\exists x, y \ max_\leqslant(\vec{X}, x) \wedge max_\leqslant(\vec{Y}, y) \wedge S_\leqslant(x, y) \wedge y \in Y_i$$

The main result of this section is the following Theorem:

Theorem 5.3. *Any total* THM $\mathcal{H}$ *can be effectively translated to an* MSO-SI $\mathcal{I}$ *such that* $[\![\mathcal{H}]\!] = [\![\mathcal{I}]\!]$.

Sketch of proof. The main ingredients of the proof are the notions of local and (coherent) global profiles. Intuitively, a branch-outputting run of $\mathcal{H}$ is entirely characterized by the finite visits it does to each input nodes. A local profile on an input node is then a sequence of visits. A global profile simply associates to each input node a local profile. It is coherent if the local profiles collectively describe a (partial) branch-outputting run. As a THM moves step by step, the coherence of a global profile can be decided by neighboring properties and hence it is MSO-definable.

To define the MSO-SI $\mathcal{I}$, we set c as the number of possible local profiles. By associating each set X_p to a local profile p, an evaluation (t, ν) where the X_i are a partition of t defines a global profile on $\mathrm{Dom}(t)$. The domain formulas $\varphi_\gamma(\vec{X})$ state that the coloring $\vec{X}$ is a coherent global profile and describes a branch-outputting and a distinguished node γ in its last transition. The ancestor formula $\varphi_{anc}(\vec{X}, \vec{Y})$ states that the global profile defined by $\vec{Y}$ is an extension of the one defined by $\vec{X}$. Finally, the successor formulas $\varphi_i(\vec{X}, \vec{Y})$ are similar to the ancestor formula, additionally requiring that the extension is only one step, i.e. with a unique new visit that produces an element of Γ, the i-th successor of $\vec{X}$. □

Corollary 5.4. Tree-to-string *Hennie machines (i.e.* THMs *whose output alphabet have maximum rank 1) can be effectively translated to equivalent* MSO transductions.

Proof. Since the size of a string is equal to its height (plus one), by Claim 1.3 tree-to-string THMs are linear size increase. MSO-SI of linear size increase can be translated to MSO transductions [27, Theorem 1.5], therefore Theorem 5.3 gives the result. □

Corollary 5.5. *The subset of inputs of a* uTHM *on which it is defined and visits each node at most N times is effectively a regular language.*

[13] Other similar equivalence results include HDT0L systems vs. a variant of pushdown transducers [23] & copyful streaming string transducers vs. marble transducers [15, 16].

Proof. Let $\mathcal{H}$ be a uTHM with $[\![\mathcal{H}]\!] : \mathsf{T}_\Sigma \rightharpoonup \mathsf{T}_\Gamma$. We apply Proposition 3.8, taking a default leaf symbol $\blacksquare \notin \Gamma$. We then get a total THM $\mathcal{H}'$ with output alphabet $\Gamma \cup \{\blacksquare\}$ such that $[\![\mathcal{H}']\!]^{-1}(\mathsf{T}_\Gamma)$ is the aforementioned subset. The *first-order* formula $\forall x.\, \neg\blacksquare(x)$ defines T_Γ within $\mathsf{T}_\Gamma(\{\blacksquare\})$, so its inverse image by any MSO set interpretation is regular [10, §2.3]. (Corollary 8.1 states a stronger preservation property of THMs, but its proof goes through λ-calculus.) □

6 Actors: an interactive model of computation

This section treats part of the equivalence proof between tree-to-tree Hennie machines and a λ-calculus characterization — a part that *that does not involve any λ-term*. We introduce a model of computation for tree-to-tree functions, based on a formalism from from Clairambault and Murawski [8] of tree-generating, finite-state processes which we call *actors*. The main result in this section (Lemma 6.10) is a translation from actor-based tree transducers with a certain boundedness property to Hennie machines.

The idea is that an actor and its environment interact by exchanging messages; during the course of the interaction, both the actor and the environment may also output tree nodes. We describe the allowed messages for an actor using *types*, which form a subset of linear logic formulas (though a more general notion of interface would have been possible): $A, B ::= o \mid A \multimap B \mid A \,\&\, B$.

Definition 6.1 ([8, §3.1]). The sets $\mathrm{IM}(A)$ of incoming messages and $\mathrm{OM}(A)$ of outgoing messages at a type A are defined by:

$$\mathrm{IM}(o) = \{\bullet\} \qquad \mathrm{OM}(o) = \varnothing$$
$$\mathrm{IM}(A \multimap B) = \mathrm{OM}(A) + \mathrm{IM}(B) \qquad \mathrm{OM}(A \multimap B) = \mathrm{IM}(A) + \mathrm{OM}(B)$$
$$\mathrm{IM}(A \,\&\, B) = \mathrm{IM}(A) + \mathrm{IM}(B) \qquad \mathrm{OM}(A \,\&\, B) = \mathrm{OM}(A) + \mathrm{OM}(B)$$

using the disjoint sum operation $X + Y = \{\mathsf{L}\} \times X \cup \{\mathsf{R}\} \times Y$.

Intuitively, an actor is in a *passive* state when it is waiting for an incoming message from the environment. Upon receiving this message, it becomes *active*, then performs some computation, and eventually sends an outgoing message, returning the control to the environment by becoming again passive.

Definition 6.2 ([8, Def. 2]). An actor α of type A over the ranked alphabet Γ (notation: $\Gamma \Vdash \alpha : A$) consists of:

- a finite set $Q^\ominus$ of *passive states* & an *initial state* $q_0^\ominus \in Q^\ominus$;
- a finite set $Q^\oplus$ of *active states*, nonempty, disjoint from $Q^\ominus$;
- two transition functions $\delta^\ominus : Q^\ominus \times \mathrm{IM}(A) \rightharpoonup Q^\oplus$ and

$$\delta^\oplus : Q^\oplus \rightharpoonup Q^\oplus \cup \{\underbrace{\gamma(q_1^\oplus, \dots)}_{\text{this notation implies that } \gamma \text{ takes } \mathrm{rank}(\gamma) \text{ arguments}} \mid \gamma \in \Gamma,\ q_i^\oplus \in Q^\oplus\} \cup (Q^\ominus \times \mathrm{OM}(A))$$

When $A = o$, we associate to this actor the tree-generating machine:

- whose set of configurations is $Q^\oplus$;
- whose initial configuration is $\delta^\ominus(q_0^\ominus, \bullet)$;
- whose computation-step function is $\delta^\oplus$.

The output of this machine is the *tree generated by* $\Gamma \Vdash \alpha : o$.

Example 6.3. Every tree over Γ can be generated by an actor with a single passive state, and whose active states correspond to the internal nodes of the tree.

Example 6.4. We illustrate the use of multiple passive states with an actor of type $A \multimap (A \,\&\, A)$. It relays messages back and forth between the A on the left of $\multimap$ and the two As on the right; the incoming messages on the left are forwarded to a recipient determined by the state. Formally: $Q^\ominus = \{\mathsf{L}, \mathsf{R}\}$ and $Q^\oplus, \delta^\ominus, \delta^\oplus$ are chosen to satisfy

$$(\delta^\oplus \circ \delta^\ominus) : Q^\ominus \times \overbrace{\mathrm{IM}(A \multimap (A \,\&\, A))}^{\mathrm{OM}(A)+(\mathrm{IM}(A)+\mathrm{IM}(A))} \rightharpoonup Q^\ominus \times \overbrace{\mathrm{OM}(A \multimap (A \,\&\, A))}^{\mathrm{IM}(A)+(\mathrm{OM}(A)+\mathrm{OM}(A))}$$
$$(x, (\mathsf{L}, \mathfrak{m})) \mapsto (x, (\mathsf{R}, (x, \mathfrak{m})))$$
$$(x, (\mathsf{R}, (y, \mathfrak{m}))) \mapsto (y, (\mathsf{L}, \mathfrak{m}))$$

We now explain how to "plug together" two actors $\Gamma \Vdash \alpha : A$ and $\Gamma \Vdash \beta : A \multimap B$. The *application* $\beta(\alpha)$ is a sort of product[14] automaton, keeping track of the state of both α and β, and transferring messages between them: for instance, a message $\mathfrak{m} \in \mathrm{OM}(A)$ emitted by α is then received by β under the form $(\mathsf{L}, \mathfrak{m}) \in \{\mathsf{L}\} \times \mathrm{OM}(A) \subset \mathrm{IM}(A \multimap B)$. The exchanges of messages between α and β are treated as internal computation steps of $\beta(\alpha)$.

Definition 6.5 (special case of [8, Def. 3]). Let $\Gamma \Vdash \alpha : A$ and $\Gamma \Vdash \beta : A \multimap B$. For their sets of states and transition functions, we use the notations of the previous definition with indices α, β. The *application* $\beta(\alpha)$ has the sets of states

$$Q^\ominus_{\beta(\alpha)} = Q^\ominus_\alpha \times Q^\ominus_\beta \qquad Q^\oplus_{\beta(\alpha)} = (Q^\oplus_\alpha \times Q^\ominus_\beta) \cup (Q^\ominus_\alpha \times Q^\oplus_\beta)$$

Indeed, α perceives β to be part of its environment, and vice versa. For this reason, at any point during an interaction, if one of them is active, then the other must be passive.

The initial state is $(q_{0,\alpha}, q_{0,\beta})$. The transition functions are

$$\delta^\ominus_{\beta(\alpha)} : ((q^\ominus_\alpha, q^\ominus_\beta), \mathfrak{m}) \mapsto (q^\ominus_\alpha, \delta^\ominus_\beta(q^\ominus_\beta, (\mathsf{R}, \mathfrak{m})))$$

(redirecting incoming messages from the environment to β) and

$$\delta^\oplus_{\beta(\alpha)} : (q^\oplus_\alpha, q^\ominus_\beta) \mapsto \begin{bmatrix} p^\oplus \mapsto (p^\oplus, q^\ominus_\beta) \\ \gamma(p_1^\oplus, \dots) \mapsto \gamma((p_1^\oplus, q^\ominus_\beta), \dots) \\ (p^\ominus, \mathfrak{m}) \mapsto (p^\ominus, \delta^\ominus_\beta(q^\ominus_\beta, (\mathsf{L}, \mathfrak{m}))) \end{bmatrix} \big(\delta^\oplus_\alpha(q^\oplus_\alpha)\big)$$
$$(q^\ominus_\alpha, q^\oplus_\beta) \mapsto \begin{bmatrix} p^\oplus \mapsto (q^\ominus_\alpha, p^\oplus) \\ \gamma(p_1^\oplus, \dots) \mapsto \gamma((q^\ominus_\alpha, p_1^\oplus), \dots) \\ (p^\ominus, (\mathsf{L}, \mathfrak{m})) \mapsto (\delta^\ominus_\alpha(q^\ominus_\alpha, \mathfrak{m}), p^\ominus) \\ (p^\ominus, (\mathsf{R}, \mathfrak{m})) \mapsto ((q^\ominus_\alpha, p^\ominus), \mathfrak{m}) \end{bmatrix} \big(\delta^\oplus_\beta(q^\oplus_\beta)\big)$$

Definition 6.6. An *actor-based tree transducer* with input alphabet Σ and output alphabet Γ (both ranked) is specified by a type A and:

- $\Gamma \Vdash \beta_\sigma : A \multimap (A \multimap \dots (A \multimap A) \dots)$ for each $\sigma \in \Sigma$; ("transition actors", with $\mathrm{rank}(\sigma) + 1$ times A)
- $\Gamma \Vdash \beta_{\mathrm{out}} : A \multimap o$ ("output actor").

On an input $t \in \mathsf{T}_\Sigma$, it outputs the tree generated by $\Gamma \Vdash \beta_{\mathrm{out}}(\alpha_t) : o$, where we define inductively

$$\Gamma \Vdash \alpha_{\sigma(t_1,\dots)} = \beta_\sigma(\alpha_{t_1}) \dots (\alpha_{t_{\mathrm{rank}(\sigma)}}) : A \quad \text{for } \sigma \in \Sigma$$

Let us denote the components of the actor β_σ with indices: $Q^\ominus_\sigma, q^\ominus_{0,\sigma}, Q^\oplus_\sigma, \delta^\ominus_\sigma, \delta^\oplus_\sigma$; and similarly for β_{out}. By induction on $t \in \mathsf{T}_\Sigma$, one can deduce from Definition 6.5 that:

[14] It can be seen as a variant with two-way communication of the cascade product of sequential transducers.

Claim 6.7. Up to bijections that just reindex products, the states of $\beta_{\text{out}}(\alpha_t)$ are:

$$Q^{\ominus} \cong Q^{\ominus}_{\text{Dom}(t)} \times Q^{\ominus}_{\text{out}} \quad \text{where } Q^{\ominus}_X = \prod_{u \in X} Q^{\ominus}_{\text{lab}_t(u)} \text{ for } X \subseteq \text{Dom}(t)$$

$$Q^{\oplus} \cong Q^{\ominus}_{\text{Dom}(t)} \times Q^{\oplus}_{\text{out}} \cup \bigcup_{u \in \text{Dom}(t)} \{u\} \times Q^{\oplus}_{\text{lab}_t(u)} \times Q^{\ominus}_{\text{Dom}(t)\setminus\{u\}} \times Q^{\ominus}_{\text{out}}$$

The intuitive idea is that if the composite actor $\beta_{\text{out}}(\alpha_t)$ is in an active state $x = (u, \dots)$, this means that its subprocess β_σ corresponding to the node u in t (with $\sigma = \text{lab}_t(u)$) is currently active while the others are passive — and the passive states of these other subprocesses are also recorded in the $Q^{\ominus}_{\text{Dom}(t)\setminus\{u\}}$ component of x. We simulate $\beta_{\text{out}}(\alpha_t)$ by a uTHM running on the input t, whose position corresponds to the current active subprocess, and which writes in memory the passive states. We then get:

Lemma 6.8. *Every actor-based tree transducer $\mathcal{A}$ can be translated to a* uTHM $\mathcal{H}$ *such that the partial function $[\![\mathcal{H}]\!]$ extends $[\![\mathcal{A}]\!]$.*

The effective construction uses the bottom-up initialization and regular lookaround features presented in Section 3.5.

To conclude this section, we give a sufficient condition for the a priori unrestricted Hennie machine built above to be bounded-visit.

Definition 6.9. An actor $\Gamma \Vdash \alpha : A$ with states $Q^{\ominus}, Q^{\oplus}$ and transitions $\delta^{\ominus}, \delta^{\oplus}$ is *weight-reducing* when one can map each state $q \in Q^{\ominus} \cup Q^{\oplus}$ to a weight $\omega(q)$ in such a way that:

- for every $q^{\ominus} \in Q^{\ominus}$ and $\mathfrak{m} \in \text{IM}(A)$, if $\delta^{\ominus}(q^{\ominus}, \mathfrak{m})$ is defined, then its weight is strictly less than $\omega(q^{\ominus})$;
- for $q^{\oplus} \in Q$, every state that appears in $\delta^{\oplus}(q^{\oplus})$ has a weight less than or equal to $\omega(q^{\oplus})$.

An actor-based transducer is *weight-reducing* when all the actors that compose it are weight-reducing.

Lemma 6.10. *Every weight-reducing actor-based tree transducer $\mathcal{A}$ can be translated to a* THM $\mathcal{H}$ *such that $[\![\mathcal{A}]\!] \subseteq [\![\mathcal{H}]\!]$.*

7 Affine λ-transducers with additive branching

The goal of this section is to show:

Theorem 7.1. *The following devices can compute the same total tree-to-tree functions:*

(1) *tree-to-tree affine λ-transducers with additive branching;*
(2) *weight-reducing actor-based tree transducers;*
(3) *tree-to-tree Hennie machines.*

Lemma 6.10 gives us (2) $\subseteq$ (3). We show (1) $\subseteq$ (2) in Section 7.1 and (3) $\Rightarrow$ (1) in Section 7.2. In §7.3, we deduce the closure of this function class by precomposition by LSHI-MTT.

But first, we state a few definitions. As in the previous section, our grammar of types is $A, B ::= o \mid A \multimap B \mid A \,\&\, B$. We use the abbreviations (taking into account that our type system does not contain the multiplicative conjunction $\otimes$):

$$A^{\&k} = \underbrace{A \,\&\, \cdots \,\&\, A}_{k \text{ times}} \qquad A^{\otimes k} \multimap B = \underbrace{A \multimap \cdots \multimap A}_{k \text{ times}} \multimap B$$

where $\multimap$ is right-associative. It does not matter whether '&' is left- or right-associative as long as we stay consistent when encoding additive k-tuples as nested pairs (notation: $\langle T_1, \dots, T_k \rangle$).

Our typing judgments are of the form $\Phi \mid \Theta \vdash T : A$ where Φ is a context of *reusable constants*, Θ is a context of *affine variables*, T is a λ-term and A is a type. We always take the context of constants Φ to be an encoding of a ranked alphabet Γ; there are two variants:

$$\textit{additive branching:} \quad \Gamma^{\&} = \{\gamma : o^{\&\,\text{rank}(\gamma)} \multimap o \mid \gamma \in \Gamma\}$$

$$\textit{multiplicative branching:} \quad \Gamma^{\otimes} = \{\gamma : o^{\otimes\,\text{rank}(\gamma)} \multimap o \mid \gamma \in \Gamma\}$$

Our grammar of λ-terms, typing rules and reduction rules are standard, cf. e.g. [8, Figure 1], and are recalled in the appendix. By turning $\multimap$ into $\to$ and & into $\times$, we can translate our affine λ-terms to simply typed λ-terms with products, in a way that preserves reduction steps. Therefore, we inherit the strong normalization and confluence of β-reduction from the simply typed case. This ensures that every affine λ-term has a unique normal form.

