On the expressivity of linear recursion schemes

Pierre Clairambault
Univ Lyon, EnsL, UCBL, CNRS, LIP, F-69342, LYON Cedex 07, France

Andrzej S. Murawski
Department of Computer Science, University of Oxford, UK

Abstract

We investigate the expressive power of higher-order recursion schemes (HORS) restricted to linear types. Two formalisms are considered: multiplicative additive HORS (MAHORS), which feature both linear function types and products, and multiplicative HORS (MHORS), based on linear function types only.

For MAHORS, we establish an equi-expressivity result with a variant of tree-stack automata. Consequently, we can show that MAHORS are strictly more expressive than first-order HORS, that they are incomparable with second-order HORS, and that the associated branch languages lie at the third level of the collapsible pushdown hierarchy.

In the multiplicative case, we show that MHORS are equivalent to a special kind of pushdown automata. It follows that any MHORS can be translated to an equivalent first-order MHORS in polynomial time. Further, we show that MHORS generate regular trees and can be translated to equivalent order-0 HORS in exponential time. Consequently, MHORS turn out to have the same expressive power as 0-HORS but they can be exponentially more concise.

Our results are obtained through a combination of techniques from game semantics, the geometry of interaction and automata theory.

2012 ACM Subject Classification Theory of computation → Program semantics

Keywords and phrases higher-order recursion schemes, linear logic, game semantics, geometry of interaction

Digital Object Identifier 10.4230/LIPIcs.MFCS.2019.43

Funding *Pierre Clairambault*: Supported by ANR RAPIDO (ANR-14-CE25-0007) and Labex MiLyon (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007), operated by the French National Research Agency (ANR).
Andrzej S. Murawski: Supported by a Royal Society Leverhulme Trust Senior Research Fellowship (LT170023).

Acknowledgements We would like to thank Sylvain Salvati for consultations on formal languages.

1 Introduction

Higher-order recursion schemes (HORS) have recently emerged as a promising technique for model-checking higher-order programs [17]. Linear higher-order recursion schemes (LHORS) were introduced in [5] to facilitate a finer analysis of HORS by mixing intuitionistic and linear types. In this paper, we investigate the expressivity of their purely linear fragment.

First, we consider *multiplicative additive* HORS (MAHORS), which in addition to the linear function types ($\multimap$) feature product types ($\&$), and thus allow for sharing but not re-use. We show that MAHORS are equivalent to a tree-generating variant of tree-stack automata (TSA), originally introduced to capture multiple context-free languages in the word language setting [7]. The translation from MAHORS to TSA amounts to representing the game semantics of MAHORS in the spirit of abstract machines derived from Girard's Geometry of Interaction (GoI) [11, 6]. The GoI view of computation makes it possible to interpret computation as a token machine that traverses a graph strongly related to the syntactic structure of the term. Somewhat suprisingly, so far this nearly automata-theoretic

$$\frac{}{\Gamma, x : \varphi \mid \Delta \vdash x : \varphi} \qquad \frac{}{\Gamma \mid \Delta, x : \varphi \vdash x : \varphi} \qquad \frac{\Gamma \mid \Delta \vdash t : \varphi_1 \,\&\, \varphi_2}{\Gamma \mid \Delta \vdash \pi_i\, t : \varphi_i} \qquad \frac{}{\Gamma \mid \Delta \vdash \bot : \varphi}$$

$$\frac{\Gamma \mid \Delta_1 \vdash t : \varphi \multimap \psi \qquad \Gamma \mid \Delta_2 \vdash u : \varphi}{\Gamma \mid \Delta_1, \Delta_2 \vdash t\,u : \psi} \qquad \frac{\Gamma \mid \Delta, x : \kappa \vdash t : \varphi}{\Gamma \mid \Delta \vdash \lambda x^{\kappa}.t : \kappa \multimap \varphi} \qquad \frac{\Gamma \mid \Delta \vdash t_i : \varphi_i \qquad (i \in \{1,2\})}{\Gamma \mid \Delta \vdash \langle t_1, t_2 \rangle : \varphi_1 \,\&\, \varphi_2}$$

Figure 1 Typing rules for the additive linear λ-calculus

flavour of GoI has not been exploited to establish connections with automata models, and we believe we are the first to do so explicitly. As a consequence, we can conclude that the branch languages of trees generated by MAHORS are multiple context-free and, thus, that they belong to the third level of the collapsible pushdown hierarchy [12]. In addition, we show that MAHORS are strictly more expressive than first-order HORS[1], and that they are not comparable with second-order HORS.

Secondly, we consider *multiplicative* HORS (MHORS), featuring linear function types only. In this case, our earlier MAHORS-to-TSA translation specialises to a translation into a special kind of tree-generating pushdown automata (LPDA) in which reachable configurations must be reached in a unique run. We show that MHORS and LPDA are equi-expressive and, moreover, that any MHORS can be translated to an equivalent MHORS of order 1. Further, using reachability techniques for pushdown automata, we show that LPDA are equivalent to bounded pushdown automata that forget elements stored at the bottom of the stack after the stack height exceeds a certain depth. It follows that MHORS generate regular trees, though the MHORS representation may be exponentially more succinct than order-0 HORS.

2 Linear Recursion Schemes

In this section we introduce the object of study of this paper, MAHORS and MHORS.

The main ingredient of MAHORS is the *linear λ-calculus with products* – also called the *additive linear λ-calculus*, as the product is an additive connective in the sense of Linear Logic [10]. The following definitions follow [5], restricting type formers to linear connectives (note that [5] imposes some syntactic restrictions on the shape of types and terms that we can drop here to simplify presentation, as they play no role in the technical development).

Types are formed with the ground type o and the connectives $\multimap$ and $\&$. We define the **typed terms** directly by the typing rules of Figure 1. Typing judgments have the form $\Gamma \mid \Delta \vdash t : \varphi$, where Γ and Δ are two lists of variable declarations. Intuitively, Δ is the main context containing variables that can be used at most once (such terms are often called *affine* but we opt for the name *linear* nonetheless). In contrast, Γ comprises *duplicable* variables that may be reused at will, as witnessed by the application rule. In M(A)HORS, Γ will be used only for terminal and non-terminal symbols. Linear λ-terms are equipped with standard reduction rules; we write $\rhd_\beta$ for β-reduction for functions and products, whose definition can be found *e.g.* in [5]. Any term t has a normal form, written $\mathrm{BT}(t)$.

Trees arise as ground-type terms typable in replicable contexts representing a ranked alphabet. Recall that in HORS, a symbol b of arity n is represented as a constant $\mathsf{b} : o \to$

[1] Type order is defined by $ord(o) = 0$ and $ord(\theta \to \theta') = \max(ord(\theta) + 1, ord(\theta'))$. The order of a HORS is the highest order of (the types of) its non-terminals.

$\dots \to o \to o$ with n arguments. Here, a ranked alphabet Σ may be represented in two distinct ways: *multiplicatively*, with $\mathsf{b} : o \multimap \dots \multimap o \multimap o$, or *additively*, with $\mathsf{b} : \&_n o \multimap o$, where $\&_n o$ stands for $o \,\&\, \cdots \,\&\, o$ (n copies)[2]. The choice does not impact how finite trees are represented: in both cases a $\triangleright_\beta$-normal $\Sigma \mid \_ \vdash t : o$ (if not $\bot$) must start with a variable from Σ with some arity n, followed by n $\triangleright_\beta$-normal sub-trees; *i.e.* it represents a tree (with certain branches possibly leading to $\bot$). The multiplicative vs additive distinction matters in the definition of schemes, though: with additive typing, resources (variables) may be shared when calculating two sub-branches of an infinite tree, which is disallowed with multiplicative typing.

Linear recursion schemes consist of a system of recursive equations, where each clause is given by a λ-term with a restricted shape. A term $\Gamma \mid \_ \vdash t : \varphi$ is called **applicative** if it is $\triangleright_\beta$-normal, and has the form $\lambda x_1^{\varphi_1} \dots \lambda x_n^{\varphi_n}.t'$ where t' has no abstraction.

▶ **Definition 1.** *A* ***Multiplicative Additive Recursion Scheme (MAHORS)*** *is a 4-tuple $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ where: (1) Σ is a ranked alphabet; (2) $\mathcal{N}$ is a finite set of typed* ***non-terminals****; we use upper-case letters $F, G, H, \dots$ to range over them. We denote the type of F by $\mathcal{N}(F)$ and write $F : \mathcal{N}(F)$; (3) $S \in \mathcal{N}$ is a distinguished* ***start symbol*** *of type o; and (4) $\mathcal{R}$ is a function associating to each F in $\mathcal{N}$ an applicative term $\Sigma, \mathcal{N} \mid \_ \vdash \mathcal{R}(F) : \mathcal{N}(F)$, with Σ represented additively. A* ***MHORS*** *is defined as a MAHORS where Σ is represented multiplicatively and the typing of $\mathcal{N}$ does not involve products.*

If $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ is a MAHORS, then for each $F \in \mathcal{N}$ and $n \in \mathbb{N}$ there is $\Sigma \mid \_ \vdash \mathsf{unf}_n(F) : \mathcal{N}(F)$ defined by $\mathsf{unf}_0(F) = \bot$ and $\mathsf{unf}_{n+1}(F) = \mathcal{R}(F)[\mathsf{unf}_n(G)/G \mid G \in \mathcal{N}]$. The family $(\mathsf{unf}_i(F))_{i \in \mathbb{N}}$ forms a chain for $\le$ defined as usual by $\bot \le t$, closed by congruence. As evaluation is monotone, $(\mathrm{BT}(\mathsf{unf}_i(F))_{i \in \mathbb{N}}$ also forms a chain, hence it has a lub which may be defined as the ideal completion of finite normal terms $\Sigma \mid \_ \vdash t : o$ ordered by $\le$. We may then define $\mathrm{BT}(\mathcal{G}) = \bigsqcup_{i \in \mathbb{N}} \mathrm{BT}(\mathsf{unf}_i(S))$, the **infinite tree generated by** $\mathcal{G}$.

Our schemes comprise an explicit divergence symbol $\bot$. This is unusual, but does not affect expressivity as it could always be defined with a new non-terminal with rule $\mathcal{R}(\Omega) = \Omega$. Finally, we identify silently trees and terms $\Sigma \mid \_ \vdash t : o$.

3 Finite Memory Game Semantics and Geometry of Interaction

Game semantics is a semantic technique to give a compositional interpretation of higher-order programs [14]. By presenting higher-order computation as a *game* between two players embodying the program and its execution environment (Player for the program, Opponent for the environment), it effectively reduces higher-order computation to an exchange of tokens between terms. At first forgetting recursion, we briefly review the interpretation of the linear λ-calculus with products in *simple games*, then introduce its refined interpretation as finite-memory strategies, which will inform the translation of M(A)HORS to TSA.

3.1 Games and strategies

A **game** is a tuple $A = \langle M_A, \lambda_A, P_A \rangle$ where M_A is a set of *moves*, $\lambda_A : M_A \to \{O, P\}$ is a *polarity function* (we write $M_A^O = \lambda_A^{-1}(\{O\})$ and $M_A^P = \lambda_A^{-1}(\{P\})$), and $P_A \subseteq M_A^*$ is a non-empty prefix-closed set of *valid plays*, whose elements are O-starting and alternating: if $s = s_1 \dots s_n \in P_A$, then $\lambda_A(s_1) = O$ and $\lambda_A(s_i) \ne \lambda_A(s_{i+1})$. We write $\epsilon \in P_A$ for the empty play and $s \sqsubseteq s'$ for the prefix ordering.

[2] [5] considers also intermediate typings, but this does not contribute extra expressivity.

$$
\begin{array}{ccccccc}
(o_1 & \multimap & o_2) & \multimap & o_3 & \multimap & o_4 \\
 & & & & & & \circ_4 \\
 & & \bullet_2 & & & & \\
\circ_1 & & & & & & \\
 & & & & \bullet_3 & &
\end{array}
$$

Figure 2 A play on $[\![(o \multimap o) \multimap o \multimap o]\!]$

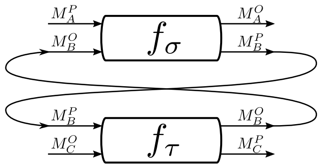

Figure 3 Composition of history-free skeletons

Games represent *types*. Plays in a game for a type φ represent executions on φ following (for this paper) a call-by-name evaluation strategy. For instance, Figure 2 shows a play in the game for $(o \multimap o) \multimap o \multimap o$, read from top to bottom. We use indices on atom occurrences and moves for disambiguation, but the usual convention in game semantics is to signify the identity of moves simply by their position under the corresponding type component. After Opponent ($\circ$, the environment) starts computation by the initial move on the right, Player ($\bullet$, the program) responds by interrogating its function argument. Opponent, playing for this argument, calls its argument. Player terminates by calling its second argument. This play is, in fact, the maximal play of the interpretation of $\lambda f^{o \multimap o}. \lambda x^o. f\, x : (o \multimap o) \multimap o \multimap o$.

Each type φ may be interpreted as a game $[\![\varphi]\!]$. The game $[\![o]\!]$ has $M_{[\![o]\!]} = \{\circ\}$ with $\lambda(o) = O$, and $P_{[\![o]\!]} = \{\epsilon, \circ\}$. To match the type constructor $\multimap$, the **linear arrow game** $A \multimap B$ has as moves the tagged disjoint union $M_{A \multimap B} = M_A + M_B = \{1\} \times M_A \cup \{2\} \times M_B$ with polarity $\lambda_{A \multimap B}(1, a) = \overline{\lambda_A(a)}$ and $\lambda_{A \multimap B}(2, b) = \lambda_B(b)$, where $\overline{O} = P$ and $\overline{P} = O$. The plays $P_{A \multimap B}$ include all O-starting, alternating sequences $s \in M^*_{A \multimap B}$ such that the *restrictions* $s \upharpoonright A \in M^*_A$ and $s \upharpoonright B \in M^*_B$, defined in the obvious way, are in P_A and P_B respectively. Hence, $A \multimap B$ can be viewed as playing the two games A and B in parallel, with the polarity reversed in A, in such a way that any play must start in B and Player is able to switch between the components. With these definitions the reader can check that $[\![(o \multimap o) \multimap o \multimap o]\!] = ([\![o]\!] \multimap [\![o]\!]) \multimap ([\![o]\!] \multimap [\![o]\!])$ includes four moves corresponding to the four atom occurrences, and has only two maximal plays: the one in Figure 2, and $\circ_4 \bullet_3$.

The **tensor game** $A \otimes B$ has moves $M_{A \otimes B} = M_A + M_B$, polarity $\lambda_{A \otimes B}(1, a) = \lambda_A(a)$ and $\lambda_{A \otimes B}(2, b) = \lambda_B(b)$, and plays are those $s \in M^*_{A \otimes B}$ that are alternating, O-starting and such that $s \upharpoonright A \in P_A$ and $s \upharpoonright B \in P_B$. Dually to $\multimap$, it follows from the definition that here only O can change between components. The **product game** $A \,\&\, B$ has the same moves and polarity as $A \otimes B$, but only the plays where either $s \upharpoonright A$ or $s \upharpoonright B$ is empty. Hence, with their first move, Opponent fixes the component in which the rest of the game will be played.

A **strategy** σ on A, written $\sigma : A$, is $\sigma \subseteq P^{ev}_A$ (writing P^{ev}_A for the set of even-length plays) which is non-empty, closed under even-length prefix, and *deterministic*, in the sense that if $sab, sab' \in \sigma$, then $b = b'$. The interpretation of terms yields strategies; for instance

$$[\![\lambda f^{o \multimap o}. \lambda x^o. f\, x : (o \multimap o) \multimap o \multimap o]\!] = \{\epsilon,\ \circ_4 \bullet_2,\ \circ_4 \bullet_2 \circ_1 \bullet_3\}$$

is a strategy on $[\![(o \multimap o) \multimap o \multimap o]\!]$ with moves following the naming convention of Figure 2.

The interpretation of terms exploits a number of constructions on strategies. In particular, to compute the **composition** of $\sigma : A \multimap B$ and $\tau : B \multimap C$ we first let σ, τ interact by considering all sequences in $(M_A + M_B + M_C)^*$ whose restrictions to A, B and B, C are respectively in σ and τ; and then project those to $P_{A \multimap C}$ to obtain $\tau \circ \sigma : A \multimap C$. We omit the details [14]. Overall, the structure needed to interpret the linear λ-calculus with products is succinctly summarized by stating that games and strategies form a *symmetric monoidal closed category with products* [14] – to any $\_ \mid x_1 : \varphi_1, \ldots, x_n : \varphi_n \vdash t : \varphi$ this lets us associate $[\![t]\!] : \bigotimes_{1 \le i \le n} [\![\varphi_i]\!] \multimap [\![\varphi]\!]$ in such a way that this is invariant under reduction – note however

$$(o_1 \multimap o_2) \multimap ((o_3 \multimap o_4) \mathbin{\&} (o_5 \multimap o_6)) \qquad (o_1 \multimap o_2) \multimap ((o_3 \multimap o_4) \mathbin{\&} (o_5 \multimap o_6))$$

$$\circ_4 \qquad\qquad \circ_6$$
$$\bullet_2 \qquad\qquad \bullet_2$$
$$\circ_1 \qquad\qquad \circ_1$$
$$\bullet_3 \qquad\qquad \bullet_5$$

Figure 4 The two maximal plays of contraction on $[\![o \multimap o]\!]$.

that in this paper, we avoid the categorical language as much as possible.

3.2 History-free and finite memory strategies

A strategy $\sigma : A$ is **history-free** if its behaviour only depends on the last move, *i.e.* there is a partial function $f : M_A^O \rightharpoonup M_A^P$ such that for all $s \in \sigma$, for all $sa \in P_A$, we have $sab \in \sigma$ iff $f(a)$ is defined and $b = f(a)$. It is key in *AJM games* [1] that, without products, terms yield history-free strategies. If $\sigma : A$ is history-free, it is characterized by the corresponding partial function $f : M_A^O \rightharpoonup M_A^P$, known as its *history-free skeleton*. For instance, the strategy $[\![\lambda f^{o \multimap o}.\, \lambda x^o.\, f\, x]\!]$ with a unique maximal play in Figure 2, has history-free skeleton $\{\circ_4 \mapsto \bullet_2,\, \circ_1 \mapsto \bullet_3\}$.

One can also directly interpret terms as history-free skeletons: this is usually referred to as *Geometry of Interaction* [11], which has close ties with game semantics [3]. In particular, composition of history-free strategies can be performed directly on skeletons. If $\sigma : A \multimap B$ and $\tau : B \multimap C$ are history-free, their history-free skeletons, which have the types

$$f_\sigma : M_A^P + M_B^O \rightharpoonup M_A^O + M_B^P \qquad\qquad f_\tau : M_B^P + M_C^O \rightharpoonup M_B^O + M_B^P,$$

may be composed via *feedback* on B, pictured in Figure 3. For any Opponent move in $A \multimap C$, we apply the corresponding function f_σ or f_τ. As long as the response is in B, we keep applying f_σ and f_τ alternately. This process may stay in B forever (a *livelock*, in which case the composition $f_{\tau\circ\sigma}$ is undefined), but otherwise we eventually get a Player move in $A \multimap C$ as required; defining a partial function $f_{\tau\circ\sigma} : M_{A\multimap C}^O \rightharpoonup M_{A\multimap C}^P$. One may visualize a token entering on the left carrying an Opponent move, then bouncing in B until it eventually exits on the right. Other constructions used in the interpretation may be presented similarly, altogether giving (for the linear λ-calculus) a presentation of evaluation through a finite automaton called a *token machine*, where a token enters through an Opponent move, and bounces through the term until it eventually exits, giving the result of computation [18].

This is our starting point to represent evaluation of M(A)HORS via an automaton. However, there is an issue: strategies for linear λ-terms *with products* are not in general history-free. For instance, Figure 4 displays the two maximal plays of a *contraction/duplication* strategy $[\![\lambda f^{o \multimap o}.\, \langle f, f\rangle : (o \multimap o) \multimap ((o \multimap o) \mathbin{\&} (o \multimap o))]\!]$. It reacts to $\circ_1$ differently depending on the history. To account for this, one may replace partial functions $f : M_A^O \rightharpoonup M_A^P$ with $f : M_A^O \times \mathcal{M} \rightharpoonup M_A^P \times \mathcal{M}$, *i.e. transducers*, where $\mathcal{M}$, the *memory*, is a finite set (see the *memoryful geometry of interaction* of [13] – however, we are not aware of this being used to define finite memory strategies). We give below a definition in this spirit, adapted to ease the translation to TSA and to deal with the branching in M(A)HORS due to *terminal symbols*.

We fix a ranked alphabet Σ (the multiplicative/additive distinction plays no role here).

▶ **Definition 2.** *A **transducer** $\mathcal{T}$ on a game A, written $\mathcal{T} : A$, is $\mathcal{T} = \langle \mathcal{M}_- \uplus \mathcal{M}_+, m_0, \delta_-, \delta_+\rangle$ where $\mathcal{M}_-$ is a finite set of **passive memory states** with a distinguished **initial memory state** $m_0 \in \mathcal{M}_-$, $\mathcal{M}_+$ is a finite set of **active memory states**, and **transition functions**:*

$$\begin{array}{lcrcl} \delta_- & : & \mathcal{M}_- \times M_A^O & \to & \mathcal{M}_+ \\ \delta_+ & : & \mathcal{M}_+ & \rightharpoonup & \mathcal{M}_+ \;+\; \mathcal{M}_- \times M_A^P \;+\; \{\mathsf{b}(m_1, \ldots, m_{|\mathsf{b}|}) \mid m_i \in \mathcal{M}_+,\, \mathsf{b} \in \Sigma\}\,. \end{array}$$

$$\delta_+^{\mathcal{T}\odot\mathcal{S}}((m_{\mathcal{S}}^-, m_{\mathcal{T}}^+)) = \begin{cases} (m_{\mathcal{S}}^-, m') & \text{if } \delta_+^{\mathcal{T}}(m_{\mathcal{T}}^+) = m' \\ \mathsf{b}((m_{\mathcal{S}}^-, m_1), \dots, (m_{\mathcal{S}}^-, m_{|\mathsf{b}|})) & \text{if } \delta_+^{\mathcal{T}}(m_{\mathcal{T}}^+) = \mathsf{b}(m_1, \dots, m_{|\mathsf{b}|}) \\ ((m_{\mathcal{S}}^-, m_{\mathcal{T}}^-), (2, c)) & \text{if } \delta_+^{\mathcal{T}}(m_{\mathcal{T}}^+) = (m_{\mathcal{T}}^-, (2, c)) \\ (\delta_-^{\mathcal{S}}(m_{\mathcal{S}}^-, (2, b)), m_{\mathcal{T}}^-) & \text{if } \delta_+^{\mathcal{T}}(m_{\mathcal{T}}^+) = (m_{\mathcal{T}}^-, (1, b)) \end{cases}$$

$$\delta_+^{\mathcal{T}\odot\mathcal{S}}((m_{\mathcal{S}}^+, m_{\mathcal{T}}^-)) = \begin{cases} (m', m_{\mathcal{T}}^-) & \text{if } \delta_+^{\mathcal{S}}(m_{\mathcal{S}}^+) = m' \\ \mathsf{b}((m_1, m_{\mathcal{T}}^-), \dots, (m_{|\mathsf{b}|}, m_{\mathcal{T}}^-)) & \text{if } \delta_+^{\mathcal{S}}(m_{\mathcal{S}}^+) = \mathsf{b}(m_1, \dots, m_{|\mathsf{b}|}) \\ ((m_{\mathcal{S}}^-, m_{\mathcal{T}}^-), (1, a)) & \text{if } \delta_+^{\mathcal{S}}(m_{\mathcal{S}}^+) = (m_{\mathcal{S}}^-, (1, a)) \\ (m_{\mathcal{S}}^-, \delta_-^{\mathcal{T}}(m_-^{\mathcal{T}}, (1, b))) & \text{if } \delta_+^{\mathcal{S}}(m_{\mathcal{S}}^+) = (m_{\mathcal{S}}^-, (2, b)) \end{cases}$$

Figure 5 Positive transitions of the composition of strategic transducers

Any transducer $\mathcal{T}$ on $[\![o]\!]$ will be called **closed**. Apart from the forced initial $\delta_-(m_0, \circ)$, it is a finite tree-generating automaton, producing a tree $\mathsf{Tree}(\mathcal{T})$. But in general transducers may play on arbitrary games. In passive states, a transducer is waiting for an Opponent move, while in active states, it is performing internal computation that may result in a terminal symbol or in a Player move and the transition to a passive state. If $\delta_+(m) = \mathsf{b}(m_1, \dots, m_{|\mathsf{b}|})$, it produces the terminal symbol b; exploring the ith child results in continuing with m_i.

Like strategies, transducers can be *composed*.

▶ **Definition 3.** *Let* $\mathcal{S} = (\mathcal{M}_-^{\mathcal{S}} \uplus \mathcal{M}_+^{\mathcal{S}}, m_0^{\mathcal{S}}, \delta_-^{\mathcal{S}}, \delta_+^{\mathcal{S}}) : A \multimap B$ *and* $\mathcal{T} = (\mathcal{M}_-^{\mathcal{T}} \uplus \mathcal{M}_+^{\mathcal{T}}, m_0^{\mathcal{T}}, \delta_-^{\mathcal{T}}, \delta_+^{\mathcal{T}}) :$ $B \multimap C$ *be transducers. The transducer* $\mathcal{T} \odot \mathcal{S}$ *on game* $A \multimap C$ *has* $\mathcal{M}_-^{\mathcal{T}\odot\mathcal{S}} = \mathcal{M}_-^{\mathcal{S}} \times \mathcal{M}_-^{\mathcal{T}}$ *and* $\mathcal{M}_+^{\mathcal{T}\odot\mathcal{S}} = \mathcal{M}_+^{\mathcal{S}} \times \mathcal{M}_-^{\mathcal{T}} \uplus \mathcal{M}_-^{\mathcal{S}} \times \mathcal{M}_+^{\mathcal{T}}$*, with initial state* $(m_0^{\mathcal{S}}, m_0^{\mathcal{T}})$*. The* transition function *is defined via* $\delta_-^{\mathcal{T}\odot\mathcal{S}}((m_{\mathcal{S}}^-, m_{\mathcal{T}}^-), (2, c)) = (m_{\mathcal{S}}^-, \delta_-^{\mathcal{T}}(m_{\mathcal{T}}^-, (2, c)))$*,* $\delta_-^{\mathcal{T}\odot\mathcal{S}}((m_{\mathcal{S}}^-, m_{\mathcal{T}}^-), (1, a)) =$ $(\delta_-^{\mathcal{S}}(m_{\mathcal{S}}^-, (1, a)), m_{\mathcal{T}}^-)$*, and positive transitions given in Figure 5.*

Besides composition, all operations on strategies used in the interpretation of the linear λ-calculus with products have a counterpart on transducers. Altogether, for any $\Sigma \mid x_1 : \varphi_1, \dots, x_n : \varphi_n \vdash t : \varphi$, this yields a transducer $\langle t \rangle : \bigotimes_{1 \le i \le n} [\![\varphi_i]\!] \multimap [\![\varphi]\!]$. In particular, if $\Sigma \mid \_ \vdash t : o$, this yields a *closed* transducer $\langle t \rangle : [\![o]\!]$. It is obtained directly by induction on syntax following denotational semantics, and in particular in polynomial time. We can prove:

▶ **Proposition 4.** *For any* $\Sigma \mid \_ \vdash t : o$*,* $\mathsf{Tree}(\langle t \rangle) = \mathrm{BT}(t)$*.*

The proof works by linking transducers with game semantics. The simple game semantics presented above cannot directly deal with the presence of non-terminals replicable at will and the associated branching, so we must first extend it to "tree-generating game semantics". The details, though rather direct, are too lengthy for the paper, so we instead present the connection ignoring the terminal symbols.