Claim 7.2 ([7, §2.2.3]). The encoding that maps, for instance, the tree $a(b(c), c)$ to the λ-term $a\,\langle (b\,c),\ c\rangle$ (resp. $a\,(b\,c)\,c$) defines, for any ranked alphabet Γ, a bijection between T_Γ and the λ-terms T in normal form such that $\Gamma^{\&} \mid \varnothing \vdash T : o$ (resp. $\Gamma^{\otimes} \mid \varnothing \vdash T : o$).

We now adapt the definition of a λ-transducer from [46] (which is a simplification of the classical notion of higher-order transducer) to potentially use the encoding with additive branching for the output, but not for the input — observe the $A^{\otimes\,\text{rank}(\sigma)}$ below.

Definition 7.3. An *affine λ-transducer* $\mathsf{T}_\Sigma \to \mathsf{T}_\Gamma$ is specified by a *memory type* A and the λ-terms:

- $\Gamma^{\odot} \mid \varnothing \vdash T_\sigma : A^{\otimes\,\text{rank}(\sigma)} \multimap A$ ("transition terms") for $\sigma \in \Sigma$;
- $\Gamma^{\odot} \mid \varnothing \vdash U : A \multimap o$ ("output term").

where $\odot$ is equal to:

- & for a λ-transducer with *additive branching*;
- $\otimes$ for a λ-transducer with *multiplicative branching*.

For an input tree $t \in \mathsf{T}_\Sigma$, the output of the λ-transducer is the tree encoded by the normal form of $U\,T_t$, where we define inductively $T_{\sigma(t_1,\dots,t_{\text{rank}(\sigma)})} = T_\sigma\, T_{t_1}\, \dots\, T_{t_{\text{rank}(\sigma)}}$ for $\sigma \in \Sigma$.

Example 7.4. Here is a λ-transducer that computes `exa` (cf. §2.1).

- Its memory type is $o \,\&\, (o \multimap o)$.
- Its b-transition is $\langle a\,\langle b, b\rangle,\ \lambda y.\, a\,\langle y, y\rangle\rangle$ and its a-transition is $\lambda x_1.\, \lambda x_2.\, \langle \pi_2\, x_1\ (a\,\langle \pi_1\, x_2,\ \pi_1\, x_2\rangle),\ \lambda y.\, \pi_2\, x_1\ (a\,\langle y, y\rangle)\rangle$.
- Its output term is $\lambda x.\, \pi_1\, x$.

Note the resemblance with the MTT of Example 4.2. Morally, the two components o and $o \multimap o$ of the memory type correspond to the MTT states q_0 (of rank 0) and q_1 (of rank 1) respectively.

7.1 λ-transducers to actors: game semantics

The tree-generating game semantics from [8, §3] maps each affine λ-term T to its *strategy* $[\![T]\!]$: the set of all its possible interactions with its environment. These interactions consist of exchanging messages (i.e, playing moves, in the game metaphor) and outputting tree nodes. Furthermore, there is a partial function Strat from the actors of §6 to strategies defined in [8, §3.2]. Overloading metaphors, we say that an actor α *plays* a term T when $\text{Strat}(\alpha) = [\![T]\!]$.

We recall some properties of this "playing" relation.

- Every λ-term $\Gamma^{\&} \mid \varnothing \vdash T : A$ is played by some effectively computable actor $\Gamma \Vdash \alpha : A$ [8, Theorem 23 in Appendix B].

  - For example [8, Lemma 21 in §B], the actor from Example 6.4 plays $\Gamma^{\&} \mid \varnothing \vdash \lambda x.\, \langle x, x\rangle : B \multimap (B \,\&\, B)$.
- Furthermore, if $\Gamma \Vdash \beta : A \multimap B$ plays U, then $\beta(\alpha)$ plays $U\,T$ [8, Proposition 22 in §B].
- Finally, if $A = o$, then α generates the tree encoded by the normal form of T [8, Proposition 4].

Thanks to these properties, to translate an affine λ-transducer with additive branching into a actor-based tree transducer, we can take:

- for each input letter, a transition actor that plays the transition term for this letter,
- an output actor that plays the output term.

We still have to make sure that the actors we take are weight-reducing — which is not the case for Example 6.4. Fortunately, the following property suffices to do so.

Proposition 7.5. *For every actor $\Gamma \Vdash \alpha : A$ there exists a weight-reducing actor $\Gamma \Vdash \alpha' : A$ such that $\mathrm{Strat}(\alpha) = \mathrm{Strat}(\alpha')$.*

Proof sketch. For lack of space, we do not recall the precise definition of strategies and explain the argument informally.

Strat maps an actor to the interaction sequences that it can perform and that are valid for the game interpretation of A. As noted in [8, proof of Theorem 7], each message can occur at most once in a valid interaction in this game semantics, reflecting the affineness of the type system. By adding to the states of α a counter of the messages already received, and aborting the computation when this counter would otherwise exceed $|\mathrm{IM}(A)|$, we get a weight-reducing actor α' — the weight of a state of α' is the value of the counter — with the same valid interactions as α. □

Remark 7.6. The "restriction" on tree stack automata considered in [8, Theorem 7] is analogous to our bounded-visit condition.

7.2 THMs to λ-transducers

Let $\mathcal{H}$ be a *weight-reducing* THM with the notations of §3. We give some guiding intuitions and the definition of an affine λ-transducer with additive branching meant to be equivalent to $\mathcal{H}$. The correctness proof is relegated to the appendix.

The idea is to represent as a λ-term the "behavior" of $\mathcal{H}$ when it enters a subtree $t\restriction_u$ of the input, taking into account the current memory values for u and its descendants. This behavior is a function of the current state. After possibly producing part of the output, $\mathcal{H}$ might exit the subtree, i.e. move upwards from u into the surrounding context $t[u \leftarrow \square]$. Upon exit, the behavior should "return" the new state and a new behavior, reflecting the new memory values, to the "caller" i.e. the context. This way, the new behavior can be called the next time $\mathcal{H}$ enters $t\restriction_u$.

One subtlety is that there may be multiple return points, because $\mathcal{H}$ may produce multiple output branches in parallel. To handle this, we program in continuation-passing style: a behavior takes as argument a continuation that expects a behavior. Naively, this suggests a recursive type of behaviors, but we leverage the weight-reducing condition to unfold the recursion finitely:

$$A_0 = o^{\&Q} \qquad A_{n+1} = (A_n \multimap o^{\&Q}) \multimap o^{\&Q} \quad \text{for } n \in \mathbb{N}$$

following [8, §5]. Tuples $\langle t_q \mid q \in Q\rangle$ indexed by the set of states Q are encoded as nested pairs; one can think of them as functions from Q. The type A_n represents "n-truncated behaviors", allowing for at most n exits of $t\restriction_u$, i.e. upwards moves from u. Morally, this truncation is lossless when n is greater than the weight of the current memory value at u — we define the weight of a memory symbol $m \in M$ as $\omega(m) = |\{m' \in M \mid m' < m\}|$.

Let us fix some default leaf symbol $\gamma_0 \in \Gamma^{(0)}$. We build truncated behaviors compositionally, using λ-terms

$$\Gamma^{\&} \mid \varnothing \vdash T^{\vec{n};k}_{\sigma,m} : A_{n_1} \multimap \cdots \multimap A_{n_{\mathrm{rank}(\sigma)}} \multimap A_k$$

defined by strong induction on $n_1 + \cdots + n_{\mathrm{rank}(\sigma)} + k$:

$$T^{\vec{n};k}_{\sigma,m} = \lambda\vec{z}.\, \lambda x.\, \langle (\text{encoding of } \delta(q,\sigma,m))[\dots] \mid q \in Q\rangle$$

where $[\dots]$ is a simultaneous substitution that consists of:

- $(q', m', \uparrow) \leftarrow \pi_{q'}\left(x\left(T^{\vec{n};k-1}_{\sigma,m'}\,\vec{z}\right)\right)$ if $k \geqslant 1$, else γ_0
  Intuitively: we "return" the new state q' and the new $(k-1)$-truncated behavior, where the memory at the current node has changed from m to m', by calling the continuation x.
- $(q', m', i) \leftarrow \pi_{q'}\left(z_i\,(\lambda z_{\mathrm{new}}.\, T^{\vec{n}';k}_{\sigma,m'}\,\vec{z}'\,x)\right)$ for $i \in [\mathrm{rank}(\sigma)]$ (or just z_i without argument when $n_i = 0$)
  where $\vec{n}' = n_1, \dots, n_{i-1}, n_i - 1, n_{i+1}, \dots, n_{\mathrm{rank}(\sigma)}$
  $$T\,\vec{z}' = T\,z_1\ \dots\ z_{i-1}\ z_{\mathrm{new}}\ z_{i+1}\ \dots\ z_{\mathrm{rank}(\sigma)}$$
  Intuitively: we "call" the behavior of the subtree $t\restriction_{ui}$ rooted at the i-th child of the current node u, passing it the new state q'. Once it "returns" z_{new} (by calling the continuation that we pass), u becomes the current node again. It is as if we had just entered $t\restriction_u$, but the memory at u has become m', and the memory at ui and its descendants may also have changed as reflected in the new behavior z_{new}.

Lemma 7.7. *$[\![\mathcal{H}]\!]$ is computed by the λ-transducer with:*

- *memory type A_N for $N = |M|$;*
- *transition term $T^{N,\dots,N;N}_{\sigma,\mathsf{T}} : A_N^{\otimes \mathrm{rank}(\sigma)} \multimap A_N$ for $\sigma \in \Sigma$;*
- *output term $\lambda z.\, \pi_{q_{init}}\,(z\,(\lambda z_{\mathrm{new}}.\, \langle \gamma_0 \mid q \in Q\rangle)) : A_N \multimap o$.*

To prove this lemma we show (in the appendix) that β-reduction simulates the tree-generating machine for $\mathcal{H}$.

7.3 Precomposition by LSHI-MTT

Lemma 7.8. *The functions computed by affine λ-transducers with additive (resp. multiplicative) branching are closed under precomposition by affine λ-transducers with multiplicative branching.*

Proof. Let $\mathcal{A}, \mathcal{B}$ be two λ-transducers given respectively by:

$$\Gamma^{\otimes} \mid \varnothing \vdash T_\sigma : A^{\mathrm{rank}(\sigma)} \multimap A \qquad \Pi^{\odot} \mid \varnothing \vdash T'_\gamma : B^{\mathrm{rank}(\gamma)} \multimap B$$
$$\Gamma^{\otimes} \mid \varnothing \vdash U : A \multimap o \qquad \Pi^{\odot} \mid \varnothing \vdash U' : B \multimap o$$

where $\odot \in \{\&, \otimes\}$. Then $[\![\mathcal{B}]\!] \circ [\![\mathcal{A}]\!]$ is computed by this λ-transducer:

$$\Pi^{\odot} \mid \varnothing \vdash T_\sigma[\gamma \leftarrow T'_\gamma\ \forall \gamma \in \Gamma] : A[o \leftarrow B]^{\mathrm{rank}(\sigma)} \multimap A[o \leftarrow B]$$
$$\Pi^{\odot} \mid \varnothing \vdash \lambda x.\, U'\,(U[\gamma \leftarrow T'_\gamma\ \forall \gamma \in \Gamma]\,x) : A[o \leftarrow B] \multimap o$$

The multiplicative encoding of Γ is crucial: it is the reason why the substitution $[o \leftarrow B]$ sends the type of γ to the type of T_γ. □

This is the same argument as [46, Proposition 1.8]. In that paper, Nguyễn and Vanoni study λ-transducers for an affine λ-calculus without '&' — in particular, the branching is multiplicative. Thus, the above proposition applies to precomposition by the devices

from [46]. Their main result [46, Theorem 1.5][15] is that tree-to-tree MSO transductions are equivalent to

multiplicative affine λ-transducers $\circ$ MSO relabelings

Theorem 7.9. *The functions computed by* THMs *are closed under precomposition by* MSO *transductions.*

Proof. By the above result of [46], it suffices to check that THMs are closed under precomposition by:

MSO relabelings: cf. Corollary 3.21;
the affine λ-transducers from [46]: cf. Lemma 7.8. □

8 Postcomposition properties of THMs

We have already mentioned that our affine λ-calculus can be directly translated to the simply typed λ-calculus with products. It is known that simply typed λ-transducers with products are regularity reflecting — see the references given in Remark 1.4. Therefore:

Corollary 8.1. THMs *are regularity reflecting: the inverse image of a regular tree language by a* THM *is effectively regular.*

We can then deduce that:

Corollary 8.2. *The class of total functions computed by* THMs *is closed under postcomposition by bottom-up relabelings.*

The proof combines the regular lookaround feature of §3.5 with:

Claim 8.3. Let $\mathcal{H}$ be a THM with $[\![\mathcal{H}]\!] \colon \mathsf{T}_\Sigma \to \mathsf{T}_\Gamma$.
The function $g_{\mathcal{H}}$ that maps a configuration of $\mathcal{H}$, encoded as per Definition 3.18, to the output tree obtained by starting $\mathcal{H}$ from this configuration is itself definable by a THM.
Therefore, it preserves regular languages by inverse image.

Finally, the most technical result of this section is:

Theorem 8.4. *If $\mathcal{H}$ is a* THM *and $\mathcal{N}$ is narrow-visit on the range of $[\![\mathcal{H}]\!]$, then $[\![\mathcal{N}]\!] \circ [\![\mathcal{H}]\!]$ can be realized by some* THM.

Proof idea. We only give some high-level ideas here. The construction is detailed in the appendix.
Essentially, the composite THM virtually evaluates $\mathcal{N}$ on the tree outputted by $\mathcal{H}$: when $\mathcal{N}$ asks to explore this intermediate tree, $\mathcal{H}$ is lazily evaluated in order to produce the relevant part of its output. This is reminiscent of the composition of *reversible* two-way transducers [13, §3.1]. Indeed, our composite THM records enough information in memory to "rewind the computation" of $\mathcal{H}$, so that we may simulate upwards moves of $\mathcal{N}$ on the intermediate tree.
However, in order to simulate the read-write memory accesses by $\mathcal{N}$, we need to remember the nodes of the intermediate tree that we have already visited (and the memory values stored therein), instead of re-generating them on the fly. In other words, the composition needs to be evaluated in "call-by-need" whereas the composition of reversible two-way transducers is "call-by-name".[16]

[15] This statement by Nguyễn and Vanoni should be seen as a convenient rephrasing of a result of Gallot, Lemay and Salvati [26, Theorem 3] (see also [29, Chapter 6]).
[16] The reference semantics is call-by-value: compute the full output of $\mathcal{H}$, then feed it to $\mathcal{N}$. It is well known that call-by-name and call-by-value differ in their observable results in the presence of effects such as mutable state. What happens in our proof is that a form of call-by-need, whose laziness allows us to respect computational bounds, turns out to be a valid optimization of call-by-value in this very specific setting.

We store a representation of a "global state" in $\mathsf{T}_\Gamma(\mathrm{Conf}(\mathcal{H}, t))$ (cf. §2.3) of the computation of $\mathcal{H}$ that is distributed among the nodes of the input t. The distribution reflects the origins[17] of the nodes of the intermediate tree (in T_Γ) already generated in that global state. Since the memory at each node is finite, this means that the size of the global state must be linearly bounded by the input size. This is possible thanks to the narrow-visit condition and Corollary 3.12: a branch-outputting run of $\mathcal{N}$ explores $O(1)$ many branches, so it only requires a portion of the intermediate tree whose size is linearly bounded by the intermediate height; the latter is $O(|t|)$ because $\mathcal{H}$ is linear size-to-height increase.
Similarly, the composite THM is bounded-visit because $\mathcal{N}$ is bounded-visit and explores $O(1)$ branches of the intermediate tree, each of them generated in a bounded-visit fashion by $\mathcal{H}$. □

Corollary 8.5. LHI-MTT $\circ$ THM = THM.
As a consequence, $\bigcup_{k\in\mathbb{N}}$ LHI-MTT$^k \subset$ THM.

Proof. This reduces to postcomposition by bottom-up relabelings (Corollary 8.2) and by narrow-visit uTHMs (Theorem 8.4), thanks to the decomposition of Corollary 4.9 for LHI-MTTs. □

9 On the LHI-MTT composition hierarchy

As announced in the introduction, we show here:

Corollary 9.1. LHI-MTT$^{k-1} \subsetneq$ LHI-MTT^k *for every* $k \geqslant 1$.

We actually derive this from the stronger Theorem 9.2 below, stated in terms of *tree-walking tree transducers* (TWTs).[18] Their classical definition may be found in e.g. [11, Chapter 8]; equivalently, we may describe them here concisely as THMs:

- with a single memory symbol (the idea is that they do not write any information on the nodes: a TWT configuration consists only of a state and a position);
- whose transitions can depend on whether the current node is the root, and if not, for which $i \in \mathsf{R}_\Sigma$ it is an i-th child.

In terms of expressive power, TWT $\subset$ MTT [19, Lemma 34], which is why Corollary 9.1 follows from the following result:

Theorem 9.2. LHI-TWT$^k \not\subset$ MTT^{k-1} *for every* $k \geqslant 1$.
As a consequence, THM $\not\subset$ MTT^{k-1}.