Ignoring branching transitions, transducers generate strategies. Writing $m^- \xrightarrow{a} m^+$ when $\delta_-(m^-, a) = m^+$, $m_1^+ \to m_2^+$ when $\delta_+(m_1^+) = m_2^+$ and $m^+ \xrightarrow{b} m^-$ when $\delta_+(m^+) = (m^-, b)$; the set $\mathrm{Traces}(\mathcal{T})$ comprises all sequences $s_1 \dots s_{2n} \in M_A^*$ such that (with $m_0, \dots, m_n \in \mathcal{M}_-$)

$$m_0 \xrightarrow{s_1}\to^*\xrightarrow{s_2} m_1 \xrightarrow{s_3}\to^*\xrightarrow{s_4} m_2 \dots m_{n-1} \xrightarrow{s_{2n-1}}\to^*\xrightarrow{s_{2n}} m_n\,.$$

We say that $\mathcal{T}$ is a **strategic transducer** if for all $s \in \mathrm{Traces}(\mathcal{T}) \cap P_A$, if $sa \in P_A$ and $sab \in \mathrm{Traces}(\mathcal{T})$, then $sab \in P_A$. Then, $\mathrm{Traces}(\mathcal{T}) \cap P_A$ is a strategy written $\mathrm{Strat}(\mathcal{T})$. We say that $\sigma : A$ has **finite memory** if $\sigma = \mathrm{Strat}(\mathcal{T})$ for a strategic transducer $\mathcal{T}$. We also recover *history-free strategies* as those for which $\mathcal{M}_-$ is a singleton. For instance, the strategy in Figure 4 is generated using $\mathcal{M}_- = \{m_0, m_1\}$ and $\mathcal{M}_+ = \mathcal{M}_- \times M^O_{[\![o \multimap o]\!]}$, $\delta_-(m, a) = (m, a)$, $\delta_+(\_, \circ_4) = (m_0, \bullet_2)$, $\delta_+(\_, \circ_6) = (m_1, \bullet_2)$, $\delta_+(m_0, \circ_1) = (m_0, \bullet_3)$ and $\delta_+(m_1, \circ_1) = (m_1, \bullet_5)$.

Proposition 4 boils down to the fact that all constructions on transducers in the interpretation preserve strategic transducers, and match operations on strategies – for instance, $\mathrm{Strat}(\mathcal{T} \odot \mathcal{S}) = \mathrm{Strat}(\mathcal{T}) \circ \mathrm{Strat}(\mathcal{S})$. This entails that for all t, $\mathrm{Strat}(\langle t \rangle) = [\![t]\!]$. But for closed transducers $\langle t \rangle$ and tree-generating game semantics, $\mathsf{Tree}(\langle t \rangle) = \mathrm{Strat}(\langle t \rangle)$. Since game semantics is invariant under reduction, $[\![t]\!] = [\![\mathrm{BT}(t)]\!] = \mathrm{BT}(t)$, and Proposition 4 follows.

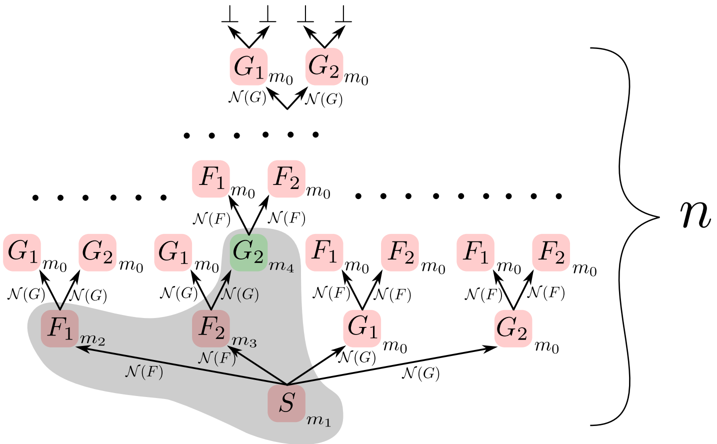

Figure 6 Illustration of a state of the n-th unfolding

4 Game Semantics to TSA

The previous section lets us associate, to any $\Sigma \mid \_ \vdash t : o$, a finite tree-generating automaton. We extend this with recursion in two steps: first we evaluate finite unfoldings using finite automata, and then we build a single automaton with additional memory (a *Tree Stack Automaton*) whose runs amount to dynamically exploring these finite unfoldings.

4.1 Unfolding recursive calls

Let us fix a M(A)HORS $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$. By definition, for each $F \in \mathcal{N}$ we have $\Sigma, \mathcal{N} \mid \_ \vdash \mathcal{R}(F) : \mathcal{N}(F)$. Let $N \in \mathbb{N}$ be such that for all $F, G \in \mathcal{N}$, G appears at most N times in $\mathcal{R}(F)$. For all $F \in \mathcal{N}$, we choose a term $\Sigma \mid \mathcal{N}_1, \dots, \mathcal{N}_N \vdash \mathcal{R}'(F) : \mathcal{N}(F)$ obtained by giving different names $G_1, \dots, G_p$ ($p \leq N$) to all occurrences of $G \in \mathcal{N}$ in $\mathcal{R}(F)$. How these names are assigned does not matter. Although $\mathcal{R}'$ differs from $\mathcal{R}$, it can be equivalently used to define the finite approximations of $\mathrm{BT}(\mathcal{G})$. For each $F \in \mathcal{N}$ and $n \in \mathbb{N}$, we redefine $\Sigma \mid \_ \vdash \mathsf{unf}_n(F) : \mathcal{N}(F)$ by setting $\mathsf{unf}_0(F) = \bot$, and $\mathsf{unf}_{n+1}(F) = \mathcal{R}'(F)[\mathsf{unf}_n(G)/G_i \mid G \in \mathcal{N}_i, 1 \leq i \leq N]$. Although defined differently, this gives the same result as in Section 2.

But, unlike the original unfolding, this one can be replicated with strategic transducers. For each $F \in \mathcal{N}$, the interpretation of the previous section yields a strategic transducer:

$$\langle \mathcal{R}'(F) \rangle : \bigotimes_{1 \leq i \leq N} \bigotimes_{G \in \mathcal{N}} [\![\mathcal{N}(G)]\!] \multimap [\![\mathcal{N}(F)]\!] \,.$$

The unfolding above can then be replicated as follows.

▶ **Proposition 5.** *Setting $\mathcal{T}_F^0 = \bot$ with all positive transitions undefined, and $\mathcal{T}_F^{n+1} = \langle \mathcal{R}'(F) \rangle \odot (\bigotimes_{1 \leq i \leq N} \bigotimes_{G \in \mathcal{N}} \mathcal{T}_G^n) : [\![\mathcal{N}(F)]\!]$, for all $n \in \mathbb{N}$, we have $\mathsf{Tree}(\mathcal{T}_S^n) = \mathrm{BT}(\mathsf{unf}_n(S))$.*

Proof. By the substitution lemma for symmetric monoidal closed categories with products, syntactic substitution matches composition in the denotational model. It follows by induction that for all $F \in \mathcal{N}$, for all $n \in \mathbb{N}$, $\langle \mathsf{unf}_n(F) \rangle$ and $\mathcal{T}_F^n$ are transducers generating the same finite memory strategy. By Proposition 4, $\mathsf{Tree}(\mathcal{T}_S^n) = \mathsf{Tree}(\langle \mathsf{unf}_n(S) \rangle) = \mathrm{BT}(\mathsf{unf}_n(S))$. ◀

Figure 6 displays the structure of transducer compositions arriving at the finite tree automaton $\mathcal{T}_S^n$, for a M(A)HORS $\mathcal{G}$ where $\mathcal{R}(S)$ has two occurrences of F and two occurrences of G, $\mathcal{R}(F)$ has two occurrences of G, and $\mathcal{R}(G)$ has two occurrences of F. Each node stands for the matching strategic transducer (corresponding to a non-terminal), edges represent compositions. Running $\mathcal{T}_S^n$ passes control between the composed transducers, with always exactly one active after the initial transition. Figure 6 shows a possible state during a run: the grey area marks nodes that have already been explored. Outside of the grey area, the (local) transducer memory must be m_0. The green node is active, and all others passive. Following the transition function of $\langle\mathcal{R}(G)\rangle$, we may next update the local memory m_4, produce a terminal and branch, or update to a passive state and send control up or down.

4.2 Tree Stack Automata

Now we give a single automaton with infinite memory whose bounded restrictions match the approximations above. It has a *stack* to deal with recursion, such that each state of the stack corresponds to a node in Figure 6. As these nodes stand for strategic transducers, they all have a finite memory. Accordingly, the automaton maintains a *store* associating, to each previously visited stack state/node, its *local memory*, accessed or updated only when visiting that node. We think of the store as a tree: the stack alphabet denotes *directions*, and stack values denote *positions* in the tree, *i.e. nodes* in (the infinite version of) Figure 6. *Pushes* and *pops* correspond to moving *up* and *down* the tree. Such an automata model is known as a *Tree Stack Automaton (TSA)* [7] – here, we introduce *tree-generating TSA*.

▶ **Definition 6.** *A tree-generating TSA $\mathcal{A}$ is a tuple $\langle\Sigma, Q, \Gamma, \mathcal{M}, \delta, q_0, \gamma_0, m_0\rangle$ where Σ is a ranked alphabet of* terminals*, Q is a set of* states*, Γ is a finite* stack alphabet*, $\mathcal{M}$ is a finite* memory alphabet*, $q_0 \in Q$ is the* starting state*, $\gamma_0 \notin \Gamma$ is the* bottom-of-stack marker *and $m_0 \in \mathcal{M}$ is the* initial local memory*. Letting $\Gamma^\bullet = \Gamma \uplus \{\gamma_0\}$, the transition function δ has type:*

$$\delta : Q \times \mathcal{M} \times \Gamma^\bullet \rightharpoonup Q + \{\mathsf{b}(q_1, \ldots, q_{|\mathsf{b}|}) \mid q_i \in Q, \mathsf{b} \in \Sigma\} + Q \times \mathcal{M} \times (\{up_\gamma \mid \gamma \in \Gamma\} + \{down\}) .$$

Informally, the transitions operate as follows. Initially, only γ_0 is on the stack. Subsequently, given state q, local memory m, and top of the stack $\gamma \in \Gamma^\bullet$:

1. if $\delta(q, m, \gamma) = q'$, the automaton changes state to q', leaving the stack and local memory unchanged;
2. if $\delta(q, m, \gamma) = \mathsf{b}(q_1, \ldots, q_{|\mathsf{b}|})$, it outputs $\mathsf{b} \in \Sigma$ and branches – to explore the ith child ($1 \leq i \leq |b|$) it proceeds to state q_i leaving other components unchanged;
3. if $\delta(q, m, \gamma) = (q', m', up_{\gamma'})$, it updates the local memory to m', changes state to q' and pushes γ' onto the stack / moves up in direction γ' (if this is the first visit to that node, its local memory is set to m_0);
4. if $\delta(q, m, \gamma) = (q', m', down)$, it updates the local memory to m' and the state to q', and then pops / moves down (we adopt the convention that γ_0 cannot be popped so, if $\gamma = \gamma_0$ in this case, the automaton blocks).

Running a TSA $\mathcal{A}$ produces a possibly infinite tree $\mathsf{Tree}(\mathcal{A})$.

In the degenerate case where $\mathcal{M} = \{m_0\}$, tree-generating TSAs turn out to be precisely tree-generating deterministic pushdown automata (PDA): the local memory cannot store information, so only the stack remains. In general, however, it is not hard to see that TSAs are Turing-complete; fortunately we will only need TSAs satisfying a further condition called *restriction* [7]. A tree-generating TSA is k-**restricted** if every node can be accessed from below at most k times. It is **restricted** if it is k-restricted for some $k \in \mathbb{N}$.

We implement the evaluation of a MAHORS $\mathcal{G}$ with a restricted TSA $\mathcal{A}(\mathcal{G})$ with states

$$\mathcal{Q} = \big(\sum_{F\in\mathcal{N}} M^O_{\otimes_{1\leq i\leq N} \otimes_{G\in\mathcal{N}} [\![\mathcal{N}(G)]\!] \multimap [\![\mathcal{N}(F)]\!]}\big) + \big(\sum_{F\in\mathcal{N}} \mathcal{M}_+^{\langle\mathcal{R}'(F)\rangle}\big).$$

$$\begin{array}{rcll}
(\mathsf{Move}(F,a),(F,m),_) & \mapsto & \mathsf{State}(F,\delta_-^F(m,a)) & \\
(\mathsf{State}(F,m),_,_) & \mapsto & \mathsf{State}(F,m') & \text{if } \delta_+^F(m) = m' \\
(\mathsf{State}(F,m),_,_) & \mapsto & \mathsf{b}(\mathsf{State}(F,m_1),\dots,\mathsf{State}(F,m_{|\mathsf{b}|})) & \text{if } \delta_+^F(m) = \mathsf{b}(m_1,\dots,m_{|\mathsf{b}|}) \\
(\mathsf{State}(F,m),_,_) & \mapsto & (\mathsf{Move}(G,(2,a)),(F,m'),up_{(F,i)}) & \text{if } \delta_+^F(m) = (m',(1,i,G,a)) \text{ with } a \in M^O_{\mathcal{N}(G)} \\
(\mathsf{State}(G,m),_,(F,i)) & \mapsto & (\mathsf{Move}(F,(1,i,G,a)),(G,m'),down) & \text{if } \delta_+^G(m) = (m',(2,a)) \text{ with } a \in M^P_{\mathcal{N}(G)}
\end{array}$$

Figure 7 Transition function for the GoI TSA.

We use constructors Move and State to refer to elements from the left and right components of $\mathcal{Q}$ respectively. The *memory alphabet* is $\mathcal{M} = \sum_{F\in\mathcal{N}} \mathcal{M}_-^{\langle\mathcal{R}'(F)\rangle}/\equiv$, where $\equiv$ is the smallest equivalence relation with $(F,m_0) \equiv (G,m_0)$ for all $F,G \in \mathcal{N}$. We write m_0 for this equivalence class, providing the *initial memory state*. The *stack alphabet* is $\Gamma = N \times \mathcal{N}$ where N is the smallest integer such that all non-terminals have fewer than N occurrences in $\mathcal{R}(F)$, for all $F \in \mathcal{N}$. The start state is $q_0 = \mathsf{Move}(S,\circ)$ and the transition function is given in Figure 7.

The TSA $\mathcal{A}(\mathcal{G})$ is designed so that a run of stack size bounded by n simulates a run of $\mathcal{T}_S^n$. When in state $\mathsf{State}(F,m)$, the automaton is currently operating in a F node of $\mathcal{T}_S^n$ (as in Figure 6), performing internal computation following δ_+^F. If this internal computation produces a move, this move will be addressed either up or down the stack, depending of whether it is a Player move in $\mathcal{N}(F)$ (in which case we must move down), or an Opponent move in $\otimes_{1\le i\le N}\otimes_{G\in\mathcal{N}}[\![\mathcal{N}(G)]\!]$ (in which case we must move up, passing the control to a recursive call). If the state is $\mathsf{State}(G,m)$ and the top of the stack is (F,i), that means that we are currently running non-terminal G, which was called as the i-th occurrence of G in F. So the stack, together with the non-terminal symbol in the state, indicate the address of a node in Figure 6. When moving up or down the stack, we first change to a transient state $\mathsf{Move}(F,a)$ in which the automaton reads the input move using δ_-^F and resumes as above.