Our proof exhibits a concrete separating function: the k-fold iteration EncPeb^k of a function EncPeb defined by Engelfriet and Maneth in [19, Section 4]. More rigorously, EncPeb is a family of similar tree-to-tree functions, one for each input alphabet — which never coincides with the output alphabet, so the composition EncPeb^k actually involves k different members of the family. To avoid cumbersome notations, we ignore these subtleties concerning alphabets. For lack of space, we do not recall the full definition of EncPeb; our only argument that requires examining it is for:

Proposition 9.3. EncPeb $\in$ LHI-TWT.

Proof. A tree-walking tree transducer computing EncPeb is built in [19, proof of Lemma 9].
Ignoring the node labels, the shape of EncPeb(t) is obtained by grafting, to each node u in the tree t, an additional subtree which is

[17] In a sense analogous to [3].
[18] In the key reference [19], they are called "0-pebble transducers", cf. Remark 9.5.

a copy of t rerooted at u by tree rotations. Therefore, a downwards path from the root of $\mathrm{EncPeb}(t)$ to a leaf corresponds to:

- a downwards path from the root of t to a node u,
- followed by the reverse upwards path from u to the root,
- followed by a downwards path from the root of t to a leaf.

Hence $\mathrm{height}(\mathrm{EncPeb}(t)) \leqslant 3 \times \mathrm{height}(t)$. □

Other than that, we mainly use the connection that EncPeb has to "pebble tree transducers" — a keyword that we syntactically manipulate by matching its occurrences in two theorem statements, with no need to access its definition.

Theorem 9.4 ([19, proof of Theorem 10]). *Every function computed by some k-pebble tree transducer is in* $\mathsf{TWT} \circ \mathrm{EncPeb}^k$.

Remark 9.5. We index the pebble hierarchy according to the convention from [18, 19]: tree-walking transducers are 0-pebble transducers. The recent literature on polyregular functions [4, 15, 37] would call them 1-pebble transducers instead.

Proof of Theorem 9.2. We show that $\mathrm{EncPeb}^k \notin \mathsf{MTT}^{k-1}$.

We use the following result [18, proof of Theorem 5]: there exists a (string-to-string) function[19] $f \notin \mathsf{MTT}^k$ which is computed by a k-pebble transducer. By Theorem 9.4, we can write $f = g \circ \mathrm{EncPeb}^k$ for some $g \in \mathsf{TWT} \subset \mathsf{MTT}$. Thus, if EncPeb^k were in MTT^{k-1}, we would get $f \in \mathsf{MTT} \circ \mathsf{MTT}^{k-1}$, reaching a contradiction. □

Let us remark that together, Corollary 8.5, Proposition 9.3 and Theorem 9.4 yield a positive result concerning the expressivity of tree-to-tree Hennie machines:

Corollary 9.6. *Every function computed by some pebble tree transducer is in* $\mathsf{TWT} \circ \mathsf{THM}$.

10 Future work

We would like to investigate further the relationship of THMs with other automata models. For instance we believe that $\mathsf{TWT} \circ \mathsf{THM}$ should be a natural tree-to-tree counterpart of the "Ariadne-transducers" from [2]. We also conjecture that $\mathsf{THM} \not\subset \bigcup_k \mathsf{MTT}^k$ — this is because our translation to λ-transducers seems to be intrinsically unsafe in the sense of higher-order recursion schemes. Also, several decision problems on THMs are still open, such as equivalence of two THMs.

References

[1] Ashwani Anand, Sylvain Schmitz, Lia Schütze, and Georg Zetzsche. 2024. Verifying Unboundedness via Amalgamation. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024*, Pawel Sobocinski, Ugo Dal Lago, and Javier Esparza (Eds.). ACM, 4:1–4:15. doi:10.1145/3661814.3662133

[2] Anonymous authors. 2026. Expregular functions. Submitted for double-blind review.

[3] Mikołaj Bojańczyk. 2014. Transducers with Origin Information. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 8573)*, Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias (Eds.). Springer, 26–37. doi:10.1007/978-3-662-43951-7_3

[4] Mikołaj Bojańczyk. 2022. Transducers of polynomial growth (invited talk). In *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, Christel Baier and Dana Fisman (Eds.). ACM, 1:1–1:27. doi:10.1145/3531130.3533326

[5] Mikołaj Bojańczyk. 2023. Folding interpretations. In *38th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2023, Boston, MA, USA, June 26-29, 2023*. IEEE, 1–13. doi:10.1109/LICS56636.2023.10175796

[6] Mikołaj Bojańczyk, Sandra Kiefer, and Nathan Lhote. 2019. String-to-String Interpretations With Polynomial-Size Output. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*. 106:1–106:14. doi:10.4230/LIPIcs.ICALP.2019.106

[7] Pierre Clairambault, Charles Grellois, and Andrzej Murawski. 2017. Linearity in Higher-order Recursion Schemes. *Proceedings of the ACM on Programming Languages* 2, POPL (Dec. 2017), 39:1–39:29. doi:10.1145/3158127

[8] Pierre Clairambault and Andrzej S. Murawski. 2019. On the Expressivity of Linear Recursion Schemes. In *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 138)*, Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 50:1–50:14. doi:10.4230/LIPIcs.MFCS.2019.50 Long version with appendices: https://hal.science/hal-02313542.

[9] Thomas Colcombet. 2017. Logic and regular cost functions. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, 1–4. doi:10.1109/LICS.2017.8005061

[10] Thomas Colcombet and Christof Löding. 2007. Transforming structures by set interpretations. *Logical Methods in Computer Science* 3, 2 (2007). doi:10.2168/LMCS-3(2:4)2007

[11] Bruno Courcelle and Joost Engelfriet. 2012. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Cambridge University Press. doi:10.1017/cbo9780511977619

[12] John Darlington and Yike Guo. 1994. Formalising Actors in Linear Logic. In *Proceedings of the 1994 International Conference on Object Oriented Information Systems, OOIS 1994, London, UK, December 19-21, 1994*, Dilip Patel, Yuan Sun, and Shushma Patel (Eds.). Springer, 37–53. doi:10.1007/978-1-4471-3016-1_3

[13] Luc Dartois, Paulin Fournier, Ismaël Jecker, and Nathan Lhote. 2017. On Reversible Transducers. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland (LIPIcs, Vol. 80)*, Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 113:1–113:12. doi:10.4230/LIPIcs.ICALP.2017.113

[14] Robert P. Dilworth. 1950. A Decomposition Theorem for Partially Ordered Sets. *Annals of Mathematics* 51, 1 (Jan. 1950), 161. doi:10.2307/1969503

[15] Gaëtan Douéneau-Tabot. 2023. *Optimization of string transducers*. Ph. D. Dissertation. Université Paris Cité. https://theses.hal.science/tel-04690881

[16] Gaëtan Douéneau-Tabot, Emmanuel Filiot, and Paul Gastin. 2020. Register Transducers Are Marble Transducers. In *45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 170)*, Javier Esparza and Daniel Král' (Eds.). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 29:1–29:14. doi:10.4230/LIPIcs.MFCS.2020.29

[17] Joost Engelfriet and Hendrik Jan Hoogeboom. 2001. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic* 2, 2 (April 2001), 216–254. doi:10.1145/371316.371512

[18] Joost Engelfriet and Sebastian Maneth. 2002. Two-Way Finite State Transducers with Nested Pebbles. In *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings (Lecture Notes in Computer Science, Vol. 2420)*, Krzysztof Diks and Wojciech Rytter (Eds.). Springer, 234–244. doi:10.1007/3-540-45687-2_19

[19] Joost Engelfriet and Sebastian Maneth. 2003. A comparison of pebble tree transducers with macro tree transducers. *Acta Informatica* 39, 9 (2003), 613–698. doi:10.1007/s00236-003-0120-0

[20] Joost Engelfriet and Sebastian Maneth. 2003. Macro Tree Translations of Linear Size Increase are MSO Definable. *SIAM J. Comput.* 32, 4 (2003), 950–1006. doi:10.1137/S0097539701394511

[21] Joost Engelfriet and Heiko Vogler. 1985. Macro Tree Transducers. *J. Comput. System Sci.* 31, 1 (1985), 71–146. doi:10.1016/0022-0000(85)90066-2

[22] Joost Engelfriet and Heiko Vogler. 1986. Pushdown Machines for the Macro Tree Transducer. *Theoretical Computer Science* 42 (1986), 251–368. doi:10.1016/0304-3975(86)90052-6

[23] Julien Ferté, Nathalie Marin, and Géraud Sénizergues. 2014. Word-Mappings of Level 2. *Theory of Computing Systems* 54, 1 (Jan. 2014), 111–148. doi:10.1007/s00224-013-9489-5

[24] Emmanuel Filiot, Nathan Lhote, and Pierre-Alain Reynier. 2025. Lexicographic Transductions of Finite Words. In *50th International Symposium on Mathematical Foundations of Computer Science (MFCS 2025) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 345)*, Paweł Gawrychowski, Filip Mazowiecki, and Michał Skrzypczak (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 50:1–50:18. doi:10.4230/LIPIcs.MFCS.2025.50

[19] What Engelfriet and Maneth actually prove in [18] is that f is not in $\mathtt{yield} \circ \mathsf{MTT}^{k-1}$, but this coincides with the subclass of MTT^k consisting of string outputs. For more details on this point, see the discussion by Kiefer, Nguyễn and Pradic [38, Section 5.1], who prove a variation on [18, Theorem 5] involving polyregular functions of quadratic size increase [38, Theorem 5.1].

[25] Delbert Ray Fulkerson. 1956. Note on Dilworth's Decomposition Theorem for Partially Ordered Sets. *Proc. Amer. Math. Soc.* 7, 4 (Aug. 1956), 701. doi:10.2307/2033375

[26] Paul Gallot, Aurélien Lemay, and Sylvain Salvati. 2020. Linear High-Order Deterministic Tree Transducers with Regular Look-Ahead. In *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic (LIPIcs, Vol. 170)*, Javier Esparza and Daniel Král' (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 38:1–38:13. doi:10.4230/LIPIcs.MFCS.2020.38

[27] Paul Gallot, Nathan Lhote, and Lê Thành Dũng Nguyễn. 2025. The structure of polynomial growth for tree automata/transducers and MSO set queries. arXiv:2501.10270 [cs.FL]

[28] Paul Gallot, Sebastian Maneth, Keisuke Nakano, and Charles Peyrat. 2024. Deciding Linear Height and Linear Size-To-Height Increase of Macro Tree Transducers. In *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia (LIPIcs, Vol. 297)*, Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 138:1–138:20. doi:10.4230/LIPICS.ICALP.2024.138

[29] Paul D. Gallot. 2021. *Safety of transformations of data trees: tree transducer theory applied to a verification problem on shell scripts.* Ph. D. Dissertation. Université de Lille. https://theses.hal.science/tel-03773108

[30] Charles Grellois. 2016. *Semantics of linear logic and higher-order model-checking.* Ph. D. Dissertation. Université Paris 7. https://theses.hal.science/tel-01311150/

[31] Bruno Guillon, Giovanni Pighizzini, Luca Prigioniero, and Daniel Průša. 2022. Converting nondeterministic two-way automata into small deterministic linear-time machines. *Information and Computation* 289 (2022). doi:10.1016/J.IC.2022.104938

[32] Bruno Guillon, Giovanni Pighizzini, Luca Prigioniero, and Daniel Průša. 2023. Weight-reducing Turing machines. *Information and Computation* 292 (2023). doi:10.1016/J.IC.2023.105030

[33] Bruno Guillon and Luca Prigioniero. 2019. Linear-time limited automata. *Theoretical Computer Science* 798 (2019), 95–108. doi:10.1016/J.TCS.2019.03.037

[34] Frederick C. Hennie. 1965. One-Tape, Off-Line Turing Machine Computations. *Information and Control* 8, 6 (1965), 553–578. doi:10.1016/S0019-9958(65)90399-2

[35] Carl Hewitt, Peter Boehler Bishop, and Richard Steiger. 1973. A Universal Modular ACTOR Formalism for Artificial Intelligence. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence. Standford, CA, USA, August 20-23, 1973*, Nils J. Nilsson (Ed.). William Kaufmann, 235–245. http://ijcai.org/Proceedings/73/Papers/027B.pdf

[36] Gerd G. Hillebrand and Paris C. Kanellakis. 1996. On the Expressive Power of Simply Typed and Let-Polymorphic Lambda Calculi. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society, 253–263. doi:10.1109/LICS.1996.561337

[37] Sandra Kiefer. 2024. Polyregular Functions – Characterisations and Refutations (invited talk). In *Developments in Language Theory - 28th International Conference, DLT 2024, Göttingen, Germany, August 12-16, 2024, Proceedings (Lecture Notes in Computer Science, Vol. 14791)*, Joel D. Day and Florin Manea (Eds.). Springer, 13–21. doi:10.1007/978-3-031-66159-4_2

[38] Sandra Kiefer, Lê Thành Dũng Nguyễn, and Cécilia Pradic. 2023. Refutations of pebble minimization via output languages. arXiv:2301.09234 [cs.FL]

[39] Joeri De Koster, Tom Van Cutsem, and Wolfgang De Meuter. 2016. 43 years of actors: a taxonomy of actor models and their key properties. In *Proceedings of the 6th International Workshop on Programming Based on Actors, Agents, and Decentralized Control, AGERE 2016, Amsterdam, The Netherlands, October 30, 2016*, Sylvan Clebsch, Travis Desell, Philipp Haller, and Alessandro Ricci (Eds.). ACM, 31–40. doi:10.1145/3001886.3001890

[40] Sige-Yuki Kuroda. 1964. Classes of Languages and Linear-Bounded Automata. *Information and Control* 7, 2 (1964), 207–223. doi:10.1016/S0019-9958(64)90120-2

[41] Guanyan Li, Andrzej S. Murawski, and Luke Ong. 2022. Probabilistic Verification Beyond Context-Freeness. In *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, Christel Baier and Dana Fisman (Eds.). ACM, 33:1–33:13. doi:10.1145/3531130.3533351

[42] Paul-André Melliès. 2017. Higher-order parity automata. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, Reykjavik, Iceland, 1–12. doi:10.1109/LICS.2017.8005077

[43] Vincent Moreau. 2025. *A topological and fibrational approach to higher-order automata.* Ph. D. Dissertation. Université Paris Cité. https://theses.hal.science/tel-05428993

[44] Lê Thành Dũng Nguyễn. 2021. *Implicit automata in linear logic and categorical transducer theory.* Ph. D. Dissertation. Université Paris XIII (Sorbonne Paris Nord). https://theses.hal.science/tel-04132636

[45] Lê Thành Dũng Nguyễn and Cécilia Pradic. 2020. Implicit automata in typed λ-calculi I: aperiodicity in a non-commutative logic. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference) (LIPIcs, Vol. 168)*, Artur Czumaj, Anuj Dawar, and Emanuela Merelli (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 135:1–135:20. doi:10.4230/LIPIcs.ICALP.2020.135

[46] Lê Thành Dũng Nguyễn and Gabriele Vanoni. 2025. Slightly Non-Linear Higher-Order Tree Transducers. In *42nd International Symposium on Theoretical Aspects of Computer Science (STACS 2025) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 327)*, Olaf Beyersdorff, Michał Pilipczuk, Elaine Pimentel, and Nguyễn Kim Thang (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 68:1–68:20. doi:10.4230/LIPIcs.STACS.2025.68

[47] Cécilia Pradic and Ian Price. 2024. Implicit automata in λ-calculi III: affine planar string-to-string functions. *Electronic Notes in Theoretical Informatics and Computer Science* Volume 4 - Proceedings of MFPS XL, Article 19 (Dec 2024). doi:10.46298/entics.14804

[48] Václav Rajlich. 1975. Bounded-Crossing Transducers. *Information and Control* 27, 4 (1975), 329–335. doi:10.1016/S0019-9958(75)90159-X

[49] Sylvain Salvati. 2015. *Lambda-calculus and formal language theory.* Habilitation thesis. Université de Bordeaux. https://theses.hal.science/tel-01253426

[50] Paweł Sobociński. 2015. Relational presheaves, change of base and weak simulation. *J. Comput. System Sci.* 81, 5 (Aug. 2015), 901–910. doi:10.1016/j.jcss.2014.12.007

[51] Igor Walukiewicz. 2016. Automata theory and higher-order model-checking. *ACM SIGLOG News* 3, 4 (2016), 13–31. doi:10.1145/3026744.3026745

A Proofs for Section 3

A.1 Proof of Proposition 3.8

Proof. We first show how to make $\mathcal{H}$ weight-reducing and N-bounded, and secondly and to make $\mathcal{H}'$ a total function with a default output. To make $\mathcal{H}'$ weight-reducing and equal to $\mathcal{H}$ on its N-bounded branch-outputting runs, we enrich the memory of $\mathcal{H}$ with a counter initialized by N at each node u. Each subsequent visit to a node u decrements its counter. Finally, when the counter reaches 0, the transition function outputs the default leaf symbol, ensuring that all branch-outputting runs are weight-reducing and N-bounded, while maintaining equivalence with $\mathcal{H}$ on N-bounded branch-outputting runs.