▶ **Theorem 7.** *For any MAHORS $\mathcal{G}$, there exists a restricted TSA $\mathcal{A}(\mathcal{G})$ (constructed in polynomial time) such that* $\mathsf{Tree}(\mathcal{A}(\mathcal{G})) = \mathrm{BT}(\mathcal{G})$.

Proof. For $n \ge 1$, write $\mathsf{Tree}_n(\mathcal{A}(\mathcal{G}))$ for the tree obtained from the truncated run-tree where the stack size is bounded by $n-1$ (where γ_0 has size 0). By construction, this truncated run-tree is weakly bisimilar to that of $\mathcal{T}_S^n$. In particular, $\mathsf{Tree}_n(\mathcal{A}(\mathcal{G})) = \mathsf{Tree}(\mathcal{T}_S^n) = \mathrm{BT}(\mathsf{unf}_n(S))$ by Proposition 5, so $\mathsf{Tree}(\mathcal{A}(\mathcal{G})) = \mathrm{BT}(\mathcal{G})$ by continuity.

This TSA is restricted: for any type φ, there is a bound on the length of plays in $P_{[\![\varphi]\!]}$ – in fact $M_{[\![\varphi]\!]}$ is finite, and plays in $P_{[\![\varphi]\!]}$ cannot use the same move twice. Let k be an upper bound to the maximal length of a play in $P_{\otimes_{G\in\mathcal{N}}[\![\mathcal{N}(G)]\!]}$. Then, $\mathcal{A}(\mathcal{G})$ is k-restricted. Indeed, fix a stack value $\gamma_{n+1}\gamma_n\dots\gamma_0$ with $\gamma_{n+1} = (F,i)$. Then, all transitions moving between $\gamma_{n+1}\dots\gamma_0$ and $\gamma_n\dots\gamma_0$ carry a move from $M_{\otimes_{G\in\mathcal{N}}[\![\mathcal{N}(G)]\!]}$. By construction, the sequence of such moves forms a play in $P_{\otimes_{G\in\mathcal{N}}[\![\mathcal{N}(G)]\!]}$. Hence, it is bounded by k. ◀

If the input scheme is an MHORS then each $\mathcal{R}'(F)$ is interpreted by a history-free strategy: $\mathcal{M}_-^{\langle\mathcal{R}'(F)\rangle}$ is a singleton. Consequently, $\mathcal{A}(\mathcal{G})$ has trivial memory and is in fact simply a PDA. This PDA is still k-restricted but also satisfies a stronger *linearity* property:

▶ **Lemma 8.** *Let $\mathcal{G}$ be an MHORS. Then the tree-generating PDA $\mathcal{A}(\mathcal{G})$ is* ***linear****, in the sense that the associated graph of reachable configurations is a tree.*

Proof. A strategic transducer on A is **reversible** if for each $a \in M_A^P$ there is at most one $m \in \mathcal{M}_+$ such that $\delta_+(m) = (\_,a)$ and for each $m \in \mathcal{M}_+$ there is at most one $(m',a) \in \mathcal{M}_- \times M_A^O$ such that $\delta_-(m',a) = m$ or at most one $m' \in \mathcal{M}_+$ such that $\delta(m') = m$, and the two possibilities are mutually exclusive. Reversible strategic transducers are closed under all operations used in the interpretation, hence if $\Sigma \mid \Delta \vdash t : A$ involves no product, $\langle t \rangle$ is *reversible* (this phenomenon is well-known in GoI [6]). This entails that $\mathcal{A}(\mathcal{G})$ is linear. ◀

5 TSA to MAHORS

In this section we show how to simulate a k-restricted TSA $\mathcal{A} = \langle \Sigma, Q, \Gamma, \mathcal{M}, \delta, q_0, \gamma_0, m_0 \rangle$ in MAHORS, i.e. we establish the converse of Theorem 7.

Let $B = |\Gamma|$ and $\Gamma = \{\gamma_1, \cdots, \gamma_B\}$. Nodes of the tree store will be represented using non-terminals $F^{u_1,\cdots,u_B;d}_{q,m,\gamma}$, where $(q,m,\gamma) \in Q \times \mathcal{M} \times \Gamma^\bullet$ represent the current state, node label and top of the stack respectively, d $(1 \le d \le k+1)$ is the number of times the node has already been visited from below and each u_j $(0 \le u_j \le k,\ 1 \le j \le B)$ is the number of times that the jth child has been visited from below. For brevity, we will write $\vec{u}$ instead of $u_1, \cdots, u_B$.

For $1 \le d \le k$, $F^{\vec{u};d}_{q,m,\gamma}$ has $B+1$ arguments: the first B arguments are used to simulate up_{γ_j} $(1 \le j \le B)$ and the last one corresponds to *down*. Each of the arguments is a Q-indexed tuple of continuations, so that projection can be used to select the right component to model the associated state change. When moving up the tree (up_{γ_j}), we call the jth argument passing as an argument another continuation that makes it possible to return (move down) later. Dually, when moving down the tree, we call the last argument passing as an argument a continuation that represents a further visit up. Using these ideas, one could code unrestricted TSA in an untyped setting, but we shall rely on carefully crafted types that allow, for each node, for up to k visits from below. In particular, if the automaton is moving down having visited a node k times from below, the corresponding upwards continuation for the $k+1$ visit is of type o, i.e. it is not usable for any future calls. The rules for $1 \le d \le k$ are summarised in the table below, using λ notation for brevity (for $F^{\vec{u};k+1}_{q,m,\gamma}$ we set $F^{\vec{u};k+1}_{q,m,\gamma} x_1 \cdots x_B = \bot$).

$\delta(q,m,\gamma)$	rule
q'	$F^{\vec{u};d}_{q,m,\gamma}\, x_1 \cdots x_B y = F^{\vec{u};d}_{q',m,\gamma}\, x_1 \cdots x_B y$
$\mathsf{b}(q_1, \cdots, q_{\lvert\mathsf{b}\rvert})$	$F^{\vec{u};d}_{q,m,\gamma}\, x_1 \cdots x_B y = \mathsf{b}\, \langle F^{\vec{u};d}_{q_1,m,\gamma}\, x_1 \cdots x_B y, \cdots, F^{\vec{u};d}_{q_{\lvert\mathsf{b}\rvert},m,\gamma}\, x_1 \cdots x_B y \rangle$
$(q', m', down)$	$F^{\vec{u};d}_{q,m,\gamma}\, x_1 \cdots x_B y = (\pi_{q'} y)\, \langle F^{\vec{u};d+1}_{q'',m',\gamma}\, x_1 \cdots x_B \mid q'' \in Q \rangle$
(q', m', up_{γ_j})	$F^{\vec{u};d}_{q,m,\gamma}\, x_1 \cdots x_B y = (\pi_{q'} x_j)\, \langle \lambda z^{\overline{T_j}}. F^{\vec{u}+e_j;d}_{q'',m',\gamma}\, x_1 \cdots x_{j-1} z x_{j+1} \cdots x_B y \mid q'' \in Q \rangle$

In the *down* case, note that the q'th component of y is used to model state change and that the continuation features m' instead of m to reflect the local memory update. Note also the change from d to $d+1$, which updates the count of visits from below.

In the *up* case, the q'th component of x_j is used to model state change and the direction of the upward move (γ_j). The use of the same γ on both sides captures the same position on the stack and m' is used on the rhs to simulate the local memory update. d does *not* change, because the continuation represents revisiting the node from above (rather than from below). However, once the node is revisited from above in the future, its jth child will have been visited $u_j + 1$ times from below: hence the change to u_j (we write $\vec{u} + e_j$ for $\vec{u}$ with the jth component incremented by 1). In the *up* case, we use a λ-term inside a rule to highlight the intention more clearly, this can be avoided by using an auxiliary non-terminal.

The start symbol $S : o$ has rule $S = (F^{0,\cdots,0;1}_{q_0,m_0,\gamma_0})\, N_1 \cdots N_B \langle \bot \mid q \in Q \rangle$. The divergent terms correspond to our convention that the automaton blocks when *down* is called at the root node. N_j $(1 \le j \le B)$ stands for $\langle N_{q,j} \mid q \in Q \rangle$, where $N_{q,j}$ are auxiliary non-terminals that represent nodes visited for the first time. They are subject to the rule $N_{q,j} y = F^{0,\cdots,0;1}_{q,m_0,\gamma_j} N_1 \cdots N_B y$.

The scheme depends on types of the form T_i $(0 \le i \le k)$ defined by $T_k = o$ and $T_i = \overline{(\overline{T_{i+1}} \multimap o)} \multimap o$, where $\overline{T}$ stands for $\&_{q \in Q} T$, i.e. $|Q|$ copies of T. In particular, we have $F^{\vec{u};d}_{q,m,\gamma} : \overline{T_{u_1}} \multimap \cdots \multimap \overline{T_{u_B}} \multimap T_{d-1}$ and $N_{q,j} : T_0$.

▶ **Theorem 9.** *For any restricted TSA, there exists an equivalent MAHORS (constructible in exponential time).*

In conjunction with Theorem 7, this shows that MAHORS and restricted TSA are equivalent.

6 Expressivity of MAHORS

It is easy to see that any (classic) first-order recursion scheme (1-HORS) can be viewed as a MAHORS, simply by giving the terminals types of the form $o \,\&\, \cdots \,\&\, o \multimap o$. Hence, MAHORS are at least as expressive as first-order HORS. Next, informed by results from the preceding sections, we can discuss their relationship with schemes of higher orders. Because our TSA model is a tree-generating variant of the automata from [7], which capture multiple context-free languages [21], we can immediately conclude the following.

▶ **Lemma 10.** *The branch language of a tree generated by a MAHORS is multiple context-free.*

Thanks to the Lemma, we can show that MAHORS and second-order HORS are incomparable.

▶ **Example 11.** There exists a second-order HORS, which is not equivalent to any MAHORS. For example, consider the 2-HORS given by: $S = Fb$, $Ff = a(f\$)(F(Gf))$, $Gfx = f(fx)$, where $a : o \to o \to o$, $b : o \to o$ and $\$: o$ are terminals and $F : (o \to o) \to o$ and $G : (o \to o) \to o \to o$ are non-terminals. The scheme generates an infinite tree whose finite branches correspond to the language $L = \{a^n b^{2^{n-1}} \$ \mid n \geq 1\}$. Because it is known that L is not multiple context-free [21, Lemma 3.5], it cannot be the branch language of a MAHORS by Lemma 10.

▶ **Example 12.** We give a MAHORS that is not equivalent to any second-order HORS, exploiting the fact that the language $L = \{w\#w\#w \mid w \in D\}$, where D is the Dyck language ($D = \epsilon \mid [D]D$), is not indexed [8] (see also page 2 of [16]). The MAHORS given below (using λ-syntax for brevity) has been obtained by lifting the grammar rules for D to triples of words, encoded with the type $\mathsf{T}_3 = ((o \multimap o) \multimap (o \multimap o) \multimap (o \multimap o) \multimap o) \multimap o$. Consequently, it generates a tree whose finite branches are the words of L prefixed by a segment of b's and followed by \$. The terminal $\mathsf{b} : (o \,\&\, o) \multimap o$ represents rule choice and the other terminals $([,], \# : o \multimap o,\ \$: o)$ are used to build the word. The scheme relies on the following non-terminals: $S : o$, $D : \mathsf{T}_3$, $K : (o \multimap o) \multimap (o \multimap o) \multimap (o \multimap o)$ and $I : o \multimap o$, which are subject to the following rules:

$$S = D(\lambda xyz.x(\#(y(\#(z\$))))), \qquad Kxyv = [(x(](yv))), \qquad Iv = v,$$
$$Df = \mathsf{b}\langle fIII, D(\lambda x_1y_1z_1.D(\lambda x_2y_2z_2.f(Kx_1x_2)(Ky_1y_2)(Kz_1z_2)))\rangle$$

If the scheme were equivalent to a 2-HORS, the language of its branches would be accepted by a 2-CPDA [12], i.e. it would be indexed [2]. However, indexed languages are closed under homomorphism, so L would be indexed too, because erasing b's and \$ is a homomorphism.

Lemma 10 identifies a strong restriction on branch languages of trees generated by MAHORS. Since multiple context-free languages form a subset of third-order collapsible pushdown languages [20], it is natural to ask whether every MAHORS might be equivalent to a third-order HORS. One could try to establish this, for example, by showing that, for every restricted TSA, there is an equivalent MAHORS that uses third-order types. Unfortunately, our proof of Theorem 7 uses types whose order grows linearly in the restriction parameter k. At the time of writing, we believe this necessary to capture the complexity of run-trees generated by our (infinite-)tree-generating TSA, though we are aware that similar hierarchies for (finite-)word languages and (finite-)tree languages do collapse, e.g. second-order abstract categorial grammars [19, 15]. The main difficulty that prevents us from translating TSA into MAHORS of order 3 is that there may be infinitely many (sub)runs that start from a given node, visit only nodes above it and return to the same node, and all such runs have to be captured in a single MAHORS. In contrast, for word languages, when TSA are seen as acceptors of finite words, it suffices to focus on the representation of a single run [7].

7 Multiplicative HORS (MHORS)

In this section we consider MHORS, i.e. &-free MAHORS. Recall from Lemma 8 that, for any MHORS, there exists an equivalent *linear* PDA (LPDA) $\langle\Sigma, Q, \Gamma, \delta, q_0, \gamma_0\rangle$ with transition function $\delta : Q \times \Gamma^\bullet \rightharpoonup Q + \{\mathsf{b}(q_1, \dots, q_{|\mathsf{b}|}) \mid q_i \in Q, \mathsf{b} \in \Sigma\} + Q \times (\{up_\gamma \mid \gamma \in \Gamma\} + \{down\})$ such that any reachable configuration must be reachable through a unique path. Next we prove the converse using first-order MHORS only. In combination with Lemma 8, this amounts to a polynomial-time translation from arbitrary MHORS to first-order MHORS.

In what follows, we view an LPDA as a pushdown system with a successor relation $\Rightarrow$, in order to exploit standard reachability techniques [4, 9]. We work with configurations of the form $(q, t) \in Q \times (\Gamma^\bullet)^*$. As we do not have the space to review all the necessary definitions, let us just recall that the techniques employ *multi-automata* over $\Gamma^\bullet$ to recognise sets of configurations. Multi-automata are finite-state machines with multiple initial states, one for each state of the analysed pushdown system. Let i_q be the initial state of a multi-automaton corresponding to $q \in Q$. Then a multi-automaton is said to recognise (q, t) if it accepts t once started from i_q (this corresponds to processing stack content top-down). In particular, we take advantage of the following facts.