Secondly, the function $[\![\mathcal{H}']\!]$ is made total by first making the transition function total by defaulting to a special leaf-symbol when $\mathcal{H}$ would be undefined, and secondly handling the case where $\mathcal{H}$ would try to move up of its root by allowing $\mathcal{H}'$ to identify the root through special memory symbols. This can be done at the first transition, since the initial configuration is at the root. □

A.2 Proof of Lemma 3.14

Let $\mathcal{H}$ be a uTHM with stay instructions of transition function δ. If $t_0 = \delta(q_0, \sigma, m_0)$, then let us build t_1 by replacing each symbol $\langle q, m, \circlearrowleft\rangle$ in t_0 with $\delta(q, \sigma, m)$. Iterating this procedure to build $t_2, t_3, \dots t_k$ deterministically either : yields a finite sequence such that t_k contains no staying transition, or : loops, which we can detect by remembering, along each branch, every pair (q_i, m_i) ever reached. Let us then build $\mathcal{H}'$, a quasi-copy of $\mathcal{H}$, by removing rule r. If we built a finite sequence, replace it with a new rule $\delta_{\mathcal{H}'}(q^0, \sigma, m^0) = t_k$. These substitutions are the same that would have happened in a run of $\mathcal{H}$, so $[\![\mathcal{H}]\!] = [\![\mathcal{H}']\!]$. If, on the other end, we looped, then we do not add any transition. We know from the existence of the loop and determinism that any run that uses this rule terminate anyways, so removing the rule preserves semantics. We use this construction to remove each transition using stay instructions one by one, which will yield a uTHM without stay instructions but equivalent to $\mathcal{H}$. Finally, it also (non-strictly) decreases the number of visits to any given input node and thus keeps the bounded-visit and narrow-visit restrictions satisfied.

A.3 Proof of Lemma 3.15

Informally, the uTHM $\mathcal{H}'$ first writes in the nodes' memory the state computed by the bottom-up automaton. It then behaves like $\mathcal{H}$, using the memory symbol in addition to the input symbol to determine the relabeling.

The first phase visits $\mathrm{rank}(\sigma) + 1 = O(1)$ times each node with label σ. The first time we encounter this node (by going down), we just go further down to the first child if possible. At the $(k+1)$-th visit for $0 \leqslant k < \mathrm{rank}(\sigma)$, we have already processed the subtrees rooted at the k first children of the current node, recording these subtrees' relabeler states in the current node's memory, and we go on to visit the $(k+1)$-th child. At the $(\mathrm{rank}(\sigma)+1)$-th visit, we apply the transition function to the information we have gathered to determine the relabeling $\rho(\sigma, p)$ of the current subtree, record it in memory — it will be used as the new label of the node —, and move up to the parent in state p.

More precisely, let $\mathcal{H} = (Q, M, \top, \Xi, \Gamma, q_0, \delta)$ and $R = (P, \Sigma, \Xi, \{\alpha[\sigma] \mid \sigma \in \Sigma\}, \rho)$. Let S be the set of sequences of elements of P of length at most R_Σ. We define $\mathcal{H}' = (Q \cup P \cup \{init\}, M \times \Xi \cup \{\top\} \cup S, \top, \Sigma, \Gamma, init, \beta)$ where β is decomposed into two modes, one doing a postfix reading of the input to add the relabeling information, and one simulating $\mathcal{H}$ using the previously computed information. The first mode is defined as:

- $\beta(init, \sigma, \top) = (init, \top, 1)$ if $\mathrm{rank}(\sigma) \neq 0$.
- $\beta(init, \sigma, \top) = (p, (\top, \rho(\sigma, p)), \uparrow)$ if $\mathrm{rank}(\sigma) = 0$ and $p = \alpha[\sigma] \in P$.
- $\beta(p', \sigma, (p_1, \dots, p_k)) = ((p, (\top, \rho(\sigma, p)), \uparrow)$ if $k + 1 = \mathrm{rank}(\sigma)$ and $\alpha[\sigma](p_1, \dots, p_k, p') = p$.
- $\beta(p', \sigma, (p_1, \dots, p_k)) = ((init, (p_1, \dots, p_k, p'), k + 2)$ if $k + 1 < \mathrm{rank}(\sigma)$.

When this mode would try to move up of the root, we switch to the state q_0 and start the simulating mode which is defined as follows:

- $\beta(q, \sigma, (m, \xi)) = (q', (m', \xi), d')$ where $(q', m', d') = \delta(q, \xi, m)$.

A.4 Proof of Theorem 3.20

Consider a uTHM$^{\mathsf{R}}$ with the notations from Definition 3.19. We translate it to a uTHM whose execution simulates the original THM$^{\mathsf{R}}$ step-by-step while maintaining some additional information that replaces the regular lookaround. Its set of states (resp. memory values) is of the form $Q' = Q \times (\dots)$ (resp. $M' = M \times (\dots)$), so that the correspondence between new and old configurations amounts to applying suitable projections. Its transition function is

$$\delta' : (q', m', \sigma) \mapsto \text{apply } \ell_{q', m', \sigma} \text{ to every state call leaf of } \delta(\lambda(q', m', \sigma))$$

where:

- $\ell_{q', m', \sigma} : Q \times M \times \mathsf{D}_{\mathsf{R}_\Sigma} \to Q' \times M' \times \mathsf{D}_{\mathsf{R}_\Sigma}$ extends the calls in the original transitions with instructions to update the additional information in the state and memory;
- $\lambda : Q' \times M' \times \Sigma \to Q_{\mathrm{LA}}$ should output the correct lookaround state: for any reachable configuration, the image by δ_{LA} of the encoding of the corresponding configuration of the original THM$^{\mathsf{R}}$ should be equal to the image by λ of the argument passed to the transition function.

To define λ we decompose the computation of the lookaround state. Let $t \in \mathsf{T}_{\Sigma \times M}$ be an input tree labeled with memory values for the original machine. Let v be a node of t, with label $\langle \sigma, m \rangle$. Let us break down t with the vertex v removed into its connected components: $t = C[y \leftarrow \langle \sigma, m \rangle(s_1, \ldots, s_{\mathrm{rank}(\sigma)})]$, where C is a context with a single occurrence, at position v, of the parameter y. Let us also write $s \times \{\mathtt{notHere}\}$ for the result of applying $x \mapsto (x, \mathtt{notHere})$ to every node label in s. We define:

$$\rho_i(t, v) = \delta_{\mathrm{LA}}[s_i \times \{\mathtt{notHere}\}] \in Q_{\mathrm{LA}} \quad \text{for } i \leqslant \mathrm{rank}(\sigma), \quad \text{or } \mathtt{null} \text{ otherwise}$$
$$\varphi(t, v) = \delta_{\mathrm{LA}}[C \times \{\mathtt{notHere}\}] \in (Q_{\mathrm{LA}} \to Q_{\mathrm{LA}})$$

(The action of a single-parameter context on the states of a bottom-up automaton is defined in the usual way, with the key property that $\delta_{\mathrm{LA}}(C'[y \leftarrow s]) = \delta_{\mathrm{LA}}(C')(\delta_{\mathrm{LA}}(s))$.)

If we know this information, then we can recover the lookaround state of a configuration. Indeed, for $q \in Q$, the configuration (v, q, t) of the original machine (we use here a definition of configuration whose third component is a memory-annotated input tree) is encoded as

$$(C \times \{\mathtt{notHere}\})[y \leftarrow \langle \sigma, m, q \rangle(s_1 \times \{\mathtt{notHere}\}, \ldots, s_{\mathrm{rank}(\sigma)} \times \{\mathtt{notHere}\})]$$

whose image by δ_{LA} is equal to

$$\varphi(t, v)(\delta_{\mathrm{LA}}[(\sigma, m, q)](\rho_1(t, v), \ldots, \rho_{\mathrm{rank}(\sigma)}(t, v))) \in Q_{\mathrm{LA}}$$

We choose a definition of λ that mirrors this equation:

$$\lambda \colon (q', m', \sigma) \mapsto \varphi'(q', m')(\delta_{\mathrm{LA}}[(\sigma, m, q)](\rho'_1(q', m'), \ldots, \rho'_{\mathrm{rank}(\sigma)}(q', m')))$$

where m (resp. q) is the first component of m' (resp. q'), for some maps

$$\rho'_i \colon Q' \times M' \to Q_{\mathrm{LA}} \qquad \varphi' \colon Q' \times M' \to (Q_{\mathrm{LA}} \to Q_{\mathrm{LA}})$$

For this λ to satisfy its desired specification, it suffices to make sure that

$$\rho'_i(q', m') = \rho(t, v) \qquad \varphi'(q', m') = \varphi(t, v)$$

whenever q' and m' are respectively the state and the memory at the current head position in a configuration of the new THM that projects to (v, q, t).

To achieve this, we take the new sets of states and memory values to be respectively

$$Q' = Q \times (Q_{\mathrm{LA}} \cup (Q_{\mathrm{LA}} \to Q_{\mathrm{LA}}))$$
$$M' = M \times \mathsf{D}_{\mathrm{R}_\Sigma} \times (Q_{\mathrm{LA}} \cup \{\mathtt{null}\})^{\mathrm{R}_\Sigma} \times (Q_{\mathrm{LA}} \to Q_{\mathrm{LA}})$$

For $q' = (q, p) \in Q'$ and $m' = (m, d, r_1, \ldots, r_{\mathrm{R}_\Sigma}, f) \in M'$, we define

$$\rho'_i(q', m') = \begin{cases} p & \text{if } d = {\downarrow_i} \\ r_i & \text{otherwise} \end{cases} \qquad \varphi'(q', m') = \begin{cases} p & \text{if } d = {\uparrow} \\ f & \text{otherwise} \end{cases}$$

In other words, all the lookahead information can be read from memory, except for one value (and d indicates which one): it is the responsibility of the previous transition to compute this missing value and record it in p as part of the state.

Let us now turn to the transitions. If the original $\mathsf{uTHM}^{\mathrm{R}}$ moves in a direction d'', then, informally, the new uTHM also does so in the step-by-step simulation. Furthermore:

- In the directional component of the memory, it records d''. Indeed, if $d'' = {\downarrow_i}$, then the machine might mutate the memory of the i-th child of the current node v before the head position comes back to v (if it ever does). Thus, we cannot predict the lookaround information for the i-th subtree at the next visit of v, so we record the direction $\downarrow_i$ as our one allowed exception. A similar analysis holds when $d = {\uparrow}$ regarding the context above v.
- However, since the moves of the head are local, the memory of the rest of the tree (that is, outside of the aforementioned subtree or context) is not going to change before visiting v again. This is why the new machine also records $\rho'_1(q', m'), \ldots$ to memory, to be used at the next visit. (Here (q', m', σ) is the argument passed to the transition function.)

Formally, recall that the transition function involves a relabeling of state call leaves; we define it as:

$$\ell_{q', m', \sigma}(q'', m'', d'') = ((q'', p''), (m'', d'', r''_1, \ldots, r''_{\mathrm{rank}(\sigma)}, f''), d'')$$

where $r''_i = \rho'_i(q', m')$, $f'' = \varphi'(q', m')$ and

$$p'' = \begin{cases} \delta_{\mathrm{LA}}[(\sigma, m'', \mathtt{notHere})](r''_1, \ldots, r''_{\mathrm{rank}(\sigma)}) \in Q_{\mathrm{LA}} & \text{if } d'' = {\uparrow} \\ f'' \circ \delta_{\mathrm{LA}}[(\sigma, m'', \mathtt{notHere})](r''_1, \ldots, r''_{i-1}, -, r''_{i+1}, \ldots, r''_{\mathrm{rank}(\sigma)}) & \text{if } d'' = {\downarrow_i} \end{cases}$$

Let us explain the case $d'' = {\uparrow}$ of this last equation (the other case is similar). Let (v, q', t') be a reachable configuration, with (σ, m') being the label of v in t'. By moving in the direction $\uparrow$ we go to the parent w of the node v. It must be the case that w has already been visited before (in order to reach v from the root), and the last visit to w has concluded with a move towards the child v, in direction $\downarrow_i$ for some i. The responsibility of the transition from (v, q', t') to the successor configuration $(w, (q'', p''), s')$ is therefore to make sure that $p'' = \rho_i(s, w)$

where s is the projection of s'. By unfolding the definitions, and noting that s' and t' differ only at the node v, one can check that the above choice for p'' works.

A final detail: our new uTHM uses the bottom-up initialization feature from Lemma 3.15. If the input subtree rooted at a node is $\sigma(t_1, \dots, t_k)$ then its initial memory is

$$(\top, \delta_{\mathrm{LA}}[t_1 \times \{(\top, \texttt{notHere})\}], \dots, \delta_{\mathrm{LA}}[t_k \times \{(\top, \texttt{notHere})\}], \underbrace{\texttt{null}, \dots, \texttt{null}}_{\mathrm{R}_\Sigma - k \text{ times}}, \mathrm{id})$$

This guarantees that the correspondence between ρ, φ and ρ', φ' holds at the first visit of each node.

B Proofs for Section 4

We prove first the first statement of Theorem 4.7, namely that any MTT can be translated into an equivalent uTHM.

Proof of Theorem 4.7. Before formally defining the equivalent uTHM $\mathcal{H}$, we explain how it simulates $\mathcal{M}$. The main difference between MTT and uTHM is that the MTT processes its input once while having pointers to different parts of its output, while the uTHM can process its input several times, but only produces its output in a top-down fashion. On the other hand, a uTHM has access to a finite memory for each node and for each branch of the output. In order to simulate $\mathcal{M}$, the uTHM $\mathcal{H}$ applies, for each branch, an outside-in evaluation process, meaning that it always evaluates the topmost call state of a branch. It also stores in each node the sequence of remaining calls to be processed.

$\mathcal{H}$ starts simulating $\mathcal{M}$ at the root, applying its initial transition. When $\mathcal{H}$ reaches a node from its parent, it does so in a given state q of $\mathcal{M}$. Together with the node label σ, it gives a transition (q, σ) of $\mathcal{M}$, where $r \in \mathsf{T}_{\Gamma \cup \langle Q, X_j \rangle}(Y_k)$ with j being the rank of σ and k being the rank of q. The corresponding transition of $\mathcal{H}$ produces a tree where each branch b is truncated at the first $\langle Q, X_j \rangle$ symbol $\langle q', x_i \rangle$. It launches a head in state q' on direction i, and writes on its node the truncated part of the branch b. If there is no $\langle Q, X_j \rangle$ symbol, then either the branch is done and nothing else needs to be done, or there is a Y_i symbol, in which case the reading head moves up, back to its parent, in a state indicating the branch i being produced.

When $\mathcal{H}$ reaches a nodes from one of its children, then it contains a remainder of the branch to produce, and the state indicates which branch to follow. It then acts similarly to above, according to the state at the top of the branch, minus the state call at the root whose simulation has finished. The computation stops when all branches have reached their respective ends.

Formally, let $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \delta)$. We define a uTHM $\mathcal{H} = (P, N, \top, \Sigma, \Gamma, p_0, \alpha)$ as follows. We set the set of states $P = Q \cup [\mathrm{R}_Q]$ and $p_0 = q_0$. To define the memory N, we need to consider the possible productions of M. We first consider Out to be the set of output trees $\delta(q, \sigma)$ for q, σ in Q, Σ. Let Out' be the set of subtrees of elements of Out that are rooted at an element of $\langle Q, X_j \rangle$. The set of memory symbols N is finally defined as $\{\top\} \cup Out'$.

It remains to define the transition function α:

- $\alpha(q, \sigma, \top)$ produces the tree $\delta(q, \sigma)$ where each branch is truncated before the first symbol that does not belong to Γ, and for each truncated subtree t, we set a processing head:
  - $\langle q, t, j \rangle$ if the top symbol belongs is $\langle q, x_j \rangle$,
  - $\langle i, \top, \uparrow \rangle$ if t is reduced to a parameter Y_i.
- $\alpha(i, \sigma, t)$ behaves similarly to the first case, using the i-th branch of t (without its root) instead of $\delta(q, \sigma)$. Note that in this case $\alpha(i, \sigma, t)$ nonetheless produces the possible finite Γ branches attached to the root of t.
- the case $\alpha(q, \sigma, t)$ is undefined as it does not appear since $\mathcal{H}$ only reaches a node in a state in Q if its memory is emptied (i.e. in memory state $\top$).
- Note that the uTHM can only reach a node in a state i coming from one of its children, so the case $(i, \sigma, \top)$ does not happen either.

We now prove the correctness of the construction, namely that $[\![\mathcal{M}]\!] = [\![\mathcal{H}]\!]$. As mentioned earlier, we in fact prove that $\mathcal{H}$ realizes an outside-in evaluation of calls of $\mathcal{M}$ on every branches, meaning that it produces its output similarly to $\mathcal{M}$ does by always applying the topmost derivation possible on each branch, called the outside-in derivation. This proves the correctness of simulation since all derivations of MTTs lead to the same output.

In order to prove this, we describe what outside-in partial derivations of $\mathcal{M}$ look like. The MTT $\mathcal{M}$ processes its input in a top-down fashion, stacking state calls on the partial output. A state call $\langle q, s \rangle$ is replaced, by a derivation step, by a tree whose state calls are children of s. If the topmost state call is processed each step (branch-wise), then the state calls of children of a node are processed before its own state calls. The resulting derivation is a tree where for any branch, all state calls of siblings of a given node are grouped together. More, the depths of state calls in the derivation tree are inverted compared to the depths of the node they point to.