- For any LPDA $\mathcal{A}$, there exists a multi-automaton $\mathcal{A}_{era}$, constructible in polynomial time, which captures erasable stack content, i.e. $\{(q, t) \in Q \times \Gamma^* \mid \exists q' \in Q. (q, t) \Rightarrow^* (q', \epsilon)\}$. Using terminology from [4], this corresponds to $pre^*(Q \times \{\epsilon\})$. Hence, given $\mathcal{A}$, one can calculate the relation $R_{\mathcal{A}} = \{(q, \gamma, q') \in Q \times \Gamma \times Q \mid (q, \gamma) \Rightarrow^* (q', \epsilon)\}$ in polynomial time.
- For any LPDA $\mathcal{A}$, there exists a multi-automaton $\mathcal{A}_{rea}$, constructible in polynomial time, which represents all configurations reachable from (q_0, γ_0), i.e. all $(q, t\gamma_0)$ such that $(q_0, \gamma_0) \Rightarrow^* (q, t\gamma_0)$. This corresponds to representing $post^*(\{(q_0, \gamma_0)\})$ [9].

▶ **Lemma 13.** *For any LPDA $\mathcal{A}$, there exists an equivalent MHORS (of order 1) and its construction can be carried out in polynomial time.*

Proof. The translation is similar to the PDA-to-1-HORS translation in [12] except that reachability analysis ($R_{\mathcal{A}}$) is used to identify places where variables actually get used. This is needed to produce a term that is linearly typable. ◀

Consequently, LPDA and MHORS are equivalent. We end this section by showing they generate regular trees. Our first lemma states that, if the stack of an LPDA grows sufficiently, there is a point after which elements lying below a certain level will no longer be accessible.

▶ **Lemma 14.** *Let $s \in \Gamma^*$. There exists a bound $H_s \geq 0$ such that, for any $t \in \Gamma^*$, if $(q, ts\gamma_0)$ is reachable and $|t| > H_s$ then there is no q' such that $(q, t) \Rightarrow^* (q', \epsilon)$.*

Proof. Consider $X = \{(q, s\gamma_0) \mid (q_0, \gamma_0) \Rightarrow^* (q, s\gamma_0)\}$. Observe that $0 \leq |X| \leq |Q|$. Because we work with an LPDA, there can be at most $|Q|$ runs from (q_0, γ_0) to X. Let H_s be the maximum stack height occurring in these runs (take 0 if $X = \varnothing$). Suppose $(q, ts\gamma_0)$ is reachable and $|t| > H_s$. If we had $(q, t) \Rightarrow^* (q', \epsilon)$ for some q' then there would be a run $(q_0, \gamma_0) \Rightarrow^* (q, ts\gamma_0) \Rightarrow^* (q', s\gamma_0)$ in which the stack height exceeds H_s (because it visits $(q, ts\gamma_0)$). This contradicts the choice of H_s. ◀

The above bound depends on s. We show that there is a uniform bound, polynomial with respect to the size of $\mathcal{A}$. First, given $s \in \Gamma^*$, the multi-automaton $\mathcal{A}_{rea}$ discussed earlier can be modified to represent $\{(q,t) \mid (q_0, \gamma_0) \Rightarrow^* (q, ts\gamma_0)\}$ simply by changing the accepting states of $\mathcal{A}_{rea}$ (to those from which an original accepting state is reachable via an $s\gamma_0$-labelled path). Let $\mathcal{A}^s_{rea}$ be the resultant automaton. Note that the size of $\mathcal{A}^s_{rea}$ is bounded by a polynomial in $|\mathcal{A}|$ that is independent of s, because the only difference between $\mathcal{A}^s_{rea}$ and $\mathcal{A}_{rea}$ is the set of accepting states, and its size bounded by $|Q|$.

Observe that $\{(q,t) \mid (q_0,\gamma_0) \Rightarrow^* (q, ts\gamma_0),\ (q,t) \Rightarrow^* (q', \epsilon) \text{ for some } q'\}$ is exactly the set of configurations that are represented by both $\mathcal{A}^s_{rea}$ and $\mathcal{A}_{era}$. Consider the product $\mathcal{A}'$ of the two multi-automata. By Lemma 14, $\mathcal{A}'$ cannot have reachable loops. Consequently, the longest word that it accepts from any initial state is bounded by the number of states of the automaton, which is polynomial in $|\mathcal{A}|$. As this reasoning is independent of s, we obtain:

▶ **Lemma 15.** *For any LPDA $\mathcal{A}$, there exists a bound H, polynomial in $|\mathcal{A}|$, such that, for any $s,t \in \Gamma^*$, if $(q, ts\gamma_0)$ is reachable and $|t| > H$ then there is no q' such that $(q,t) \Rightarrow^* (q',\epsilon)$.*

This implies that an LPDA can only use H top elements from its stack, i.e. its stack can be simulated by a finite state automaton, which is exponentially bigger. Because any 0-HORS is also an MHORS, MHORS and 0-HORS are equivalent, i.e. they generate exactly the regular trees. However, it is worth noting that MHORS may be more succinct.

▶ **Example 16.** The MHORS built from terminals $a, b : o \multimap o$, non-terminals $S : o, F_i : o \multimap o$ $(1 \le i \le n)$ with $S = F_n(bS)$, $F_0(x) = ax$ and $F_i(x) = F_{i-1}(F_{i-1}x)$ for $1 \le i \le n$ generates an infinite branch $(a^{2^n}b)^\omega$, which could only be generated by a 0-HORS of exponential size in n.

References

1 Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *Inf. Comput.*, 163(2):409–470, 2000. URL: `https://doi.org/10.1006/inco.2000.2930`, `doi:10.1006/inco.2000.2930`.

2 Klaus Aehlig, Jolie G. de Miranda, and C.-H. Luke Ong. Safety is not a restriction at level 2 for string languages. In *Proceedings of FOSSACS*, volume 3441 of *Lecture Notes in Computer Science*, pages 490–504. Springer, 2005.

3 Patrick Baillot. *Approches dynamiques en sémantique de la logique linéaire: jeux et géométrie de l'interaction*. PhD thesis, Aix-Marseille 2, 1999.

4 Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proceedings of CONCUR*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.

5 Pierre Clairambault, Charles Grellois, and Andrzej S. Murawski. Linearity in higher-order recursion schemes. *PACMPL*, 2(POPL):39:1–39:29, 2018.

6 Vincent Danos and Laurent Regnier. Reversible, irreversible and optimal lambda-machines. *Theor. Comput. Sci.*, 227(1-2):79–97, 1999.

7 Tobias Denkinger. An automata characterisation for multiple context-free languages. In *Proceedings of DLT*, volume 9840 of *Lecture Notes in Computer Science*, pages 138–150. Springer, 2016.

8 Joost Engelfriet and Sven Skyum. Copying theorems. *Inf. Process. Lett.*, 4(6):157–161, 1976.

9 Javier Esparza, David Hansel, Peter Rossmanith, and Stefan Schwoon. Efficient algorithms for model checking pushdown systems. In *Proceedings of CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 232–247. Springer, 2000.

10 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

11 Jean-Yves Girard. Geometry of Interaction 1: Interpretation of System F. *Studies in Logic and the Foundations of Mathematics*, 127:221–260, 1989.

12 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. *ACM Trans. Comput. Log.*, 18(3):25:1–25:42, 2017.
13 Naohiko Hoshino, Koko Muroya, and Ichiro Hasuo. Memoryful geometry of interaction: from coalgebraic components to algebraic effects. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 52:1–52:10, 2014. URL: `https://doi.org/10.1145/2603088.2603124`, `doi:10.1145/2603088.2603124`.
14 Martin Hyland. Game semantics. *Semantics and logics of computation*, 14:131, 1997.
15 Makoto Kanazawa. Second-order abstract categorial grammars as hyperedge replacement grammars. *Journal of Logic, Language and Information*, 19(2):137–161, 2010.
16 Makoto Kanazawa and Sylvain Salvati. The copying power of well-nested multiple context-free grammars. In *Proceedings of LATA*, volume 6031 of *Lecture Notes in Computer Science*, pages 344–355. Springer, 2010.
17 Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proceedings of POPL*, pages 416–428, 2009.
18 Olivier Laurent. A token machine for full Geometry of Interaction. In *Proceedings of TLCA*, pages 283–297, 2001. URL: `https://doi.org/10.1007/3-540-45413-6_23`, `doi:10.1007/3-540-45413-6\_23`.
19 Sylvain Salvati. Encoding second order string acg with deterministic tree walking transducers. In *Proceedings of FG*, pages 143–156. CSLI Publications, 2006.
20 Sylvain Salvati. MIX is a 2-MCFL and the word problem in $\mathbb{Z}^2$ is captured by the IO and the OI hierarchies. *J. Comput. Syst. Sci.*, 81(7):1252–1277, 2015.
21 Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On multiple context-free grammars. *Theor. Comput. Sci.*, 88(2):191–229, 1991.

A Additional material (Section 2)

[5] considers intermediate (between multiplicative and additive) typings for terminals, which have the form $\mathsf{b} : \&_{n_1} o \multimap \cdots \multimap \&_{n_k} o \multimap o$. This does not contribute any extra expressivity with respect to the additive typing $\mathsf{b} : \&_{\sum_{i=1}^k n_i} o \multimap o$.

This is because $\mathsf{b} : \&_{n_1} o \multimap \cdots \multimap \&_{n_k} o \multimap o$ can be replaced with a new non-terminal $N_\mathsf{b} : \&_{n_1} o \multimap \cdots \multimap \&_{n_k} o \multimap o$ and the rule

$$N_\mathsf{b} x_1 \cdots x_k = \mathsf{b}\langle \pi_1 x_1, \cdots, \pi_{n_1} x_1, \cdots, \pi_1 x_k, \cdots \pi_{n_k} x_k \rangle$$

where $\mathsf{b} : \&_{\sum_{i=1}^k n_i} o \multimap o$.

B Additional material (Section 3)

In this section we give the details for the TSA evaluation of MAHORS given in Section 3. In this appendix, we assume some basic familiarity with game semantics and categorical logic.

B.1 Tree-generating game semantics

We build on the well-known symmetric monoidal closed category Gam of *simple games*. We do not give a detailed construction for this category, but it can be found for instance in [14]. We shall first generalize it to deal with strategies that may produce, and branch, on non-terminals. Let us fix a tree signature Σ. The construction does not depend on whether terminals are typed additively or multiplicatively (but we will come back to that later).

▶ **Definition 17.** *We define a game* (M, P, λ) *with* $M = \Sigma \uplus \mathbb{N}$*,* $\lambda(\mathsf{b}) = -$ *for* $\mathsf{b} \in \Sigma$*,* $\lambda(n) = +$ *for* $n \in \mathbb{N}$*, and non-empty plays (prefixes of) those sequences*

$$\mathsf{b}_1 n_1 \dots \mathsf{b}_p n_p$$

such that for all $1 \le i \le p$*,* $n_i \le |\mathsf{b}_i|$*.*

Overloading notations we still write Σ *for this game – this should cause no confusion.*

In games coming from the interpretation of types, no move may be played several times. In contrast in Σ, there is no limitation as to how many times a certain move may be played. Because of that we get:

▶ **Lemma 18.** *There are strategies* $\mathrm{e}_\Sigma : \Sigma \multimap 1$ *and* $\mathrm{d}_\Sigma : \Sigma \multimap \Sigma \otimes \Sigma$ *making* $(\Sigma, \mathrm{e}, \mathrm{d})$ *a comonoid in* Gam*.*

Proof. First, we define e_Σ simply as the empty strategy.

If $s_1, s_2 \in P_\Sigma$, write $s_1 \bowtie s_2$ for the set of plays resulting from interleavings of s_1 and s_2. We then define

$$\mathrm{d}_\Sigma = \{s \in P_{\Sigma_1 \multimap \Sigma_2 \otimes \Sigma_3} \mid \forall t \sqsubseteq^+ s,\ t \upharpoonright \Sigma_1 \in (t \upharpoonright \Sigma_2 \bowtie t \upharpoonright \Sigma_3)\}$$

where the different copies of Σ are marked for disambiguation. It is routine to check that this is a strategy and that it satisfies the comonoid laws. ◀

Because Σ is a comonoid, it follows that $\Sigma \otimes -$ is a comonad on Gam. We write Gam_Σ for its Kleisli category. Gam_Σ inherits all the categorical structure from Gam, that is to say, it is symmetric monoidal closed category with products, and as such supports the interpretation of the affine λ-calculus with products and free variables in Σ duplicable at will.

In particular, the interpretation of a terminal $\mathsf{b} : \&_{p_1} o \multimap \dots \multimap \&_{p_n} o \multimap o$ is the strategy:

$$[\![\mathsf{b}]\!] : \Sigma \otimes (\&_{\sum p_i} [\![o]\!]) \to [\![o]\!]$$

playing b on the left, and if Opponent calls the argument (i, j), so does Player. Furthermore:

$$[\![\mathsf{b}(\langle t_{1,j} \mid 1 \le j \le p_1\rangle, \dots, \langle t_{n,j} \mid 1 \le j \le p_n\rangle)]\!] = [\![\mathsf{b}]\!] \circ_\Sigma \langle t_{i,j} \mid 1 \le i \le n, 1 \le j \le p_i\rangle$$

In other words, the games model treats all terminal symbols as purely additively typed. This reflects the fact that syntactically, every MAHORS can be re-typed with all terminals typed purely additively. This however comes with a loss of information, because multiplicative branching implies that no sharing is allowed between the branches. This will become particularly relevant for the part of our development that concerns MHORS. Hence we will later add further information on strategies for MHORS to recover a posteriori the linearity information coming from multiplicative branching.

Now, strategies on $\Sigma \multimap o$ can be regarded as trees – apart from the initial move $\circ$, we have only player playing terminal symbols and Opponent playing directions allowing us to explore the tree. We keep silent the implicit conversion between distinct representations of trees as normal terms $\Sigma \mid \_ \vdash t : o$ and as strategies on $\Sigma \multimap o$; in particular the interpretation of a normal term yields directly the representation of the same tree as a strategy.