Let us consider a partial output t' of $\mathcal{M}$ over some input t, meaning that $\langle q_0, t \rangle \Rightarrow_M^* t'$, obtained using the outside-in derivation. We show that $\mathcal{H}$ reaches a configuration where it has produced the maximal prefix t'' of t' that belongs to T_Γ, and for each topmost node $n = \langle q, s \rangle$ appearing of t' (hence not in t''), $\mathcal{H}$ has a processing head on s in state q. Moreover for each node s' and such node n, the memory of s' for the branch denoted by n is $\top$ if s' is a descendant of s, and otherwise contains the maximal subtree of t'' below n containing symbols of Γ and all state calls on its children, and such that its root is a state call.

At the start of the computation of $\mathcal{M}$, nothing is produced and the initial configuration of $\mathcal{H}$ satisfies this property.

Now assume that property holds true at some point of the computation, and let $n = \langle q, s\rangle$ be the topmost node of some branch b of t' that does not belong to Γ. The outside-in derivation of M on this branch applies to n and replaces $\langle q, s\rangle$ by $\delta(q, \mathrm{lab}(s))$. By induction hypothesis, $\mathcal{H}$ has a processing head of s in state q. Since n is the topmost state call and points to s, there is no state call related to any children of s, so the memory state of s is $\top$. The derivation of $\mathcal{M}$ replaces $\langle q, s\rangle$ with the context $c = \delta(q, \mathrm{lab}(s))$ and plugs it in t' to form a new partial output t''. Similarly, the corresponding transition $\alpha(q, \mathrm{lab}(s), \top)$ produces the Γ part of c. For any branch of c with a state call, let $n' = \langle q', si\rangle$ be its topmost state call. $\mathcal{H}$ launches a processing head on si in state q', and stores on s the (possibly empty) tree below n', satisfying the invariant for the unfinished branch. If a branch of c is entirely comprised of Γ-symbols, the branch is entirely computed and the invariant is also satisfied. Finally, if a branch of c ends in some parameter Y_i, the possible topmost state call of t'' of this branch relates to a node s' above s. The machine $\mathcal{H}$ moves up, in state i, up to the first ancestor node that still contains a state call in its memory state, possibly updating the state i. Thanks to the invariant, it corresponds to s' and the memory state of s' contains the remainder of the branches to produce and the state indicates which branch is being produced. The transition applied here then launches a processing head on the child of s' corresponding to the state call, in the corresponding state, and removes from t the produced part, satisfying the invariant.

This proves that $[\![\mathcal{M}]\!] \subseteq [\![\mathcal{H}]\!]$ when seen as relations. And since $\mathcal{M}$ and $\mathcal{H}$ realizes total functions, this means that $[\![\mathcal{M}]\!] = [\![\mathcal{H}]\!]$, which concludes the proof. □

The second point of Theorem 4.7 is a consequence of the more general following property: each call of the machine $\mathcal{M}$ to a node u corresponds exactly to one downward visit of the machine $\mathcal{H}$ to the same node u. Indeed by construction of $\mathcal{H}$, each downward visit of $\mathcal{H}$ to a node u consumes a call stored by its parent. As the memory is constructed from transitions of $\mathcal{M}$ that are yet to be expanded, we get that calls of $\mathcal{M}$ are in bijection to downward visits of $\mathcal{H}$. Then the number of downward visits at an antichain S of nodes of an input t, for a branch-outputting run of $\mathcal{H}$, is the number of calls of $\blacksquare$ that can be stored in nodes of S by the branch-outputting run, which is bounded by the maximal number of $\blacksquare$ that can be stacked during a computation of $\mathcal{M}^{\blacksquare}$.

C Proofs for Section 5

The goal of this section is to construct, given a finite visits THM, an MSO-SI that realizes the same function. The main ingredient of the proof is the use of local profiles, which are abstractions of the run of a THM at a given input position regarding a given output position. Thanks to the finite visits property, local profiles are bounded. We use them for the coloring X_i of the MSO set interpretation. A global profile of an input then associates to each input position a local profile. The domain formula of the MSO-SI states that a global profile is coherent, meaning that all local profiles concern a same output position.

To define the successor and ancestor formulas, we rely on the fact that the local profiles for a given output position are extensions of the local profiles for its ancestors, as THM produces their output in a top-down fashion. In particular, this means that a partial run up to producing a given output position has to produce all its ancestors.

For the remainder of this section, we fix a (finite visits) THM $\mathcal{H} = (Q, M, \top, \Sigma, \Gamma, q_{init}, \delta)$. We denote by B (resp. PB) the set of (resp. prefix) branches in the right hand side of transition rules of $\mathcal{H}$, i.e. the set $\bigcup_{q,\sigma,m} B(\delta(q, \sigma, m))$ (resp. $\bigcup_{q,\sigma,m} PB(\delta(q, \sigma, m))$).

C.1 Hennie Profiles

A node of the output tree $\mathcal{H}(t)$ for some input tree t is uniquely characterized by the branch-outputting run leading to it. Said branch is determined by a sequence of transitions of $\mathcal{H}$ on t, i.e. by an input node, a state and a memory state. Should we call a *visit pair* a pair composed by a state of Q and a branch of B_P, the trace left on an input node of producing an output node is a finite sequence of visit pairs. Formally, we set:

Definition C.1 (Local profiles). For $\sigma \in \Sigma$, a *local $\mathcal{H}$ Hennie profile over σ* is a finite sequence $(q_1, b_1), \dots (q_k, b_k)$ of visit pairs that satisfies the following conditions:

(1) k is smaller of equal to maximal number of visits of $\mathcal{H}$.
(2) for all $0 < i < k - 1$, b_i is a branch of B and its leaf is a reading head,
(3) $b_0 \in B(\delta(q_0, \langle \sigma, \top\rangle))$, and
(4) $\forall i < k - 1, b_{i+1} \in B(\delta(q_{i+1}, \langle \sigma, m_i\rangle))$, where m_i is defined inductively as the memory set by the previous transition.

We call $\Lambda_{\mathcal{H}}(\sigma)$ the set of local $\mathcal{H}$ Hennie profiles over σ, or simply $\Lambda(\sigma)$ when $\mathcal{H}$ is clear from context, and $\Lambda_{\mathcal{H}} = \cup_{\sigma\in\Sigma}\Lambda_{\mathcal{H}}(\sigma)$ the set of all local $\mathcal{H}$ Hennie profiles.

Intuitively, a local profile requires that at each visit of the input position, the memory state corresponds to the one written by the last visit (or $\top$ for the first visit). We also require that each visit but potentially the last selects a complete and non terminating branch of the production of the transition.

Producing a an output node, and the branch leading to it, might require to visit several if not all nodes of the input, generating local profiles for all input nodes. This gives rise to the notion of global profiles:

Definition C.2 (Global profiles). A *global $\mathcal{H}$ Hennie profile α over t* is a mapping from nodes of t to local $\mathcal{H}$ Hennie profiles respecting the condition: $\forall u \in \mathrm{Dom}(t), \alpha(u) \in \Lambda_{\mathcal{H}}(\mathrm{lab}_t(u))$. We call $N_{\mathcal{H}}(t)$ the set of global $\mathcal{H}$ Hennie profiles over t.

Global profiles assigns to each node a local profile corresponding to its label. The next notion allows us to describe global profiles that effectively characterize partial runs.

Definition C.3 (Execution order). Consider a global Hennie profile α. For u a node of t and i an integer, we denote by (u, i) the i-th visiting pair (q_i, b_i) of $\alpha(u)$ if it exists. An *execution order* of *mnu* is a total ordering $\leqslant_e$ of all visiting pairs (u, i) such that:

- For all u, $(u, i) \leqslant_e (u, j)$ iif $i \leqslant j$,
- For all visiting pair (q, b) of some node u, with the leaf of b being some head $\langle q', m, d\rangle$ if, and only if, there exists some visiting pair (q', b') in ud and (q', b') is the $\leqslant_e$-successor of (q, b).

We finally define coherent full profiles that we later prove to be the colorings we need to define the MSO-SI transformation.

Definition C.4 (Coherence). A global profile is said to be coherent if it admits an execution order whose smallest element is the first visiting pair of the root, with state q_{init}.

Proposition C.5. *In a coherent global profile ν over t, there exists a unique $u \in \mathrm{Dom}(t)$ that contains a branch b which endpoint is not labeled in $\langle Q, M, D\rangle$.*

Proof. A pair (q, b) has a successor if and only if the endpoint of b is labeled in $\langle Q, M, D\rangle$, but the order $\leqslant_e$ needs to be total, so there must be exactly one visiting pair whose endpoint is not in $\langle Q, M, D\rangle$: the maximum (u, i) of $\leqslant_e$. □

The previous definitions amount to this key lemma that links coherent global profiles with output nodes.

Lemma C.6. *Given an input t of $\mathcal{H}$, there is a bijection between the output nodes of $[\![\mathcal{H}]\!](t)$ (or equivalently branch-outputting runs with a distinguished node in the last branch) and the set of coherent $\mathcal{H}$-profiles over t.*

Proof. First, let $\alpha : \mathrm{Dom}(t) \to \Lambda_{\mathcal{H}}$ be a coherent global profile with execution order $\leqslant_e$. Using Proposition C.5, there exists exactly one node u with a visiting pair $(q, b) \in \alpha(u)$ such that the endpoint of b is an element of Γ. We prove by induction on the execution order that the concatenation of the branches of the visiting pairs along the execution order is a branch-outputting run of $[\![\mathcal{H}]\!](t)$ with the last transition potentially truncated. Then by Proposition C.5 the endpoint of this truncated branch-outputting run is an element of Γ, thus an output node of $[\![\mathcal{H}]\!](t)$. Initially, the smallest element of $\leqslant_e$ is, by definition of coherence, the first visiting pair (q_{init}, b) of the root of t, where b is a (potentially truncated) branch of $\delta(q_{init}, \mathrm{lab}(\varepsilon), \top)$. Hence at the first step b is a possibly truncated branch-outputting run of the initial transition of $\mathcal{H}$ on t.

Now suppose that we have constructed a branch-outputting run $b_1 \dots b_n$ up to some visiting pair (q, b_n) of some node u of t, and let $\langle q', m, d\rangle$ be the endpoint of b_n. Note that if the endpoint of b is an element γ of Γ, by proposition C.5 we have exhausted the execution order and we can conclude. By definition of the execution order there is a visiting pair (q', b') in ud that is the successor of the visiting pair (q, b_n). Then b' is a branch (or a prefix of a branch, in which case we can also conclude) of $\delta(q', \mathrm{lab}(ud), m')$ where m' is the last memory state written at ud by the previous visiting pair (or $\top$ if it is the first visiting pair of the node). We concatenate b' to the already produced branch-outputting run by replacing $\langle q', m, d\rangle$ by b'. As we followed the transition of $\mathcal{H}$, we successfully extended our induction.

Conversely, given an output node o of $[\![\mathcal{H}]\!](t)$, we construct a coherent global profile as follows. Using the branchwise semantics of THM, consider the unique sequence $x_0 \xrightarrow{b_0} \dots \xrightarrow{b_{n-1}} x_n$ where $x_0 = C_{init}$ and $\pi_2(b_0 \dots b_{n-1}\gamma)$ is the step by step branch computation from the initial configuration up to the computation step that output node o. Let $x_i = (u_i, q_i, t_i)$ for all $i < n$ Then we construct the local profiles as follows: local profiles are initialized as the empty sequence. Then for i ranging from 0 to $n-2$, we add to the node u_i the visiting pair (q_i, b_i), and we add to u_{n-1} the pair (q_{n-1}, b') where b' is b_{n-1} truncated at o. We prove that the set of sequences we defined is a coherent global profile. First, they are local profiles since :

(1) each profile has a length corresponding to the number of visits of the node, it is hence bounded by the maximal number of visits.
(2) Each b_i is one computation step of $\mathcal{H}$, so it is a full, branch of a right-hand side of δ, and ends with a reading head except for the last configuration x_n,
(3) The first visit of a given input node has the memory set to $\top$,
(4) The branch b_i is the result of applying the transition of the reading head in state q_i, with the memory set by the last visit of the position.

It is then a global profile since all visiting pairs of the local profile apply transitions of the label of the corresponding node. The execution order is naturally defined by the sequence of b_i ordering their corresponding pairs. By construction it satisfies both conditions of the definitions. And finally it is coherent since we start at the initial configuration. □

We are now able to define the labeling formulas ϕ_γ.

Lemma C.7. *We can construct* MSO *formulas $\phi_\gamma(\{X_\lambda \mid \lambda \in \Lambda_{\mathcal{H}}\})$, for $\gamma \in \Gamma$, such that for any input t and assignment $\alpha : \mathrm{Dom}(t) \to \Lambda_{\mathcal{H}}$, $(t, \alpha) \models \phi_\gamma(\{X_\lambda \mid \lambda \in \Lambda_{\mathcal{H}}\})$ if, and only if, α is a coherent global profile in bijection with an output node labeled by γ.*

Proof. We first define a formula $\phi(\{X_\lambda \mid \lambda \in \Lambda_{\mathcal{H}}\})$ such that $(t, \alpha) \models \phi$ if, and only if, α is a coherent global profile. First, the formula states that the sets X_i form a partition of the domain of t. The predicate $x \in^X_\lambda$ is interpreted as the node quantified by x is labeled by local profile λ. The global profile condition is simply the subformula $\psi_{gp} = \forall x, \wedge_{\sigma\in\Sigma}\big(\sigma(x) \to \vee_{\lambda\in\Lambda_{\mathcal{H}}(\sigma)} x \in X_\lambda\big)$. Next, we define the execution order via a formula $\psi_{i,j}(x, y)$ that is satisfied if the successor in the execution order of the i-th visiting pair of the profile of x is the j-th visiting pair of y. As the successor in the execution order is always a neighbor (self, child or parent) and only depends on the local profile of the two which is finite information, the formula $\psi_{i,j}(x, y)$ can be written as a disjunction of types on x and y in its direct neighborhood. Then the existence of the execution order is the formula stating that the transitive closure the successor formula is a total order, i.e. transitive, reflexive and antisymmetric, which are all MSO definable properties. Finally, the coherence is defined by stating that the minimal element of the execution order is the first visiting pair of the root. The label γ correspond to the label of the maximal element of the execution order. □

We conclude this Section by giving the reduction from THM to MSO-SI.

Proof of Theorem 5.3. Similarly to the proof of Lemma C.7, we set $c = |\Lambda_{\mathcal{H}}|$, the number of second order variables of T is equal to the number of local profiles. The labeling formulas ϕ_γ are taken from Lemma C.7.

Finally, we need to define the ϕ_i formula interpreting the i-th child relation as well as the ϕ_{anc} formula interpreting the ancestor relation. A global profile α describes the ancestor of an other global profile α' if, and only if, for all node u, $\alpha'(u)$ is obtained from $\alpha(u)$ by adding visiting pairs to it, and by enriching the maximal element of the execution order. This is a local property that can be tested separately on each node, verifying that $\alpha'(u)$ is indeed an extension of $\alpha(u)$, and both are coherent global profiles. Similarly, a coherent global profile α' describes the i-th child of an other coherent global profile α if α' is obtained from α by replacing the maximal element of the execution order (q, b) with b ending in some γ by (q, b') where b' is b enriched with the i-th child in the transition $\delta(q, \langle\sigma, m\rangle)$. Then either the leaf of b' is an element of Γ, or the local profiles of α' are local profile of α enriched with visiting pairs whose branches are reduced to a single node. The fact that α' is a coherent global profile assures then that all these branches added are reduced to a single reading head, except for the maximal element. As these properties are MSO definable by a disjunction over the finite set of local profiles, we are able to define formulas ϕ_i for the successor. □

D Proofs for Section 6

D.1 Proof and further details for Claim 6.7

We first look at the k-ary application of actors in general. Recall that $\mathrm{OM}(A \multimap B) = \{\mathsf{L}\} \times \mathrm{IM}(A) \cup \{\mathsf{R}\} \times \mathrm{OM}(B)$ and that $\mathrm{IM}(A \multimap B)$ is defined similarly. We use the abbreviation $\mathsf{arg}_i\mathfrak{m} = (\mathsf{R}, \dots (\mathsf{R}, (\mathsf{L}, \mathfrak{m})) \dots)$ with $i-1$ times R.

Claim D.1. Let $k \in \mathbb{N}$. Let $\Gamma \Vdash \alpha : B_1 \multimap \cdots \multimap B_k \multimap A$ and $\Gamma \Vdash \beta_i : B_i$ for $i \in [k]$. Then:

$$Q^\ominus_{\beta(\alpha_1)\dots(\alpha_k)} = Q^\ominus_\beta \times \prod_{i=1}^k Q^\ominus_{\alpha_i} \qquad Q^\oplus_{\beta(\alpha_1)\dots(\alpha_k)} = Q^\oplus_\beta \times \prod_{i=1}^k Q^\ominus_{\alpha_i} \cup \bigcup_{i=1}^k Q^\ominus_\beta \times Q^\ominus_{\alpha_1} \times \cdots \times Q^\ominus_{\alpha_{i-1}} \times Q^\oplus_{\alpha_i} \times Q^\ominus_{\alpha_{i+1}} \cdots \times Q^\ominus_{\alpha_k}$$

where $\cong$ denotes bijections that only reindex products. Intuitively, at any time, at most one of the subprocesses is active.