Leveraging the categorical structure we just established, we have:

▶ **Theorem 19.** *For any* $\Sigma \mid \_ \vdash t : o$*, we have*

$$[\![t]\!] = \mathrm{BT}(t)$$

leaving implicit the syntactic correspondence between trees and strategies on $\Sigma \multimap o$*.*

Proof. From the categorical laws the interpretation preserves the reduction, and then it is an immediate verifications for the normal forms. ◀

B.2 Finite memory and history-free strategies

To generate finite memory and history-free strategies, we use the strategic transducers of Definition 2. In the main text, the definition of the strategy generated by a strategic transducer ignores the terminal symbols – here, we must extend it to deal with those.

Let $\mathcal{T} = (\mathcal{M}_- \uplus \mathcal{M}_+, m_0, \delta_-, \delta_+)$ be a strategic transducer on A. Then, we define a labeled transition system whose nodes are $\mathcal{M}_- \uplus \mathcal{M}_+ \uplus \{m^{\mathsf{b}} \mid m \in \mathcal{M}_+, \mathsf{b} \in \Sigma\}$.

$$
\begin{array}{lcll}
m_- & \overset{a}{\to} & m_+ & \text{if } a \in M_A^- \text{ and } \delta_-(m_-, a) = m_+ \\
m_+ & \overset{\epsilon}{\to} & m'_+ & \text{if } \delta_+(m_+) = m'_+ \\
m_+ & \overset{b}{\to} & m_- & \text{if } b \in M_A^+ \text{ and } \delta_+(m_+) = (m_-, b) \\
m_+ & \overset{\mathsf{b}}{\to} & m_+^{\mathsf{b}} & \text{if } \delta_+(m_+) = \mathsf{b}(\dots) \\
m_+^{\mathsf{b}} & \overset{i}{\to} & m'_i & \text{if } \delta_+(m_+) = \mathsf{b}(m'_1, \dots, m'_{|\mathsf{b}|})
\end{array}
$$

We refine the definition of $\mathrm{Traces}(\mathcal{T})$ given in the main text to take account of the terminal symbols. If $s \in M^*_{\Sigma \multimap A}$, for $m, m' \in \mathcal{M}_- \uplus \mathcal{M}_+ \uplus \{m^{\mathsf{b}} \mid m \in \mathcal{M}_+, \mathsf{b} \in \Sigma\}$, we define $m \overset{s}{\Rightarrow} m'$ as the smallest relation such that $m \overset{\epsilon}{\Rightarrow} m$, and if $m \overset{s}{\Rightarrow} m'$ and $m' \overset{s'}{\to} m''$, then $m \overset{ss'}{\Rightarrow} m''$. The **traces** of $\mathcal{T}$ are those $s \in M^*_{\Sigma \multimap A}$ such that $m_0 \overset{s}{\Rightarrow} m$ for some m.

▶ **Proposition 20.** *If $\mathcal{T}$ is a strategic transducer on A, then if $s \in \mathrm{Strat}(\mathcal{T})$, if $sa \in P_{\Sigma \multimap A}$ and $sab \in \mathrm{Traces}(\mathcal{T})$, then $sab \in P_{\Sigma \multimap A}$ as well.*

Proof. Ignoring the part of the play on Σ, this is by definition of strategic transducers. To this the present definition adds subsequences $\mathsf{b}i$ in Σ, preserving correctness in $\Sigma \multimap A$. ◀

If $\mathcal{T}$ is a strategic transducer on A, then $\mathrm{Plays}(\mathcal{T})$ is a strategy on $\Sigma \multimap A$. By construction it is non-empty set of even-length plays closed by even-length prefixes. It is deterministic as a strategic transducer is deterministic (its transitions are given by functions). We say that a strategy $\sigma : \Sigma \multimap A$ has **finite memory** if it is generated by a strategic transducer.

Furthermore, we say that a strategy $\sigma : \Sigma \multimap A$ is **history-free** if it is generated by a strategic transducer with $\mathcal{M}_-$ a singleton set. At first sight, this seems different from the traditional notion of history-free strategy. Indeed, the traditional notion is too strict to deal correctly with the presence of Σ here. For instance, all symbols in Σ share the same moves for the subsequent directions, so the strategy for *e.g.* $\mathsf{b}(\mathsf{b}(\bot))$ would not be history-free in the tranditional sense as the same move (the direction 0) yields the move b the first time and no response the second time. Our revised notion of history-free treats Σ in a distinct way as it should. An history-free strategy in the present sense that does not play on Σ is exactly a history-free strategy in the traditional sense; so we regard the present notion as a generalization of history-freeness in the context of tree-generating strategies.

In particular, all plain copycat strategies are history-free. This includes all the structural isomorphism from the symmetric monoidal closed structure of Gam. However, there are of course more finite-memory strategies than history-free. In particular, we have the following.

▶ **Lemma 21.** *Let A be a game. Then, the duplication strategy:*

$$\delta_A : \Sigma \otimes A \multimap A \,\&\, A$$

has finite memory.

Proof. Note that the strategy does not play in Σ, but we include it anyway as it is a structural morphism in the Kleisli category Gam_Σ.

We have to provide a strategic transducer for δ_A. For clarity, let us label the copies of A in $A \multimap A \,\&\, A$ as $A_1 \multimap A_2 \,\&\, A_3$. In the rest of the argument, the subscripts on moves indicate in which components they rest, while their polarity annotation is that taken for the full game.

Our strategic transducer has $\mathcal{M}_- = \{m_0, m_l, m_r\}$ and $\mathcal{M}_+ = \mathcal{M}_- \times M_{\Sigma\otimes A_1 \multimap A_2 \& A_3}$. Its negative transition function is $\delta_-(m, a) = (m, a)$. Its positive transition function is

$$\begin{array}{rrcl} \delta_+ & : \quad \mathcal{M}_+ & \to & \mathcal{M}_- \times M^+_{\Sigma\otimes A_1 \multimap A_2 \& A_3} \\ & (\_, a_2^-) & \mapsto & (m_l, a_1^+) \\ & (\_, a_3^-) & \mapsto & (m_r, a_1^+) \\ & (m_l, a_1^-) & \mapsto & (m_l, a_2^+) \\ & (m_r, a_1^-) & \mapsto & (m_r, a_3^+) \end{array}$$

and undefined otherwise – note that its codomain does not use all the output possibilities for a positive transition function of a strategic transducer. It is straightforward to check that this is a strategic transducer, and that it generates δ_A. ◀

In general, the symmetric monoidal closed category with products Gam_Σ admits a sub-smcc with products of history-free strategies. To establish that, we must show that all usual operations on strategies can be replicated on strategic transducers in a compatible way.

▶ **Proposition 22.** *Let $\mathcal{S} : A \multimap B$ and $\mathcal{T} : B \multimap C$ be strategic transducers. Then, $\mathcal{T} \odot \mathcal{S}$ is a strategic transducer and*

$$\mathrm{Strat}(\mathcal{T} \odot \mathcal{S}) = \mathrm{Strat}(\mathcal{T}) \circ_\Sigma \mathrm{Strat}(\mathcal{S})$$

In particular, the composition of finite memory strategies is a finite memory strategy, and the composition of history-free strategies is history-free.

Proof. $\subseteq$. If $s \in \mathrm{Strat}(\mathcal{T} \odot \mathcal{S})$, then there is $(m_0, m_0) \overset{s}{\Rightarrow} (m_1, m_2)$ in $\mathcal{T} \odot \mathcal{S}$. Projecting these transitions on $\mathcal{S}$ and $\mathcal{T}$, we get two sequences $m_0 \overset{s_\mathcal{S}}{\Rightarrow} m_1$ and $m_0 \overset{s_\mathcal{T}}{\Rightarrow} m_2$ where $s_\mathcal{S} \in P_{\Sigma\otimes A \multimap B}$ and $s_\mathcal{T} \in P_{\Sigma \otimes B \multimap C}$. Note that the fact that these are indeed correct plays follows from Proposition 20. By construction, these plays are such that $s_\mathcal{S} \upharpoonright B = s_\mathcal{T} \upharpoonright B$. By induction on the transitions in $\mathcal{T} \odot \mathcal{S}$ for s, exploiting that the definition of $\mathcal{T} \odot \mathcal{S}$ follows the usual state diagram of interactions in simple games, this lets us build $u \in \tau \circledast (\Sigma \otimes \sigma)$ whose projections yields $s_\mathcal{S}$ and $s_\mathcal{T}$ and where moves in Σ, A, C appear in the same order in u as in s. This yields $s' \in \tau \circ (\Sigma \otimes \sigma) : \Sigma \otimes \Sigma \otimes A \multimap C$ differing only from s in that the calls to Σ from $\mathcal{S}$ and $\mathcal{T}$ appear in different components. Composing with duplication on Σ, we get $s \in \mathrm{Strat}(\mathcal{T}) \circ_\Sigma \mathrm{Strat}(\mathcal{S})$ as required.

$\supseteq$. If $s \in \mathrm{Strat}(\mathcal{T}) \circ_\Sigma \mathrm{Strat}(\mathcal{S})$, then there is a witness $u \in \mathrm{Strat}(\mathcal{T}) \circledast (\Sigma \otimes \mathrm{Strat}(\mathcal{S}))$, with projections $t_\mathcal{S} = u \upharpoonright \Sigma_1, A, B \in \mathrm{Strat}(\mathcal{S})$ and $t_\mathcal{T} = u \upharpoonright \Sigma_2, B, C \in \mathrm{Strat}(\mathcal{T})$ are respectively obtained by sequences of transitions from m_0 in $\mathcal{S}$ and $\mathcal{T}$. Following the state diagram of interactions, it is direct to synchronise these two sequences of transitions and obtain a sequence of transitions on $\mathcal{T} \odot \mathcal{S}$ generating s.

The conclusion on history-free strategies follows from $\mathcal{M}_-^{\mathcal{T}\odot\mathcal{S}} = \mathcal{M}_-^\mathcal{S} \times \mathcal{M}_-^\mathcal{T}$. ◀

Other constructions on strategies are straightforward as there is no interaction between them. In particular, if $\mathcal{S}_1 : A_1 \multimap B_1$ and $\mathcal{S}_2 : A_2 \multimap B_2$, the tensor strategic transducer $\mathcal{S}_1 \otimes \mathcal{S}_2$ has $\mathcal{M}_-^{\mathcal{S}_1\otimes\mathcal{S}_2} = \mathcal{M}_-^{\mathcal{S}_1} \times \mathcal{M}_-^{\mathcal{S}_2}$ and $\mathcal{M}_+^{\mathcal{S}_1\otimes\mathcal{S}_2} = \mathcal{M}_-^{\mathcal{S}_1} \times \mathcal{M}_+^{\mathcal{S}_2} \uplus \mathcal{M}_+^{\mathcal{S}_1} \times \mathcal{M}_-^{\mathcal{S}_2}$ and transition functions defined componentwise in the obvious sense. It is direct to prove that $\mathcal{S}_1 \otimes \mathcal{S}_2$ is a strategic transducer and that $\mathrm{Strat}(\mathcal{S}_1 \otimes \mathcal{S}_2) = \mathrm{Strat}(\mathcal{S}_1) \otimes_\Sigma \mathrm{Strat}(\mathcal{S}_2)$. Overall, we have:

▶ **Theorem 23.** *There are two sub symmetric monoidal closed categories of* Gam_Σ*, written* $\mathsf{Gam}^{\mathsf{hf}}_\Sigma$ *and* $\mathsf{Gam}^{\mathsf{fm}}_\Sigma$*, where strategies are respectively history-free and finite memory strategies. Moreover, the latter also has products.*

In particular, this means that besides the interpretation $[\![-]\!]$ of the linear λ-calculus with products in Gam_Σ, there is also a function interpreting each linear λ-term with products $\Sigma \mid \Gamma \vdash t : A$ as a strategic transducer $(\!\langle t \rangle\!) : [\![\Gamma]\!] \multimap [\![A]\!]$ such that $\mathrm{Strat}((\!\langle t \rangle\!)) = [\![t]\!]$. In particular, if $\Sigma \mid \_ \vdash t : o$, it follows that $\mathsf{Tree}((\!\langle t \rangle\!)) = \mathrm{Strat}((\!\langle t \rangle\!)) = [\![t]\!] = \mathrm{BT}(t)$.

C Auxiliary material (Section 4.2)

Our automata are a variant of tree stack automata proposed in [7], adapted to generate deterministically a potentially infinite tree. The automata feature finite-state control and a tree-shaped memory. In tree nodes one can store and update elements of a finite memory alphabet $\mathcal{M}$. To keep track of the current position in the tree, [7] introduces an implicit stack pointer. In our definition the stack is integrated into the machine and the automaton has access to the top stack symbol. This allows us to conclude immediately that the model reduces to a pushdown automaton for $\mathcal{M} = \{\star\}$. Otherwise, the difference is immaterial for expressivity, because one could maintain the information about the top stack symbol using state and tree memory[3].

▶ **Definition 24.** *A tree-generating TSA* $\mathcal{A}$ *is a tuple* $\langle \Sigma, Q, \Gamma, \mathcal{M}, \delta, q_0, m_0, \gamma_0 \rangle$ *where* Σ *is a ranked alphabet of* non-terminals*,* Q *is a set of* states*,* Γ *is a finite* stack alphabet*,* $\mathcal{M}$ *is a finite* memory alphabet*,* δ *is the* transition function*,* $q_0 \in Q$ *is the* starting state*,* $m_0 \in \mathcal{M}$ *is the* starting memory*, and* γ_0 *is the* bottom-of-stack *symbol (corresponding to the root of the tree memory). The transition function has the following type*

$$\delta : Q \times \mathcal{M} \times \Gamma^{\bullet} \rightharpoonup Q + \{\mathsf{b}(q_1, \ldots, q_{|\mathsf{b}|}) \mid q_i \in Q,\, \mathsf{b} \in \Sigma\} + Q \times \mathcal{M} \times (\{up_\gamma \mid \gamma \in \Gamma\} + \{down\}),$$

where $\Gamma^{\bullet} = \Gamma \uplus \{\gamma_0\}$*, writing* γ_0 *for the bottom-of-stack marker.*

▶ **Definition 25.** *A TSA configuration* ξ *is a tuple* (q, p, ρ)*, where* $q \in Q$*,* $p \in \Gamma^*\gamma_0$ *and* $\rho : \Gamma^*\gamma_0 \rightharpoonup \mathcal{M} \times \mathbb{N}_{>0}$ *is such that* $\mathsf{dom}(\rho)$ *is non-empty, finite and suffix-closed, and* $p \in \mathsf{dom}(\rho)$*.*

ρ gives information about node labels and the number of times a particular node has been visited. p corresponds to the current stack, which indicates the current position in the tree. Because $\mathsf{dom}(\rho)$ is non-empty and suffix-closed, we must have $\gamma_0 \in \mathsf{dom}(\rho)$. The initial configuration is $\xi_0 = (q_0, \gamma_0, \{\gamma_0 \mapsto (m_0, 1)\})$. The run-tree of $\mathcal{A}$ is the tree rooted at ξ_0 obtained by following the rules given below, where $\xi \to \mathsf{b}(\xi_1, \cdots, \xi_{|\mathsf{b}|})$ denotes branching and $\xi \to \xi'$ a silent transition. Each rule is predicated on $\pi_1(\rho(p)) = m$.