The initial state is $(q^\ominus_{0,\beta}, q^\ominus_{0,\alpha_1}, \dots, q^\ominus_{0,\alpha_k})$ and the transitions are:

$$\delta^\ominus_{\beta(\alpha_1)\dots(\alpha_k)} \colon ((q^\ominus_\beta, q^\ominus_{\alpha_1}, \dots, q^\ominus_{\alpha_k}), \mathfrak{m}) \mapsto (\delta^\ominus_\beta(q^\ominus_\beta, (\mathsf{R}, \dots (\mathsf{R}, \mathfrak{m}) \dots)), q^\ominus_{\alpha_1}, \dots, q^\ominus_{\alpha_k}) \text{ with } k \text{ times } \mathsf{R}$$

$$\delta^\oplus_{\beta(\alpha_1)\dots(\alpha_k)} \colon (q^\ominus_\beta, q^\ominus_{\alpha_1}, \dots, q^\oplus_{\alpha_i}, \dots q^\ominus_{\alpha_k}) \mapsto \begin{bmatrix} p^\oplus \mapsto (q^\ominus_\beta, q^\ominus_{\alpha_1}, \dots, p^\oplus, \dots q^\ominus_{\alpha_k}) \\ \gamma(p^\oplus_1, \dots) \mapsto \gamma((q^\ominus_\beta, q^\ominus_{\alpha_1}, \dots, p^\oplus_1, \dots q^\ominus_{\alpha_k}), \dots) \\ (p^\ominus, \mathfrak{m}) \mapsto (\delta^\ominus_\beta(q^\ominus_\beta, \mathsf{arg}_i\mathfrak{m}), q^\ominus_{\alpha_1}, \dots, p^\ominus, \dots q^\ominus_{\alpha_k}) \end{bmatrix} (\delta^\oplus_{\alpha_i}(q^\oplus_{\alpha_i}))$$

$$(q^\oplus_\beta, q^\ominus_{\alpha_1}, \dots, q^\ominus_{\alpha_k}) \mapsto \begin{bmatrix} p^\oplus \mapsto (p^\oplus, q^\ominus_{\alpha_1}, \dots, q^\ominus_{\alpha_k}) \\ \gamma(p^\oplus_1, \dots) \mapsto \gamma((p^\oplus, q^\ominus_{\alpha_1}, \dots, q^\ominus_{\alpha_k}), \dots) \\ (p^\ominus, \mathsf{arg}_i\mathfrak{m}) \mapsto (p^\ominus, q^\ominus_{\alpha_1}, \dots, \delta^\ominus_\alpha(q^\ominus_{\alpha_i}, \mathfrak{m}), \dots, q^\ominus_{\alpha_k}) \\ (p^\ominus, (\mathsf{R}, \dots (\mathsf{R}, \mathfrak{m}) \dots)) \mapsto ((p^\ominus, q^\ominus_{\alpha_1}, \dots, q^\ominus_{\alpha_k}), \mathfrak{m}) \end{bmatrix} (\delta^\oplus_\beta(q^\oplus_\beta))$$

Proof. The case $k = 0$ is tautological, while $k = 1$ is the definition of application, up to the bijection that swaps pairs. Inductive case:

$$Q^\ominus_{\beta(\alpha_1)\dots(\alpha_{k+1})} = Q^\ominus_{\alpha_{k+1}} \times Q^\ominus_{\beta(\alpha_1)\dots(\alpha_k)} = Q^\ominus_{\alpha_{k+1}} \times \left(Q^\ominus_\beta \times \prod_{i=1}^k Q^\ominus_{\alpha_i}\right) \cong Q^\ominus_\beta \times \prod_{i=1}^{k+1} Q^\ominus_{\alpha_i}$$

$$
\begin{aligned}
Q^{\ominus}_{\beta(\alpha_1)\dots(\alpha_{k+1})} &= Q^{\oplus}_{\alpha_{k+1}} \times Q^{\ominus}_{\beta(\alpha_1)\dots(\alpha_k)} \cup Q^{\ominus}_{\alpha_{k+1}} \times Q^{\oplus}_{\beta(\alpha_1)\dots(\alpha_k)} \\
&= Q^{\oplus}_{\alpha_{k+1}} \times Q^{\ominus}_{\beta} \times \prod_{i=1}^{k} Q^{\ominus}_{\alpha_i} \cup Q^{\ominus}_{\alpha_{k+1}} \times \left(Q^{\oplus}_{\beta} \times \prod_{i=1}^{k} Q^{\ominus}_{\alpha_i} \cup \bigcup_{i=1}^{k} Q^{\ominus}_{\beta} \times Q^{\ominus}_{\alpha_1} \times \cdots \times Q^{\ominus}_{\alpha_{i-1}} \times Q^{\oplus}_{\alpha_i} \times Q^{\ominus}_{\alpha_{i+1}} \cdots \times Q^{\ominus}_{\alpha_k} \right) \\
&\cong Q^{\ominus}_{\beta} \times \prod_{i=1}^{k} Q^{\ominus}_{\alpha_i} \times Q^{\oplus}_{\alpha_{k+1}} \cup Q^{\oplus}_{\beta} \times \prod_{i=1}^{k+1} Q^{\ominus}_{\alpha_i} \cup \bigcup_{i=1}^{k} Q^{\ominus}_{\beta} \times Q^{\ominus}_{\alpha_1} \times \cdots \times Q^{\ominus}_{\alpha_{i-1}} \times Q^{\oplus}_{\alpha_i} \times Q^{\ominus}_{\alpha_{i+1}} \cdots \times Q^{\ominus}_{\alpha_{k+1}} \\
&\cong Q^{\oplus}_{\beta} \times \prod_{i=1}^{k+1} Q^{\ominus}_{\alpha_i} \cup \bigcup_{i=1}^{k+1} Q^{\ominus}_{\beta} \times Q^{\ominus}_{\alpha_1} \times \cdots \times Q^{\ominus}_{\alpha_{i-1}} \times Q^{\oplus}_{\alpha_i} \times Q^{\ominus}_{\alpha_{i+1}} \cdots \times Q^{\ominus}_{\alpha_{k+1}}
\end{aligned}
$$

The expressions for the transitions can also be derived inductively from the definitions. □

Let us now fix an actor-based tree transducer with the notations of Definition 6.6. By structural induction on $t \in \mathsf{T}_\Sigma$, and using the above claim, we can deduce expressions (up to product reindexing) of the states and transitions of α_t involving $\mathrm{Dom}(t)$, using the fact that

$$\mathrm{Dom}(\sigma(t_1, \dots, t_{\mathrm{rank}(\sigma)})) \cong \{\varepsilon\} + \mathrm{Dom}(t_1) + \cdots + \mathrm{Dom}(t_{\mathrm{rank}(\sigma)})$$

Finally using the definition of application we can describe the states and transitions of $\beta_{\mathrm{out}}(\alpha_t)$. The states are given in the statement of Claim 6.7. The initial state is $q_0^{\ominus} = ((q^{\ominus}_{0,\mathrm{lab}_t(u)})_{u\in\mathrm{Dom}(t)}, q^{\ominus}_{0,\mathrm{out}})$, and we have $\delta^{\ominus}((\vec{q}, q^{\ominus}_{\mathrm{out}}), \bullet) = (\vec{q}, \delta^{\ominus}_{\mathrm{out}}(q^{\ominus}_{\mathrm{out}}, \bullet))$.

For an active state of the form $x = (u, q^{\oplus}, (q^{\ominus}_v)_{v\neq u}, q^{\ominus}_{\mathrm{out}})$, we have $\delta^{\oplus}(x) = f_x(\delta^{\oplus}_{\mathrm{lab}_t(u)}(q^{\oplus}))$ where

$$
\begin{aligned}
f_x \colon \gamma(p_1^{\oplus}, \dots, p^{\oplus}_{\mathrm{rank}(c)}) &\mapsto \gamma((u, p_1^{\oplus}, (q^{\ominus}_v)_{v\neq u}, q^{\ominus}_{\mathrm{out}}), \dots) \\
p^{\oplus} &\mapsto (u, p^{\oplus}, (q^{\ominus}_v)_{v\neq u}, q^{\ominus}_{\mathrm{out}}) \\
(q^{\ominus}_u, \mathtt{arg}_i \mathfrak{m}) &\mapsto (ui, \delta^{\ominus}_{\mathrm{lab}_t(ui)}(q^{\ominus}_{ui}, (\mathsf{R}, \dots (\mathsf{R}, \mathfrak{m}))), (q^{\ominus}_v)_{v\neq ui}, q^{\ominus}_{\mathrm{out}}) \\
(q^{\ominus}_u, \underbrace{(\mathsf{R}, \dots (\mathsf{R}, \mathfrak{m}) \dots)}_{\mathrm{rank}(\mathrm{lab}_t(u)) \text{ times } \mathsf{R}}) &\mapsto \begin{cases} (w, \delta^{\ominus}_{\mathrm{lab}_t(w)}(q^{\ominus}_w, \mathtt{arg}_i \mathfrak{m}), (q^{\ominus}_v)_{v\neq w}, q^{\ominus}_{\mathrm{out}}) & \text{if } u = wi \text{ for some } i \\ ((q^{\ominus}_v)_{v\in\mathrm{Dom}(t)}, \delta^{\ominus}_{\mathrm{out}}(q^{\ominus}_{\mathrm{out}}, (\mathsf{L}, \mathfrak{m}))) & \text{otherwise, i.e. } u = \varepsilon \end{cases}
\end{aligned}
$$

Finally, $\delta^{\oplus}(((q^{\ominus}_u)_{u\in\mathrm{Dom}(t)}, q^{\oplus}))$ is given by the case $(q^{\ominus}_\alpha, q^{\oplus}_\beta)$ of Definition 6.5.

D.2 Proof of Lemma 6.8

We design a machine $\mathcal{H}$ whose runs on the input t simulate $\beta_{\mathrm{out}}(\alpha_t)$ step by step. Its states and memory values are:

$$Q_H = Q^{\oplus}_{\mathrm{out}} \cup \bigcup_{\sigma\in\Sigma} Q^{\oplus}_\sigma \times Q^{\ominus}_{\mathrm{out}} \qquad M_H = \bigcup_{\sigma\in\Sigma} Q^{\ominus}_\sigma$$

A configuration (u, q, μ) of $\mathcal{H}$ represents an active state $q^{\oplus}$ of $\beta_{\mathrm{out}}(\alpha_t)$ when:

- either $q \in Q^{\oplus}_{\mathrm{out}}$ and $q^{\oplus} = ((\mu(v))_{v\in\mathrm{Dom}(t)}, q)$,
- or $q = (q^{\oplus}_u, q^{\ominus}_{\mathrm{out}})$ and $q^{\oplus} = (u, q^{\oplus}_u, (\mu(v))_{v\neq u}, q^{\ominus}_{\mathrm{out}})$.

The initial state of $\mathcal{H}$ is $\delta^{\ominus}_{\mathrm{out}}(q^{\ominus}_{0,\mathrm{out}}, \bullet)$. Using bottom-up initialization, for each node with label σ, we take the initial memory value to be $q^{\ominus}_{0,\sigma}$. This way, the initial configuration of $\mathcal{H}$ is the unique representation of $\delta^{\ominus}(q_0^{\ominus}, \bullet)$ where $q_0^{\ominus}$ is the initial state of $\beta_{\mathrm{out}}(\alpha_t)$. (However, the representations of active states in $Q^{\oplus} \setminus (Q^{\ominus}_{\mathrm{Dom}(t)} \times Q^{\oplus}_{\mathrm{out}})$ are not unique: the memory value at the current position is undetermined.)

The argument of the transition function is some regular lookaround state, containing enough information to determine:

- the label σ of the current node;
- the current state — let us treat here the case where it has the form $(q^{\oplus}, q^{\ominus}_{\mathrm{out}})$;
- the labels $\sigma_1, \dots, \sigma_{\mathrm{rank}(\sigma)}$ and memory values $q_1^{\ominus}, \dots$ of the current node's children;
- either the label σ_{up} and the memory value $q^{\ominus}_{\mathrm{up}}$ of the parent or the knowledge that the current node has no parent i.e. is the root.

On such an input, the transition function returns the result of applying to $\delta^{\oplus}_\sigma(q^{\oplus})$ the map

$$
\begin{aligned}
p^{\oplus} &\mapsto ((p^{\oplus}, q^{\ominus}_{\mathrm{out}}), \text{ [some arbitrary value]}, \circlearrowleft) \quad \text{(note the stay-instruction)} \\
\gamma(p_1^{\oplus}, \dots, p^{\oplus}_{\mathrm{rank}(c)}) &\mapsto \gamma(((p_1^{\oplus}, q^{\ominus}_{\mathrm{out}}), \text{ [some arbitrary value]}, \circlearrowleft), \dots) \\
(p^{\ominus}, \mathtt{arg}_i \mathfrak{m}) &\mapsto ((\delta^{\ominus}_{\sigma_i}(q_i^{\ominus}, (\mathsf{R}, \dots (\mathsf{R}, \mathfrak{m}))), q^{\ominus}_{\mathrm{out}}), \ p^{\ominus}, \ i) \\
(p^{\ominus}, (\mathsf{R}, \dots (\mathsf{R}, \mathfrak{m}))) &\mapsto \begin{cases} (\delta^{\ominus}_{\mathrm{out}}(q^{\ominus}_{\mathrm{out}}, (\mathsf{L}, \mathfrak{m})), \ p^{\ominus}, \ \circlearrowleft) & \text{if the current node is the root} \\ ((\delta^{\ominus}_{\sigma_{\mathrm{up}}}(q^{\ominus}_{\mathrm{up}}, \mathtt{arg}_j \mathfrak{m}), q^{\ominus}_{\mathrm{out}}), \ p^{\ominus}, \ \uparrow) & \text{if the current node is an } j\text{-th child} \end{cases}
\end{aligned}
$$

(for a non-root node, this number j can also be determined by regular lookaround). This definition closely reflects the transitions of $\beta_{\mathrm{out}}(\alpha_t)$ described in the previous subsection of the appendix, in order to get a step-by-step simulation.

E Details and proofs for Section 7

E.1 Specification of our affine λ-calculus

Grammar of λ-terms: $t, u ::= x \mid a \mid \lambda x.\, t \mid t\, u \mid \langle t, u\rangle \mid \pi_1\, t \mid \pi_2\, t$. Reduction rules: $(\lambda x.\, T)\, U \to_\beta T[x \leftarrow U]$ and $\pi_i\, \langle T_1, T_2\rangle \to_\beta T_i$. Typing:

$$\frac{}{\Phi \mid \Theta, x : A \vdash x : A} \qquad \frac{\Phi \mid \Theta, x : A \vdash T : B}{\Phi \mid \Theta \vdash \lambda x.\, T : A \multimap B} \qquad \frac{\Phi \mid \Theta \vdash T : A \multimap B \quad \Phi \mid \Theta' \vdash U : A}{\Phi \mid \Theta, \Theta' \vdash T\, U : B}$$

$$\frac{}{\Phi,\ c : A \mid \Theta \vdash c : A} \qquad \frac{\Phi \mid \Theta \vdash T : A \quad \Phi \mid \Theta \vdash U : B}{\Phi \mid \Theta \vdash \langle T, U\rangle : A \,\&\, B} \qquad \frac{\Phi \mid \Theta \vdash T : A_1 \,\&\, A_2}{\Phi \mid \Theta \vdash \pi_i\, T : A_i}\ (i \in \{1, 2\})$$

E.2 Proof of Lemma 7.7

In order to show that β-reduction, starting from a λ-term built from an input $t \in \mathsf{T}_\Sigma$ by our λ-transducer, simulates the tree-generating machine for $\mathcal{H}$ on the same input t, we explain how relate configurations of $\mathcal{H}$ to λ-terms.

First we note that $(u, q, \mu) \in \mathrm{Conf}(\mathcal{H}, t)$ contains the same information as the data of:

- the state q;
- the subtree $t\restriction_u$ annotated with the memory values from μ — this gives us a tree $\overline{t}^{u,\mu} \in \mathsf{T}_{\langle \Sigma, M\rangle}$;
- the context $t[u \leftarrow \Box]$ above u, plus memory annotations, yielding a one-hole context $\underline{t}_{u,\mu} \in \mathsf{T}_{\langle \Sigma, M\rangle}(\{\Box\})$ with a single leaf with label $\Box$.

We explain how to represent (memory-annotated) subtrees $\overline{t}^{u,\mu}$, then contexts $\underline{t}_{u,\mu}$, then configurations.

Definition E.1 (set of λ-representations of a subtree). $\lceil - \rceil_n : \mathsf{T}_{\langle \Sigma, M\rangle} \to \{\text{sets of terms of type } A_n\}$ is defined by:

$$\lceil \langle \sigma, m\rangle(t_1, \dots, t_{\mathrm{rank}(\sigma)})\rceil_k = \begin{cases} \varnothing & \text{if } k < \omega(m) \\ \left\{ T^{\vec{n};k}_{\sigma,m}\ T_1\ \dots\ T_{\mathrm{rank}(\sigma)} \mid \vec{n} \in \mathbb{N}^{\mathrm{rank}(\sigma)},\ T_i \in \lceil t_i \rceil_{n_i} \right\} & \text{otherwise} \end{cases}$$

For contexts note that the continuation passed to a subtree behavior can also be thought of as the behavior of the surrounding context. Let $B_n = A_{n-1} \multimap o^{\&Q}$ for $n \geqslant 1$, so that $A_n = B_n \multimap o^{\&Q}$. We define $\lfloor C \rfloor_n$ as a set of terms of type B_n for a one-hole context C.

Definition E.2 (set of λ-representations of a context). $\lfloor - \rfloor_k$ maps one-hole contexts to sets of terms of type B_k for $k \geqslant 1$. It is defined by induction on the depth of the hole $\Box$: the base case is $\lfloor \Box \rfloor_k = \{\lambda x.\, \langle \gamma_0 \mid q \in Q\rangle\}$ and the inductive case is

$$\lfloor C[\Box \leftarrow \langle \sigma, m\rangle(t_1, \dots, t_i, \Box, t_{i+1}, \dots)]\rfloor = \left\{ \lambda x.\ T^{\vec{n};\ell}_{\sigma,m}\ T_1\ \dots\ x\ \dots\ T_{\mathrm{rank}(\sigma)}\ U \mid \vec{n} \in \mathbb{N}^{\mathrm{rank}(\sigma)},\ n_i = k-1,\ T_j \in \lceil t_j \rceil_{n_j},\ \ell \geq \omega(m),\ U \in \lfloor C \rfloor_\ell \right\}$$

Definition E.3 (set of λ-representations of a configuration). $[\![-]\!]$ maps configurations to sets of terms of type o:

$$[\![(u, q, \mu)]\!] = \{\pi_q\ (T\, U) \mid \exists k \geqslant 1 : T \in \lceil \overline{t}^{u,\mu} \rceil_k,\ U \in \lfloor \underline{t}_{u,\mu} \rfloor_k\}$$

We define $(\leftrightsquigarrow) \subset \{\lambda\text{-terms } T \text{ s.t. } \Gamma^{\&} \mid \varnothing \vdash T : o\} \times \mathsf{T}_\Gamma(\mathrm{Conf}(\mathcal{H}, t))$ as the relation generated from $T \leftrightsquigarrow (u, q, \mu) \iff T \in [\![(u, q, \mu)]\!]$ by the context closure $(\forall i,\ T_i \leftrightsquigarrow g_i) \implies \gamma\, \langle T_1, \dots, T_k\rangle \leftrightsquigarrow \gamma(g_1, \dots, g_k)$ (g stands for "global state").

Lemma E.4 (Simulation). *For all $(u, q, \mu) \in \mathrm{Conf}(\mathcal{H}, t)$ and $V \in [\![(u, q, \mu)]\!]$, there is $V \to_\beta^* V'$ s.t. $V' \leftrightsquigarrow$ computation-step$((u, q, \mu))$.*

Proof. Let $\sigma = \mathrm{lab}_t(u)$ and $m = \mu(u)$. By definition of $[\![-]\!]$, for some $T_1 \in \lceil \overline{t}^{u1,\mu} \rceil_{n_1}, \dots, T_{\mathrm{rank}\,\sigma} \in \lceil \overline{t}^{u\,\mathrm{rank}(\sigma),\mu} \rceil_{n_{\mathrm{rank}(\sigma)}}$ and $U \in \lfloor \underline{t}_{u,\mu} \rfloor_k$,

$$T' = \pi_q\ (T^{\vec{n};k}_{\sigma,m}\ T_1\ \dots\ T_{\mathrm{rank}(\sigma)}\ U) \to_\beta^* (\text{encoding of } \delta(q, \sigma, m))\ \underbrace{[\text{the substitution of §7.2}]}_{\text{call this } \mathfrak{s}_{\mathrm{move}}}\ \underbrace{[\vec{z} \leftarrow \vec{T},\ x \leftarrow U]}_{\text{call this } \mathfrak{s}_\beta}$$

Meanwhile, the inductive definition of $\circledcirc$ on $\mathsf{T}_\Gamma(\mathrm{Conf}(\mathcal{H}, t))$ entails that the image of (u, q, μ) by the computation-step function can be similarly described as $\delta(q, \sigma, m)[(q', m', d) \leftarrow (u, q, \mu) \circledcirc (q', m', d)\ \forall q', m', d]$. Therefore, it suffices to check that:

$$\forall q', m', d,\ \exists V'' : \quad (q', m', d)\mathfrak{s}_{\mathrm{move}}\mathfrak{s}_\beta \to_\beta^* V'' \leftrightsquigarrow \big((u, q, \mu) \circledcirc (q', m', d)\big) = (ud, q', \mu')$$

where $\mu'(u) = m'$ and $\mu'(v) = \mu(v)$ for $v \neq u$, and (q', m', d') ranges over the labels of leaves of $\delta(q, \sigma, m)$ of this form. If there is no such leaf, then the result is vacuously true. If there is at least one, then we have: $k \geqslant \omega(m) > \omega(m') \geqslant 0$. Note that the definition of $\mathfrak{s}_{\mathrm{move}}$ depends on whether $k = 0$ or $k \geqslant 1$; thus, the weight-reducing property guarantees that we are in the latter case.

We then proceed by case disjunction on d.

- First we consider the case $d \in [\mathrm{rank}\,\sigma]$, i.e. $d \neq \uparrow$.

$$\begin{aligned} (q', m', d)\mathfrak{s}_{\mathrm{move}}\mathfrak{s}_\beta &= \pi_{q'}\ \big(z_d\ \big(\lambda z_{\mathrm{new}}.\, T^{\vec{n}';k}_{\sigma,m'}\ z_1\ \dots\ z_{d-1}\ z_{\mathrm{new}}\ z_{d+1}\ \dots\ z_{\mathrm{rank}(\sigma)}\ x\big)\big)\mathfrak{s}_\beta && n'_d = n_d - 1 \\ &= \pi_{q'}\ \underbrace{\big(T_d\ \big(\lambda z_{\mathrm{new}}.\, T^{\vec{n}';k}_{\sigma,m'}\ T_1\ \dots\ T_{d-1}\ z_{\mathrm{new}}\ T_{d+1}\ \dots\ T_{\mathrm{rank}(\sigma)}\ U\big)\big)}_{\text{call this } U'} && n'_i = n_i \text{ for } i \neq d \end{aligned}$$

One can recognize the shape of a λ-representation of a one-hole context: $U' \in \lfloor t' \rfloor_{n_d}$ where

$$t' = \underline{t}_{u,\mu}[\Box \leftarrow \langle \sigma, m' \rangle(\overline{t}^{u1,\mu}, \dots, \overline{t}^{u\cdot(d-1),\mu}, \Box, \overline{t}^{u\cdot(d+1),\mu}, \dots)]$$
$$= \underline{t}_{u,\mu'}[\Box \leftarrow \langle \sigma, \mu'(u) \rangle(\overline{t}^{u1,\mu'}, \dots, \overline{t}^{u\cdot(d-1),\mu'}, \Box, \overline{t}^{u\cdot(d+1),\mu'}, \dots)] = \underline{t}_{ud,\mu'}$$

Note also that $T_d \in \lceil \overline{t}_{ud,\mu} \rceil_{n_d} = \lceil \overline{t}_{ud,\mu'} \rceil_{n_d}$. Therefore $(q', m', d)\mathfrak{s}_{\text{move}}\mathfrak{s}_\beta = \pi_{q'}\left(T_d\, U'\right) \in [\![(ud, q', \mu')]\!]$.

- In the remaining case $d = \uparrow$, we can assume that u is not the root, since we are translating a THM that realizes a total function. Therefore, writing $\sigma' = \text{lab}_t(u\uparrow)$:

$$U \in \lfloor \underline{t}_{u,\mu} \rfloor_k = \lfloor \underline{t}_{u\uparrow,\mu}[\Box \leftarrow \langle \sigma', \mu(u\uparrow) \rangle(\overline{t}^{u\uparrow 1,\mu}, \dots, \Box, \dots, \overline{t}^{u\uparrow \text{rank}(\sigma'),\mu})] \rfloor_k = \left\{ T^{\vec{n}';\ell}_{\sigma',\mu(u\uparrow)}\ T'_1\ \dots\ x\ \dots\ T'_{\text{rank}(\sigma)}\ U' \mid \dots \right\}$$

$$(q', m', \uparrow)\mathfrak{s}_{\text{move}}\mathfrak{s}_\beta = \pi_{q'}\left(x\left(T^{\vec{n};k-1}_{\sigma,m'}\,\vec{z}\right)\right)\mathfrak{s}_\beta = \pi_{q'}\left(U\left(T^{\vec{n};k-1}_{\sigma,m'}\,\vec{T}\right)\right) \to_\beta \underbrace{\pi_{q'}\left(T^{\vec{n}';\ell}_{\sigma',\mu(u\uparrow)}\ T'_1\ \dots\ \left(T^{\vec{n};k-1}_{\sigma,m'}\,\vec{T}\right)\ \dots\ T'_{\text{rank}(\sigma')}\ U'\right)}_{\text{call this } V''}$$

First, from the shape we can see that $T^{\vec{n};k-1}_{\sigma,m'}\,\vec{T} \in \lceil \overline{t}^{u,\mu'} \rceil_{k-1}$ modulo one sanity check: since $\mathcal{H}$ is weight-reducing,

$$\omega(m') < \omega(m) \leqslant k \qquad \text{therefore} \qquad \omega(m') \leqslant k-1$$

Also, if u is an i-th child, then $T'_j \in \lceil \underline{t}_{u\uparrow j,\mu} \rceil_{n'_j}$ for $j \neq i$ and $U' \in \lfloor \underline{t}_{u\uparrow,\mu} \rfloor_\ell$ and $n'_i = k-1$ and $\ell \geqslant \mu(u\uparrow) = \mu'(u\uparrow)$ by definition of $\lfloor - \rfloor$. Therefore, $V'' \in [\![(u, q', \mu')]\!]$. □

Remark E.5. We only use the fact that the memory writes that occur during upwards exits are weight-reducing. Thus ω could be replaced by a function on memory symbols that bounds the upwards exists instead of the number of visits. This is consistent with the intuitions for "n-truncated behaviors" given in the main text.

To conclude the proof of Lemma 7.7, we observe that the λ-transducer that we build maps $t \in \mathsf{T}_\Sigma$ to a λ-term

$$(\lambda z.\, \pi_{q_{init}}\,(z\,(\lambda x.\,\langle \gamma_0 \mid q \in Q\rangle)))\ \underbrace{T_t}_{\text{in } \lceil \overline{t}^{\varepsilon,\mu_{init}} \rceil_N \text{ by structural induction}} \to_\beta \pi_{q_{init}}\,(T_t\ \underbrace{(\lambda x.\,\langle \gamma_0 \mid q \in Q\rangle)}_{\text{in } \lfloor \Box \rfloor_N \text{ by definition}}) \in [\![(\varepsilon, q_{init}, \mu_{init})]\!] = [\![C_{init}(t)]\!]$$

F Proofs of Section 8

F.1 Proof of Claim 8.3 and Corollary 8.2

Let $\mathcal{H}$ be a THM with $[\![\mathcal{H}]\!] : \mathsf{T}_\Gamma \to \mathsf{T}_\Sigma$ and $\mathcal{R}$ a bottom-up relabeler with $[\![\mathcal{R}]\!] : \mathsf{T}_\Sigma \to \mathsf{T}_\Xi$.

We first give a justification of Claim 8.3: the new machine for $g_{\mathcal{H}}$ uses bottom-up initialization to copy in memory the values provided by the input, then performs a tree traversal to position its head at the position indicated by the configuration, and finally, it simulates $\mathcal{H}$.

For any position $u \in [\mathsf{R}_\Xi]^*$ and label $\xi \in \Xi$, the language $L_{u,\xi} = \{t \in \mathsf{T}_\Xi \mid u \in \text{Dom}(t) \wedge \text{lab}_t(u) = \xi\}$ is regular. A classical property of bottom-up relabelings is that they preserve regular tree languages by inverse image; and so does $g_{\mathcal{H}}$. Therefore, $g^{-1}_{\mathcal{H}}([\![\mathcal{R}]\!]^{-1}(L_{u,\xi}))$ is regular.

Let h be the maximum height of a tree in the range of the transition function of $\mathcal{H}$. There are finitely many languages $g^{-1}_{\mathcal{H}}([\![\mathcal{R}]\!]^{-1}(L_{u,\xi}))$ for u *of length at most* h and $\xi \in \Xi$. Thus, without loss of generality, we may assume that these languages are all recognized by the *same* bottom-up automaton (δ, Q), with only the set of accepting states changing. This means for every $u \in [\mathsf{R}_\Xi]^{\leqslant h}$ there exists a partial function $\psi_u : Q \rightharpoonup \Xi$ such that $\psi_u(\delta[t]) = \text{lab}_{[\![\mathcal{R}]\!](g_{\mathcal{H}}(t))}(u)$ for any tree t. Note that if t encodes a configuration of $\mathcal{H}$ reachable on the input tree s, then $g_{\mathcal{H}}(t)$ appears as a subtree of $[\![\mathcal{H}]\!](s)$ at some position w. Then the bottom-up nature of $\mathcal{R}$ entails that $\text{lab}_{[\![\mathcal{R}]\!]([\![\mathcal{H}]\!](s))}(wu) = \text{lab}_{[\![\mathcal{R}]\!](g_{\mathcal{H}}(t))}(u)$.

We can now sketch the construction of a new THM *with regular lookaround* (cf. Theorem 3.20) that computes $[\![\mathcal{R}]\!] \circ [\![\mathcal{H}]\!]$. It simulates $\mathcal{H}$ step-by-step — more than that: its states and memory values (and therefore its configurations) are exactly the same — while applying the relabeling on the fly. The lookaround automaton is (δ, Q); we assume, again w.l.o.g., that if t encodes a configuration of $\mathcal{H}$, then the lookaround state $\delta[t]$ suffices to determine the current state of this configuration, as well as the label and memory value of its current node. The new transition function maps a lookaround state to a relabeling of the image of the corresponding (state,label,memory) tuple by the transition function of $\mathcal{H}$. To determine the new label at position u, we just apply ψ_u to the lookaround state — this is justified by the reasoning in the previous paragraph.

F.2 Proof of Theorem 8.4

Given $\mathcal{H}$ and $\mathcal{N}$ two Hennie machines, we aim to construct a Hennie machine $\mathcal{G}$ that realizes the composition $\mathcal{N} \circ \mathcal{H}$. Over an input tree t, $\mathcal{G}$ simulates the execution of $\mathcal{N}$ on the intermediate tree $[\![\mathcal{H}]\!](t)$ by simulating $\mathcal{H}$ to build it implicitly in memory. During the execution of $\mathcal{G}$, each node of t holds in memory all the right hand sides of rules of $\mathcal{H}$ that it has produced. We compute that lazily, by moving forward the simulation of $\mathcal{H}$ only when $\mathcal{N}$ asks for a yet unknown part of $[\![\mathcal{H}]\!](t)$. In addition, each node of $[\![\mathcal{H}]\!](t)$ in memory of $\mathcal{G}$ is decorated with the memory of $\mathcal{N}$ used in our simulated run of $\mathcal{N}$. We remember where the reading head of $\mathcal{N}$ is in the intermediate tree $[\![\mathcal{H}]\!](t)$ by adding a pointer to our memory symbols.

Finally, to allow movement of the simulated reading head of $\mathcal{N}$ between different $[\![\mathcal{H}]\!](t)$ fragments, the root and some leaves of tree fragments are enriched with *link* symbols, which take the place of those $[\![\mathcal{H}]\!](t)$ fragments that are stored at a different input node in t. They contain the information of what is the node itself, and a precise tree in the memory forest of said node, indicated by a natural number.

There are only finitely many transitions rules in $\mathcal{H}$. Additionally, as long as forests are of bounded size (in the number of trees they contain), then there also are only finitely many link symbols to decorate trees with. Thus, if our forests have bounded size, then we need only finitely many memory symbols.

Lets us describe formally the tools we've introduced here

Definition F.1 (Ordered pointed forest of pointed trees). A pointed tree on Σ is a tree in Σ together with a node $\bullet$ in its domain. An ordered pointed forest m of pointed trees of size k, on alphabet Σ, is a finite sequence $t_1, \dots, t_k$ of pointed trees on Σ together with a natural number $1 \leq \diamond \leq k$. For $1 \leq i \leq k$, we use $\bullet_i$ to denote the distinguished node in tree t_i.

The following notations are useful when discussing pointed forests : if t is a pointed tree on alphabet Σ with memory in N, and $s \in \mathrm{Dom}(t)$, then $t[\bullet = s]$ is the tree obtained by setting the pointed node to s. Letting $\sigma \in \Sigma$, we also define $t[\bullet \leftarrow \sigma]$ the tree obtained from t by setting the label of $\bullet$ to σ. In a similar fashion, if m is an ordered pointed forest of pointed trees on Σ of size k, and $1 \leq i \leq k$, then $m[\diamond = i]$ changes the pointed tree of m, and $m[i \leftarrow t]$ sets the i-th tree to be t.

To have an easier description of the transition function of $\mathcal{N} \circ \mathcal{H}$, we allow it to take stationary transitions, that do not move the reading head, as discussed in Section 3.5.