- $\delta(q, m, \gamma) = q'$

$$(q, \gamma p, \rho) \to (q', \gamma p, \rho)$$

- $\delta(q, m, \gamma) = \mathsf{b}(q_1, \ldots, q_{|\mathsf{b}|})$

$$(q, \gamma p, \rho) \to \mathsf{b}((q_1, \gamma p, \rho), \cdots, (q_{|\mathsf{b}|}, \gamma p, \rho))$$

[3] Because our model features a stack, we use γ to refer to the stack alphabet and $\mathcal{M}$ to refer to the memory alphabet. [7] uses Γ for the latter.

- $\delta(q, m, \gamma) = (q', m', up_{\gamma'})$

  $$(q, \gamma p, \rho) \to (q', \gamma' \gamma p, \rho')$$

  where

  $$\rho'(s) = \begin{cases} \rho(s) & s \in \mathsf{dom}(\rho),\ s \neq \gamma p, \gamma'\gamma p \\ (m', i) & s = \gamma p,\ \rho(s) = (m, i) \\ (m_0, 1) & s = \gamma'\gamma p,\ s \notin \mathsf{dom}(\rho) \\ (m'', i+1) & s = \gamma'\gamma p,\ \rho(s) = (m'', i) \end{cases}$$

- $\delta(q, m, \gamma) = (q', m', down)$

  $$(q, \gamma p, \rho) \to (q', p, \rho')$$

  where

  $$\rho'(s) = \begin{cases} \rho(s) & s \in \mathsf{dom}(\rho),\ s \neq \gamma p \\ (m', i) & s = \gamma p,\ \rho(s) = (m, i) \end{cases}$$

  This transition fires only if $\gamma \neq \gamma_0$, i.e. the automaton gets stuck if *down* is executed when the bottom-of-stack symbol is at the top of the stack.

A *run* is a prefix of any branch of the run-tree.

▶ **Definition 26.** *A TSA is k-restricted if no node is entered from below more than k times in any run. Formally, for any configuration (q, p, ρ) reachable from ξ_0, we have* $\mathsf{img}\,\rho \subseteq \mathcal{M} \times \{1, \cdots, k\}$.

D Additional material (Section 5)

This is a formal account of the translation. Consider a k-restricted TSA $\mathcal{A} = \langle \Sigma, Q, \Gamma, \mathcal{M}, \delta, q_0, \gamma_0, m_0 \rangle$ with

$$\delta : Q \times \mathcal{M} \times \Gamma^\bullet \rightharpoonup Q + \{\mathsf{b}(q_1, \ldots, q_{|\mathsf{b}|}) \mid q_i \in Q,\ \mathsf{b} \in \Sigma\} + Q \times \mathcal{M} \times (\{up_\gamma \mid \gamma \in \Gamma\} + \{down\}).$$

Let $B = |\Gamma|$ and $\Gamma = \{\gamma_1, \cdots, \gamma_B\}$. Given a type T, let us write $\overline{T}$ for $\&_{q \in Q} T$, i.e. $|Q|$ copies of T. The construction of a MAHORS that is equivalent to the given TSA depends on types T_i $(0 \leq i \leq k)$ defined by

$$T_i = \begin{cases} o & i = k \\ \overline{(\overline{T_{i+1}} \multimap o) \multimap o} & 0 \leq i < k \end{cases}$$

The corresponding MAHORS $\mathcal{G}_\mathcal{A} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ is defined as follows.

- Terminals: Σ
- Nonterminals $\mathcal{N}$:

  $$\begin{array}{rcl} S & : & o \\ N_{q,j} & : & T_0 \\ F^{\vec{u};d}_{q,m,\gamma} & : & \overline{T_{u_1}} \multimap \cdots \multimap \overline{T_{u_B}} \multimap T_{d-1} \\ G^{\vec{u};d;j}_{q,m,\gamma} & : & \overline{T_{u_1}} \multimap \cdots \overline{T_{u_{j-1}}} \multimap \overline{T_{u_{j+1}}} \cdots \multimap \overline{T_{u_B}} \multimap \overline{(\overline{T_d} \multimap o)} \multimap \overline{T_{u_j}} \multimap o \\ \Omega_o & : & o \\ \Omega_{\overline{T_1} \multimap o} & : & \overline{T_1} \multimap o \end{array}$$

  where $q \in Q$, $1 \leq j \leq B$, $m \in \mathcal{M}$, $\gamma \in \Gamma^\bullet$, $\vec{u} = (u_1, \cdots, u_B) \in \{0, \cdots, k\}^B$ and $d \in \{1, \cdots, k+1\}$.

- Rules $\mathcal{R}$:
  - Start

$$\begin{array}{rcl} S & = & F^{0,\cdots,0;1}_{q_0,m_0,\gamma_0}\, N_1 \cdots N_B D \\ N_{q,j} y & = & F^{0,\cdots,0;1}_{q,m_0,\gamma_j} N_1 \cdots N_B y \end{array}$$

for any $q \in Q$, $1 \leq j \leq B$, where

$$\begin{array}{rcl} N_j & \equiv & \langle N_{q,j} \,|\, q \in Q \rangle : \overline{T_0} \\ D & \equiv & \langle \Omega_{\overline{T_1} \multimap o} \,|\, q \in Q \rangle \end{array}$$

and $\equiv$ stands for syntactic equality.

  - $\delta(q, m, \gamma) = q'$ (state change)

$$F^{\vec{u};d}_{q,m,\gamma}\, x_1 \cdots x_B y = F^{\vec{u};d}_{q',m,\gamma}\, x_1 \cdots x_B y$$

where $\vec{u} \in \{0, \cdots, k\}^B$, $1 \leq d \leq k$ and $(q, m, \gamma) \in Q \times \mathcal{M} \times \Gamma^\bullet$.

  - $\delta(q, m, \gamma) = \mathsf{b}(q_1, \cdots, q_{|\mathsf{b}|})$ (branching)

$$F^{\vec{u};d}_{q,m,\gamma}\, x_1 \cdots x_B y = \mathsf{b}\, \langle F^{\vec{u};d}_{q_1,m,\gamma}\, x_1 \cdots x_B y, \cdots, F^{\vec{u};d}_{q_{|\mathsf{b}|},m,\gamma}\, x_1 \cdots x_B y \rangle.$$

where $\vec{u} \in \{0, \cdots, k\}^B$, $1 \leq d \leq k$ and $(q, m, \gamma) \in Q \times \mathcal{M} \times \Gamma^\bullet$.

  - $\delta(q, m, \gamma) = (q', m', \mathit{down})$

$$F^{\vec{u};d}_{q,m,\gamma}\, x_1 \cdots x_B y = (\pi_{q'} y)\, \langle F^{\vec{u};d+1}_{q'',m',\gamma}\, x_1 \cdots x_B \,|\, q'' \in Q \rangle.$$

where $\vec{u} \in \{0, \cdots, k\}^B$, $1 \leq d \leq k$ and $(q, m, \gamma) \in Q \times \mathcal{M} \times \Gamma^\bullet$.

  - $\delta(q, m, \gamma) = (q', m', \mathit{up}_{\gamma_j})$

$$\begin{array}{rcl} F^{\vec{u};d}_{q,m,\gamma}\, x_1 \cdots x_B y & = & (\pi_{q'} x_j)\, \langle G^{\vec{u}+e_j;d;j}_{q'',m',\gamma}\, x_1 \cdots x_{j-1} x_{j+1} \cdots x_B y \,|\, q'' \in Q \rangle, \\ G^{\vec{u};d;j}_{q,m,\gamma}\, x_1 \cdots x_{j-1} x_{j+1} \cdots x_B y z & = & F^{\vec{u};d}_{q,m,\gamma}\, x_1 \cdots x_{j-1} z x_{j+1} \cdots x_B y. \end{array}$$

where $\vec{u} \in \{0, \cdots, k\}^B$, $1 \leq d \leq k$ and $(q, m, \gamma) \in Q \times \mathcal{M} \times \Gamma^\bullet$.

  - Auxiliary rules to induce divergences:

$$\begin{array}{rcl} \Omega_o & = & \Omega_o \\ \Omega_{\overline{T_1} \multimap o} x & = & \Omega_{\overline{T_1} \multimap o} x \\ F^{\vec{u};k+1}_{q,m,\gamma} x_1 \cdots x_B & = & \Omega_o \end{array}$$

for $\vec{u} \in \{0, \cdots, k\}^B$ and $(q, m, \gamma) \in Q \times \mathcal{M} \times \Gamma^\bullet$.

In order to prove that the TSA-to-MAHORS translation is correct, it is necessary to understand how a TSA configuration $\xi = (q, p, \rho)$ can be represented using an applicative term $\mathcal{T}_\xi$. We start with an auxiliary definition.

▶ **Definition 27.** *Given $r \in \mathsf{dom}(\rho)$ such that $\rho(r) = (m_r, d_r)$, let $\vec{u}_r$ be the vector $(u_{r,1}, \cdots, u_{r,B})$ defined by*

$$u_{r,i} = \begin{cases} 0 & \gamma_i r \notin \mathsf{dom}(\rho) \\ \pi_2(\rho(\gamma_i r)) & \gamma_i r \in \mathsf{dom}(\rho) \end{cases}$$

Suppose $\rho(p) = (m, d)$. The term $\mathcal{T}_\xi$ will use the non-terminal

$$F^{\vec{u}_p;d}_{q,m,\gamma_p} : \overline{T_{u_{p,1}}} \multimap \cdots \multimap \overline{T_{u_{p,B}}} \multimap T_{d-1}$$

as its head variable, where γ_p is the topmost (leftmost) element of p.

The first B arguments of $F^{\vec{u}_p;d}_{q,m,\gamma}$ will correspond to visiting nodes above p. We call them *upwards continuations*. Note that, because these future visits are from below, the non-terminals involved carry a superscript $(d_r + 1)$ equal to the number of times the node has already been visited plus 1.

▶ **Definition 28.** *Let $r = \gamma_{j_r} r' \in \Gamma^* \gamma_0$. The term $\mathcal{U}_r$ is then defined as follows*

$$\mathcal{U}_r \equiv \begin{cases} \langle N_{q,j_r} \mid q \in Q \rangle & r \notin \mathsf{dom}(\rho) \\ \langle F^{\vec{u}_r;d_r+1}_{q,m_r,\gamma_{j_r}} \mathcal{U}_{\gamma_1 r} \cdots \mathcal{U}_{\gamma_B r} \mid q \in Q \rangle & r \in \mathsf{dom}(\rho),\ \rho(r) = (m_r, d_r) \end{cases}$$

Note that this definition refers to $\mathcal{U}_{\gamma_i r}$ when defining $\mathcal{U}_r$. This is well-founded, because $\mathsf{dom}(\rho)$ is finite. Observe that we have $\mathcal{U}_r : \overline{T_{d_r}}$ if $r \in \mathsf{dom}(\rho)$ (and $\rho(r) = (m_r, d_r)$) and $\mathcal{U}_r : \overline{T_0}$ otherwise.

Because $d \le k$, the non-terminal $F^{\vec{u}_p;d}_{q,m,\gamma}$ has a $(B+1)$th argument. It will correspond to visiting the node below p and we call it a *downwards continuation*. Note that, because the visit will be from above, the non-terminals involved will carry a superscript $(d_{r'})$ that coincides with the number of times the node has been visited so far.

▶ **Definition 29.** *Given $r \in \mathsf{dom}(\rho)$ such that $\rho(r) = (m_r, d_r)$, define the term $\mathcal{D}_r : \overline{\overline{T_{d_r}} \multimap o}$ as follows.*

$$\mathcal{D}_r \equiv \begin{cases} \langle \Omega_{\overline{T_1} \multimap o} \mid q \in Q \rangle & r = \gamma_0 \\ \langle G^{\vec{u}_{r'};d_{r'};j_r}_{q,m_{r'},\gamma_{j_{r'}}} \mathcal{U}_{\gamma_1 r'} \cdots \mathcal{U}_{\gamma_{j_r - 1} r'} \mathcal{U}_{\gamma_{j_r + 1} r'} \cdots \mathcal{U}_{\gamma_B r'} \mathcal{D}_{r'} \mid q \in Q \rangle & r = \gamma_{j_r} r',\ r' = \gamma_{j_{r'}} r'' \\ & \rho(r') = (m_{r'}, d_{r'}) \end{cases}$$

This definition is inductive and relies on the fact that $\gamma_0 \in \mathsf{dom}(\rho)$ and $\mathsf{dom}(\rho)$ is suffix-closed. Note that $u_{r',j_r} = d_r$, so in the second case the term indeed has type $\overline{\overline{T_{d_r}} \multimap o}$.

▶ **Definition 30.** *Given a configuration $\xi = (q, p, \rho)$ such that $\rho(p) = (m, d)$ and $p = \gamma_p r'$, the corresponding term $\mathcal{T}_p : o$ is defined to be*

$$\mathcal{T}_p \equiv F^{\vec{u}_p;d}_{q,m,\gamma_p} \mathcal{U}_{\gamma_1 p} \cdots \mathcal{U}_{\gamma_B p} \mathcal{D}_p.$$

The correctness of the translation can now be established by showing that the MAHORS rules simulate the configuration graph of the TSA faithfully. Let ξ_0 be the initial TSA configuration and observe that

$$\mathcal{T}_{\xi_0} \equiv F^{0,\cdots,0;1}_{q_0,m_0,\gamma_0} \mathcal{U}_{\gamma_1 \gamma_0} \cdots \mathcal{U}_{\gamma_B \gamma_0} \mathcal{D}_{\gamma_0} \equiv F^{0,\cdots,0;1}_{q_0,m_0,\gamma_0} N_1 \cdots N_B D$$

and that $S = F^{0,\cdots,0;1}_{q_0,m_0,\gamma_0} N_1 \cdots N_B D$ is a rule in the corresponding MAHORS, i.e. $S = \mathcal{T}_{\xi_0}$. This means that the "starting points" of both formalisms coincide. Next we perform a case analysis of MAHORS rules, showing that every step based on rules from $\mathcal{G}_\mathcal{A}$ corresponds to a transition in $\mathcal{A}$, and vice versa.