Theorem F.2. *Let $\mathcal{H} = (Q_{\mathcal{H}}, M_{\mathcal{H}}, \Sigma, \Gamma, q^0_{\mathcal{H}}, \delta_{\mathcal{H}})$ be a* THM *which visits input nodes at most C_{fv} times, and $\mathcal{N} = (Q_{\mathcal{N}}, M_{\mathcal{N}}, \Gamma, \Xi, q^0_{\mathcal{N}}, \delta_{\mathcal{N}})$ a narrow-visit Hennie machine, visiting on each branch outputing run on a tree of* $\mathrm{Dom}([\![\mathcal{N}]\!]) \cap \mathrm{Im}([\![\mathcal{H}]\!])$ *the union of at most C_{fnv} branches. Then there exists a* THM *$\mathcal{G}$ such that $[\![\mathcal{G}]\!] = [\![\mathcal{N}]\!] \circ [\![\mathcal{H}]\!]$.*

Definition F.3 ($\mathcal{N} \circ \mathcal{H}$). Letting the maximum forest size $C_{fs} = C_{fv} \times C_{fnv}$, we define $\mathcal{G}$ as follows : Its set of states $Q_{\mathcal{G}}$ is separated into three subsets we call "regimes"

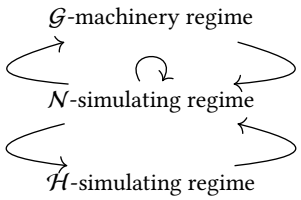
$$Q_{\mathcal{G}} = \underbrace{Q_{\mathcal{N}}}_{\mathcal{N}\text{-simulating regime}} \cup \underbrace{(Q_{\mathcal{N}} \times [C_{fs}])}_{\mathcal{G}\text{-machinery regime}} \cup \underbrace{(Q_{\mathcal{N}} \times Q_{\mathcal{H}} \times [C_{fs}] \times [C_{fs}]) \cup \{\text{qzero}\}}_{\mathcal{H}\text{-simulating regime}}$$

Its set of memory symbols is $M_{\mathcal{G}}$ the set of pointed ordered forests of pointed trees over the alphabet $\langle \Gamma, M_{\mathcal{N}} \rangle \cup \text{Link}$, with size $\leq C_{fs}$. Link are special symbols used to hold both the necessary information about the inter-fragments structure of the intermediate tree, and the memory of our simulation of $\mathcal{H}$. We explain the intuition behind their definition later while discussing the transitions $\delta_{\mathcal{G}}$ of $\mathcal{G}$, and here, we only define them as

$$\text{Link} = \underbrace{(Q_{\mathcal{H}} \times D_{R_\Sigma} \times [C_{fs}])}_{\text{DownLink , of arity 0}} \cup \underbrace{(Q_{\mathcal{H}} \times D_{R_\Sigma} \times [C_{fs}])}_{\text{UpLink , of arity 1}} \cup \underbrace{(Q_{\mathcal{H}} \times D_{R_\Sigma} \times Q_{\mathcal{H}})}_{\text{DownNewLink , of arity 0}}$$

The input and output alphabets are Σ and Ξ. The initial state is qzero, and the initial memory the empty forest.

Finally, we describe the transitions of $\mathcal{G}$, grouped by how they move between different regimes of the states of $\mathcal{G}$. The following figure is a representation of the 5 types of transitions.



A first transition is used to initialize the memory content at the root of the input tree : we let $\delta_{\mathcal{G}}(\text{qzero}, \langle \sigma, \top_{\mathcal{G}} \rangle) = \langle q_0, t^*, \circlearrowleft \rangle$ where t^* is built by taking $\delta_{\mathcal{H}}(q^0_{\mathcal{H}}, \langle \sigma, \top_{\mathcal{H}} \rangle)$, and replacing each occurrence of $\langle q_{\mathcal{H}}, m_{\mathcal{H}}, d \rangle$ with $(q_{\mathcal{H}}, m_{\mathcal{H}}, d)$.

Now let $\sigma \in \Sigma$, $q_{\mathcal{G}} \in Q_{\mathcal{G}}$, and $m_{\mathcal{G}} \in M_{\mathcal{G}}$. The right-hand side $\delta_{\mathcal{G}}(q_{\mathcal{G}}, \langle \sigma, m_{\mathcal{G}} \rangle)$ of the corresponding transition rule is defined by the following disjunction, on five types of cases :

The driving type of transitions are the $\mathcal{N} \to \mathcal{N}$ transitions, named after the second machine in our composition. They are driving in the sense that the other types of transitions are taken lazily, not any earlier than when $\mathcal{N} \to \mathcal{N}$ transitions come to need them. In the $\mathcal{N}$-simulating regime, where $\mathcal{N} \to \mathcal{N}$ transitions happen, there is a part of the intermediate tree $\mathcal{H}(t)$ loaded in memory over the node which we read, and we use it to simulate executing our second machine $\mathcal{N}$ over this fragment of the intermediate tree, updating the memory to suit the needs of simulating $\mathcal{N}$ as we go along. Here, the forest-level pointer $\diamond$ points to the fragment of the intermediate tree our simulation of $\mathcal{N}$ is currently visiting, while the tree-level pointer $\bullet_\diamond$ is used to represent the precise position of the reading head of $\mathcal{N}$ in this fragment.

Definition F.4 ($\mathcal{N} \to \mathcal{N}$ transitions). If $q_{\mathcal{G}} \in Q_{\mathcal{N}}$ and $\mathrm{lab}_{m_{\mathcal{G}}}(\bullet_\diamond) = \langle \gamma, m \rangle \in \langle \Gamma, M \rangle$, then we define the right-hand side as

$$\delta(q_{\mathcal{G}}, \langle \gamma, m \rangle)[\langle q', m', d' \rangle \leftarrow \langle q', m_{\mathcal{G}}[\bullet_\diamond \leftarrow \langle \gamma, m' \rangle]\,[\bullet_\diamond = \bullet_\diamond d'], \circlearrowleft\rangle]$$

The second and third types of transitions we define are the $\mathcal{N} \to \mathcal{G}$ and $\mathcal{G} \to \mathcal{N}$ transitions. They are named after the composed machine itself, because they do not correspond to any real step in an execution of either $\mathcal{N}$ or $\mathcal{H}$, they are only related to the internal workings of $\mathcal{G}$. They are invoked when our simulated $\mathcal{N}$ asks for a node of the intermediate tree that is not in the current fragment. When this happens, our construction guarantees they instead find an UpLink or DownLink symbol, say (n, d, i). For these transitions, we'll only need the (d, i) information. This tuple represents both : a direction, and the position of a tree in the forest stored over in this direction. This tells our machine $\mathcal{G}$ where in memory is stored the fragment containing the node we were looking for. By taking a $\mathcal{N} \to \mathcal{G}$ transition, the machine $\mathcal{G}$ commits this information to states, and moves on the input tree to try and finally find this node.

Definition F.5 ($\mathcal{N} \to \mathcal{G}$ transitions). If $q_{\mathcal{G}} = q_{\mathcal{N}} \in Q_{\mathcal{N}}$ and $\mathrm{lab}_{m_{\mathcal{G}}}(\bullet_\diamond) = (n, d, i)$, with $i \geq 1$, then we define the right-hand side as the root only tree $\langle (q_{\mathcal{N}}, i), m_{\mathcal{G}}, d \rangle$.

Because of the particular shape of the resulting state of $\mathcal{G}$, these transitions are always followed by the other type $\mathcal{G} \to \mathcal{N}$. These transitions consume the information i that was stored in state to place the pointers $\bullet$ and $\diamond$ at the right place, preparing to resume the simulation of the second machine $\mathcal{N}$.

Definition F.6 ($\mathcal{G} \to \mathcal{N}$ transitions). Arriving in state $(q_{\mathcal{N}}, i)$ and reading a new memory symbol $m'_{\mathcal{G}}$, we take a transition of right-hand side $\langle q_{\mathcal{N}}, m'_{\mathcal{G}}[\diamond = i]\,[\bullet_i = \bullet_i d_!], \circlearrowleft\rangle$. Direction $d_!$ is the unique direction that points to a neighbor of $\bullet_i$. It is unique because we only leave a fragment when pointing to a symbol in Link. The definition of $\mathcal{H} \to \mathcal{N}$ transitions guarantees that all symbols in Link are either, by construction of the memory, a root of arity 1, or a leaf of arity 0, so they have a unique neighbor.

Finally, we describe $\mathcal{N} \to \mathcal{H}$ and $\mathcal{H} \to \mathcal{N}$ transitions. They can be seen as a particular case of the previous type : since the memory is initially empty, we'll, at the first visit, actually have to compute the fragments of the intermediate tree that we want to work on before they can be processed. This is done "on the fly", when our simulation of $\mathcal{N}$ asks for them. That is to say, like with the previous $\mathcal{N} \to \mathcal{G}$ transitions, it is done only when we encounter a Link symbol. The difference this time is that these transitions are triggered only by reading DownNewLink symbols. They are the links that represent yet uncomputed fragments of $[\![\mathcal{H}]\!](t)$. Because the fragment does not yet exist, we do not have any information about its position in the forest, but what we have instead is the necessary state q_H that is used to compute it, explaining the q_H and d components of their (q_H, m_H, d) shape. Once read and committed to state, this information is discarded by overwriting with a DownLink symbol, pointing to a fragment that will be generated in the next transition. To ensure that the future fragment will be able to link back to the current one through its root, we also commit to states the information $\diamond$ of the position of the current fragment, for use in a future UpLink symbol.

Definition F.7 ($\mathcal{N} \to \mathcal{H}$ transitions). If $q_{\mathcal{G}} = q_{\mathcal{N}} \in Q_{\mathcal{N}}$ and $\mathrm{lab}_{m_{\mathcal{G}}}(\bullet_\diamond) = (q_{\mathcal{H}}, m_{\mathcal{H}}, d)$, then we define the right-hand side as the root-only tree $\langle (q_{\mathcal{N}}, q_{\mathcal{H}}, i, \diamond), m_{\mathcal{G}}[\bullet_\diamond \leftarrow (n, d, i)], \circlearrowleft\rangle$. i is defined here as the one plus the length of the forest in memory at direction d, and is computed using regular look-around.

Like the $\mathcal{G}$-machinery regime, the $\mathcal{H}$-simulating regime is temporary, and we always go back to $\mathcal{N}$-simulating with a $\mathcal{H} \to \mathcal{N}$ transition. Here we finally make use of the m_H information stored in Link symbols : since we are guaranteed to leave an input node only with a $\mathcal{N} \to \mathcal{G}$ or $\mathcal{N} \to \mathcal{H}$ transition, we know that the pointer $\bullet_\diamond$ (if it exists, i.e. the forest is non-empty), is pointing to a Link symbol. This Link symbol contains the last written memory of $\mathcal{H}$ at this point in its current simulated branch-outputting run, and thus it will be used together with the q_H in our current state to compute the new fragment. Every reading head in the right-hand side of our rule is replaced with a DownNewLink symbol, we add an UpLink symbol at the root to link back to the fragment we moved down from, and finally we set the current visited fragment to be this newly created one.

Definition F.8 ($\mathcal{H} \to \mathcal{N}$ transitions). Arriving on node sd from node s in state $(q_{\mathcal{N}}, q_{\mathcal{H}}, i, j)$ and reading a new memory symbol $m_{\mathcal{G}}$, we take a transition of right-hand side $\langle q_{\mathcal{H}}, m_{\mathcal{G}}*, \circlearrowleft\rangle$, where $m_{\mathcal{G}}*$ is built from the forest $m_{\mathcal{G}}$ by appending to it the tree $\delta_{\mathcal{H}}(q_{\mathcal{H}}, \langle \sigma, m_{\mathcal{H}} \rangle)$, where $m_{\mathcal{H}}$ is either $\top_{\mathcal{H}}$ if $m_{\mathcal{G}}$ is empty, or $\pi_2(\mathrm{lab}_{m_{\mathcal{G}}}(\bullet_\diamond))$ if it is not. Additionally, in this new tree, we replace every symbol $\langle q'_{\mathcal{H}}, m'_{\mathcal{H}}, d' \rangle$ with $(q'_{\mathcal{H}}, m'_{\mathcal{H}}, d')$, we add a new root $(m_{\mathcal{H}}, d^{-1}, j)$ (where d^{-1} is the direction such that $sdd^{-1} = s$, and can be computed using lookaround), and finally set $\diamond = i$.

Theorem F.9. $[\![\mathcal{G}]\!] = [\![\mathcal{N}]\!] \circ [\![\mathcal{H}]\!]$

The core of the proof relies on the notion of stitched tree. The stitched tree $\mathrm{Stitch}(c)$ is built from the memory μ in a configuration c of $\mathcal{G}$ on an input tree t, and is defined as the tree obtained by plugging together all the trees fragments in memory, along pairs of DownLink, UpLink symbols produced during the same $\mathcal{G} \to \mathcal{H}, \mathcal{H} \to \mathcal{G}$ transition pair. Using the stitched tree, we decode from a configuration of $\mathcal{G}$ both a configuration of $\mathcal{N}$ on $[\![\mathcal{H}]\!](t)$, and the smallest global state in an execution of $\mathcal{H}$ on t that contains all the node visited by that configuration of $\mathcal{N}$. This explicits the link between configurations of $\mathcal{G}$ and pairs of configurations of $\mathcal{N}$ and $\mathcal{H}$.

Definition F.10 (Stitched tree). If $c = (u, q, \mu)$ is a configuration of $\mathcal{G}$ over some input t, u and u' two nodes in t, we say that a node s in $\mu(u)_i$ of label $(m_{\mathcal{H}}, d, k)$ matches the root node in $\mu(u')_j$ of label in UpLink if and only if $u' = ud$ and $j = k$. Then the stitched tree $\mathrm{Stitch}(c)$

is the tree obtained by treating all DownLink symbols as symbols of arity 1, and, taking all forests $\mu(u)$ together, adding an edge from symbols in DownLink to their matching nodes.

Looking at the two transitions types of the $\mathcal{H}$-simulating regime, we verify some nices properties of Stitch. Since a node only matches another node that is created later, there are no looping sequences of matching nodes. Since each root node is matched by the DownLink that was written at the last step, each tree fragment $\mu(u)_i$ is plugged exactly one time in Stitch(c). And finally, every DownLink symbol has a matching node (if q is not a state of the H-simulating regime).

Each node in the stitched tree comes from a unique fragment $\mu(u)_i$. Let us define the origin $\mathrm{o}(s)$ of a node $s \in \mathrm{Dom}(\mathrm{Stitch}(c))$ as the node u in the input tree such that s comes from some fragment $\mu(u)_i$. If u is a node in $\mathrm{Dom}(t)$, we define $v_s(u)$ as the memory symbol $m_{\mathcal{H}}$ in the last DownLink symbol that has origin u along the branch leading to s. It is the memory symbol that was written over u by $\mathcal{H}$ when s was produced by our simulated run of $\mathcal{H}$. We decode the global state of $\mathcal{H}$ encoded in the stitched tree by replacing all symbols $(q_{\mathcal{H}}, m_{\mathcal{H}}, d) \in$ DownNewLink at some node s with $\langle q_{\mathcal{H}}, \mathrm{o}(s)d, v_s\rangle$, then removing all remaining DownLink and UpLink symbols.

This encoding describes a partial mapping $\mathfrak{R}$ from nodes in fragments in $\mu(\mathrm{Dom}(t))$ to nodes in $[\![H]\!](t)$: we first send any node that is not a Link to its place in the stitched tree, then to the corresponding node in the encoded global state of $\mathcal{H}$, and then to $[\![H]\!](t)$ by inclusion. We decode the configuration of $\mathcal{N}$ from a configuration $(q_{\mathcal{G}}, u, \mu)$ of $\mathcal{G}$ by taking the state of $\mathcal{N}$ that is explicitely in $q_{\mathcal{G}}$, setting the reading head to $\mathfrak{R}$ of the pointed node of the pointed tree in the current memory symbol $\mu(u)$, and defining the memory over a node $s \in \mathrm{Dom}([\![\mathcal{H}]\!](t))$ to be the memory stored at its antecedent by $\mathfrak{R}$, or $\top_{\mathcal{N}}$ if it doesn't have one.

Lemma F.11. $[\![\mathcal{G}]\!] \subseteq [\![\mathcal{N}]\!] \circ [\![\mathcal{H}]\!]$

Given a branch-outputting run of $\mathcal{G}$ on an input tree, we look at the sequence of stitched trees built from the sequence of configurations of $\mathcal{G}$ in the run. Looking at the definitions of the transitions of the $\mathcal{H}$-simulating regime, we verify that the successive global steps of $\mathcal{H}$ encoded in the stitched trees are an execution of $\mathcal{H}$ on the input tree : every pair of $\mathcal{G} \to \mathcal{H}, \mathcal{H} \to \mathcal{G}$ in the branch-outputting run of $\mathcal{G}$ corresponds to a transition of $\mathcal{H}$, as mentionned in the definition of $\mathcal{H} \to \mathcal{G}$ transitions. Then, the successive $\mathcal{N}$ configurations encoded in the sequence of stitched trees form a branch-outputting run of $\mathcal{N}$ on $[\![\mathcal{H}]\!](t)$ with the same output as our branch-outputting run of $\mathcal{G}$. Here, the $\mathcal{G} \to \mathcal{G}$ transitions correspond to transitions of $\mathcal{G}$.

For the proof of $[\![\mathcal{N}]\!] \circ [\![\mathcal{H}]\!] \subseteq [\![\mathcal{G}]\!]$, given a branch-outputting run of $\mathcal{N}$ on $[\![\mathcal{H}]\!](t)$, we inductively build a branch-outputting run of $\mathcal{G}$ that has the same output. This is possible because at each step, the new configuration of $\mathcal{G}$ can be made such that its induced stitched tree encodes the corresponding configuration of $\mathcal{N}$ in our given run. As a result, the encoded global states of $\mathcal{H}$ are only those visited by our branch-outputting run of $\mathcal{N}$, and because $\mathcal{N}$ is narrow-visit, we build a valid run of $\mathcal{G}$ that respects our memory size constraint.