▶ **Definition 31.** *An applicative term M built from terminals and non-terminals of $\mathcal{G}_\mathcal{A}$ is called a F-term if M is of the form $F^{\vec{u};d}_{q,m,\gamma} \cdots$. Similarly, we define G- and N-terms.*

Suppose $M \equiv \mathcal{T}_\xi$ for some ξ and $M = M'$ (one-step) according to $\mathcal{R}$. Then M is a F-term and the following cases arise for M'.

- M' is an F-term.

  This is the case if the rule for (state change) was used to derive $M = M'$. Because $M \equiv \mathcal{T}_\xi$, we have $M' \equiv \mathcal{T}_{\xi'}$, where $\xi \to \xi'$ (via $\delta(q, m, \gamma) = q'$).
- $M' \equiv \mathsf{b}\langle M_1, \cdots, M_B\rangle$

  In this case the (branching) rule must have been used. Because $M \equiv \mathcal{T}_\xi$, we have $M_i \equiv \mathcal{T}_{\xi_i}$ for $\xi_1, \cdots, \xi_B$ such that $\xi \to \mathsf{b}(\xi_1, \cdots, \xi_B)$.
- $M' \equiv (\pi_{q'} \ldots)\cdots$. In this case, the *up* or *down* rule must have been used.
  - If *up* then, because $M \equiv \mathcal{T}_\xi$, after applying the projection rule, one obtains an F- or an N-term. In the former case, it will represent the configuration ξ' such that $\xi \to \xi'$ (via $\delta(q, m, \gamma) = (q', m', up_{\gamma'})$). In the latter case, after a single application of the rule for N, we can conclude the same.
  - If *down* then, because $M \equiv \mathcal{T}_\xi$, after applying the projection rule, one obtains a G- or $\perp$-term. In the former case, after a single application of the rule for G, we arrive at a term that represents ξ' such that $\xi \to \xi'$ (via $\delta(q, m, \gamma) = (q', m', down)$). The $\perp$-case can arise only for $M \equiv F^{\vec{u};d}_{q,m,\gamma_0}\cdots$. Because $M \equiv \mathcal{T}_\xi$, we can conclude that ξ has only γ_0 on the stack, i.e. the TSA cannot move.

This shows that $\mathcal{A}$ can simulate $\mathcal{R}_\mathcal{A}$. For the converse, since we already established $S = \mathcal{T}_{\xi_0}$, it suffices to observe that

- if $M \equiv \mathcal{T}_\xi$ and $\xi \to \xi'$ then:
  - if $\xi \to \xi'$ by (state change) we have $M = \mathcal{T}_{\xi'}$ (by a single F-rule);
  - if $\xi \to \xi'$ by (up) then we have $M = M' = \mathcal{T}_{\xi'}$ (by F-rule+projection) or $M = M' = M'' = \mathcal{T}_{\xi'}$ (by F-rule+projection+N-rule);
  - if $\xi \to \xi'$ by ($down$) then we have $M = M' = M'' = \mathcal{T}_{\xi'}$ (by F-rule+projection+G-rule);
- if $M \equiv \mathcal{T}_\xi$ and $\xi \to \mathsf{b}(\xi_1, \cdots, \xi_B)$ then $M = \mathsf{b}\langle \mathcal{T}_{\xi_1}, \cdots, \mathcal{T}_{\xi_B}\rangle$ (by F-rule).

Altogether we can conclude $\mathcal{A}$ and $\mathcal{R}_\mathcal{A}$ generate the same infinite tree.

E Auxiliary material (Section 6)

Recall the grammar for the Dyck language:

$$D \quad ::= \quad \epsilon \quad | \quad [D]D.$$

We are going to recast it in terms of triples of identical words. To represent words over $\{[,]\}$, we use the type $W = o \multimap o$ and terminals $[,] : W$. The empty word then corresponds to the identity function $I : W$. To represent triples of words of type W, we use the type $\mathsf{T}_3 = (W \multimap W \multimap W \multimap o) \multimap o$ on the understanding that a triple $\langle M_1, M_2, M_3\rangle$ is represented by $\lambda f. f M_1 M_2 M_3 : \mathsf{T}_3$.

Then the Dyck grammar can be lifted to type T_3 as follows, where the terminal $\mathsf{b} : o \multimap o \multimap o$ represents choice between the two rules.

$$D f = \mathsf{b}\langle f I I I, \mathsf{Concat}\,(\mathsf{Bracket}\, D)\, D\, f\rangle$$

where

- $\mathsf{Bracket} : \mathsf{T}_3 \multimap \mathsf{T}_3$ represents a function that adds outermost brackets to each component of the given triple $((x, y, z) \mapsto ([x], [y], [z]))$,
- $\mathsf{Concat} : \mathsf{T}_3 \multimap \mathsf{T}_3 \multimap \mathsf{T}_3$ concatenates components of two triples $((x_1, y_1, z_1), (x_2, y_2, z_2) \mapsto (x_1 x_2, y_1 y_2, z_1 z_2))$.

The two terms are defined below.

$$\begin{array}{rcl} \mathsf{Bracket}\, X\, f &=& X(\lambda xyz.f(Bx)(By)(Bz)) \\ Bwv &=& [(w(]v)) \\ \mathsf{Concat}\, X_1\, X_2\, f &=& X_1(\lambda x_1y_1z_1.X_2(\lambda x_2y_2z_2.f(Cx_1x_2)(Cy_1y_2)(Cz_1z_2))) \\ Cw_1w_2v &=& w_1(w_2v) \end{array}$$

It follows that

$$\mathsf{Concat}\,(\mathsf{Bracket}\, D)\, D\, f = D(\lambda x_1y_1z_1.D(\lambda x_2y_2z_2.f(Kx_1x_2)(Ky_1y_2)(Kz_1z_2))),$$

where $Kw_1w_2v = [(w_1(](w_2v)))$. This yields the equation used in the main body of the paper:

$$Df = \mathsf{b}\langle fIII, D(\lambda x_1y_1z_1.D(\lambda x_2y_2z_2.f(Kx_1x_2)(Ky_1y_2)(Kz_1z_2)))\rangle.$$

By unfolding the right-hand side one can obtain an infinite tree whose leaves are labelled with terms of the form $fMMM$, where M ranges over (representations of) words from the Dyck language. In order to generate branches that end in words from $L = \{w\#w\#w \mid w \in D\}$, it now suffices to merge the components of the triples into single words by passing a concatenating function:

$$S = D(\lambda xyz.x(\#(y(\#(z\$))))).$$

Note the use of two new terminals: $\# : W$ to separate the words and $\$: o$ to end the branch.

Above we have used λ-syntax for brevity. Below we give an equivalent definition based on applicative terms, obtained by introducing auxiliary non-terminals for various subterms of the λ-terms.

$$\begin{array}{rcl} S &=& D(\mathsf{Init}) \\ \mathsf{Init}\, x\, y\, z &=& x(\#(y(\#(z\$)))) \\ S\, f &=& \mathsf{b}\langle fIII, D(Ef)\rangle \\ I\, v &=& v \\ E\, f\, x_1\, y_1\, z_1 &=& D(Ffx_1y_1z_1) \\ F\, f\, x_1\, y_1\, z_1\, x_2\, y_2\, z_2 &=& f(K\, x_1\, x_2)(K\, y_1\, y_2)(K\, z_1\, z_2) \\ K\, w_1\, w_2\, v &=& [(w_1(](w_2v))) \end{array}$$

F Auxiliary material (Section 7)

First we explain how to define a pushdown system $\mathcal{P}_{\mathcal{A}} = \langle Q, \Gamma^\bullet, \Delta\rangle$ [4], where $\Delta \subseteq Q \times \Gamma^\bullet \times Q \times (\Gamma^\bullet)^*$, from a PDA $\langle \Sigma, Q, \Gamma, \delta, q_0, \gamma_0\rangle$ whose transition function has the form

$$\delta : Q \times \Gamma^\bullet \rightharpoonup Q + \{\mathsf{b}(q_1, \ldots, q_{|\mathsf{b}|}) \mid q_i \in Q,\, \mathsf{b} \in \Sigma\} + Q \times (\{up_\gamma \mid \gamma \in \Gamma\} + \{down\}).$$

We define Δ as the smallest set satisfying the rules below.

- If $\delta(q,\gamma) = q'$ then $(q,\gamma,q',\gamma) \in \Delta$.
- If $\delta(q,\gamma) = \mathsf{b}(q_1,\ldots,q_{|\mathsf{b}|})$ then $(q,\gamma,q_i,\gamma) \in \Delta$ for any $1 \le i \le |\mathsf{b}|$.
- If $\delta(q,\gamma) = (q', up_{\gamma'})$ then $(q,\gamma,q',\gamma'\gamma) \in \Delta$.
- If $\delta(q,\gamma) = (q', down)$ and $\gamma \neq \gamma_0$ then $(q,\gamma,q',\epsilon) \in \Delta$.

The condition $\gamma \neq \gamma_0$ corresponds to our convention that TSA block if *down* is executed on a stack containing γ_0 at the top. The corresponding configuration graph has $Q \times (\Gamma^\bullet)^*$ as nodes and edges $\Rightarrow$ are defined by

$$(q_1, \gamma w) \Rightarrow (q_2, w'w),$$

where $(q_1, \gamma, q_2, w') \in \Delta$ and $w \in (\Gamma^\bullet)^*$. Note that this graph is more general than the TSA configuration graph defined earlier, e.g. it contains configurations with empty stack or without γ_0. However, the two coincide on configurations reachable from (q_0, γ_0).

In the paper we take advantage of reachability analysis for pushdown systems [4, 9]. In particular, the relation $R_{\mathcal{A}} = \{(q, \gamma, q') \in Q \times \Gamma \times Q \mid (q, \gamma) \Rightarrow^* (q', \epsilon)\}$ can be calculated in polynomial time. Recall that an LPDA is PDA with a linearity property: any configuration reachable from (q_0, γ_0) can be reached via a unique path.

▶ **Lemma 32.** *For any LPDA $\mathcal{A}$, there exists an equivalent MHORS (of order 1) and its construction can be carried out in polynomial time.*

Proof. W.l.o.g. assume $Q = \{1, \cdots, N\}$ and use the following non-terminals $S : o$, $F_{(q,\gamma)} : \underbrace{o \multimap \cdots \multimap o}_{N} \multimap o$. The translation uses the same term representation of stacks as the 1-CPDA to 1-HORS translation in [12] except that reachability analysis ($R_{\mathcal{A}}$) is used to identify places where the variables actually get used. Intuitively, $F_{(q,\gamma)} x_1 \cdots x_N$ represents a configuration where γ is at the top of the stack, and each x_i corresponds to a configuration that would be reached if γ was popped and the automaton transitioned to state i.

For rules, we take $S = F_{(q_0,\gamma_0)} \bot \cdots \bot$ and

$$F_{(q,\gamma)} x_1 \cdots x_N = \begin{cases} F_{(q',\gamma)} x_1 \cdots x_N & \delta(q,\gamma) = q' & \text{(state change)} \\ \mathsf{b} B_1 \cdots B_{|\mathsf{b}|} & \delta(q,\gamma) = \mathsf{b}(q_1, \cdots, q_{|\mathsf{b}|}) & \text{(branching)} \\ x_{q'} & \delta(q,\gamma) = (q', \mathit{down}) & \text{(pop)} \\ F_{(q',\gamma')} D_1 \cdots D_N & \delta(q,\gamma) = (q', \mathit{up}_{\gamma'}) & \text{(push)} \end{cases}$$

where $B_i \equiv F_{(q_i,\gamma)} y_{i1} \cdots y_{iN}$, $D_i \equiv F_{(i,\gamma)} z_{i1} \cdots z_{iN}$ and

$$y_{ij} \equiv \begin{cases} x_j & (q_i, \gamma, j) \in R_{\mathcal{A}} \\ \bot & \text{otherwise} \end{cases} \qquad z_{ij} \equiv \begin{cases} x_j & (q', \gamma', i), (i, \gamma, j) \in R_{\mathcal{A}} \\ \bot & \text{otherwise} \end{cases}$$

Because the intended target of the translation is the setting of MHORS, we need to show that, in each of the rules above, every x_j occurs at most once on the right-hand side. This is easy to see for the (pop) and (state change) cases, so it remains to check (branching) and (pushing). We are going to argue that, if linearity were violated, we can actually replace the rhs with $\bot$, because the corresponding configurations would never be reached.

Consider the branching case first and suppose that x_j occurs twice in $\mathsf{b} B_1 \cdots B_{|\mathsf{b}|}$. Then we must have $(q_{i_1}, \gamma, j) \in R_{\mathcal{A}}$ and $(q_{i_2}, \gamma, j) \in R_{\mathcal{A}}$ for $i_1 \neq i_2$. Observe that then no configuration of the form $(q, \gamma\square)$, where $\square \in (\Gamma^\bullet)^*$, can be reachable from (q_0, γ_0), because a run from (q_0, γ_0) to $(q, \gamma\square)$ can be extended (via $(q_{i_1}, \gamma\square)$ and $(q_{i_2}, \gamma\square)$ respectively) to two different runs to $(j, \square)$, which would contradict linearity. Consequently, we can set $F_{(q,\gamma)} x_1 \cdots x_N = \bot$.

Finally, consider the push case and suppose x_j occurs twice in $F_{(q',\gamma')} D_1 \cdots D_N$, i.e. there exist $i_1 \neq i_2$ such that $(q', \gamma', i_1), (i_1, \gamma, j) \in R_{\mathcal{A}}$ and $(q', \gamma', i_2), (i_2, \gamma, j) \in R_{\mathcal{A}}$. We argue that then no configuration of the shape $(q, \gamma\square)$ is reachable from (q_0, γ_0). Indeed, if this were the case, we could extend the run (from (q_0, γ_0) to $(q, \gamma\square)$) with $(q, \gamma\square) \Rightarrow (q', \gamma'\gamma\square) \Rightarrow^* (i_k, \gamma\square) \rightarrow^* (j, \square)$ for $k = 1, 2$, i.e. $(j, \square)$ would be reachable via two runs, contradicting linearity. Consequently, if x_j happens to occur twice, we can replace the rule with $F_{(q,\gamma)} x_1 \cdots x_N = \bot$. ◀