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.2019.50

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

( LT 170023).

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

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 (→) 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 surprisingly, so far this nearly automata-theoretic
On the Expressivity of Linear Recursion Schemes

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 $\rightarrow$ and $\&$. We define the **typed terms** directly by the typing rules of Figure 1. Typing judgments have the form $\Gamma \vdash t : \varphi$, where $\Gamma$ and $\Delta$ are two lists of variable declarations. Intuitively, $\Delta$ 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, $\Gamma$ comprises duplicable variables that may be reused at will, as witnessed by the application rule. In $M(\Lambda)$HORS, $\Gamma$ will be used only for terminal and non-terminal symbols. Linear λ-terms are equipped with standard reduction rules; we write $\beta$ for $\beta$-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 $b : o \rightarrow \cdots \rightarrow o \rightarrow o$ with $n$ arguments. Here, a ranked alphabet $\Sigma$ may be represented in two distinct

---

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

### 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 MAHORS to TSA.

#### 3.1 Games and strategies

A game is a tuple $A = (M_A, \lambda_A, P_A)$ 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 $s = s_1 \ldots s_n \in P_A$, then $\lambda_A(s_1) = O$ and $\lambda_A(s_i) \neq \lambda_A(s_{i+1})$. We write $\epsilon \in P_A$ for the empty play and $s \subseteq s'$ for the prefix ordering.

---

[2] [5] considers also intermediate typings, but this does not contribute extra expressivity.
Games represent types. Plays in a game for a type $\varphi$ represent executions on $\varphi$ following (for this paper) a call-by-name evaluation strategy. For instance, Figure 2 shows a play in the game for $(o \to o) \to o \to o$, 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 ($o$, 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 $\lambda f^{o \to o}. \lambda x^o. f \; x : (o \to o) \to o \to o$.

Each type $\varphi$ may be interpreted as a game $\llbracket \varphi \rrbracket$. The game $\llbracket o \rrbracket$ has $M_{\llbracket o \rrbracket} = \{o\}$ with $\lambda(o) = O$, and $P_{\llbracket o \rrbracket} = \{\epsilon, o\}$. To match the type constructor $\to$, the linear arrow game $A \to B$ has as moves the tagged disjoint union $M_{A\to B} = M_A + M_B = \{1\} \times M_A \cup \{2\} \times M_B$ with polarity $\lambda_{A\to B}(1,a) = \lambda_A(a)$ and $\lambda_{A\to B}(2,b) = \lambda_B(b)$, where $\mathcal{O} = P$ and $\mathcal{P} = O$. The plays $P_{A\to B}$ include all $O$-starting, alternating sequences $s \in M_{A\to B}'$ 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, $A \to B$ 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 $\llbracket ((o \to o) \to o \to o) \rrbracket = ((\llbracket o \rrbracket \to \llbracket o \rrbracket) \to ((\llbracket o \rrbracket \to \llbracket o \rrbracket))$ includes four moves corresponding to the four atom occurrences, and has only two maximal plays: the one in Figure 2, and $o_3$. 

The tensor game $A \otimes B$ 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 $\to$, 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 $\sigma$ on $A$, written $\sigma : A$, is $\sigma \subseteq P_A^o$ (writing $P_A^o$ for the set of even-length plays) which is non-empty, closed under even-length prefix, and deterministic, in the sense that if $sab, sabb \in \sigma$, then $b = b'$. The interpretation of terms yields strategies; for instance

$$\llbracket \lambda f^{o\to o}. \lambda x^o. f \; x : (o \to o) \to o \to o \rrbracket = \{\epsilon, o_3, o_4, o_3 o_2 o_4 o_1\}$$

is a strategy on $\llbracket (o \to o) \to o \to o \rrbracket$ 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 $\sigma : A \rightarrow B$ and $\tau : B \rightarrow C$ we first let $\sigma, \tau$ interact by considering all sequences in $(M_A + M_B + M_C)^*$ whose restrictions to $A, B$ and $B, C$ are respectively in $\sigma$ and $\tau$; and then project those to $P_{A\rightarrow C}$ to obtain $\sigma \circ \tau : A \rightarrow C$. We omit the details [14]. Overall, the structure needed to interpret the linear $\lambda$-calculus with products is succinctly summarized by stating that games and strategies form a symmetric monoidal closed category with products [14] – to any $\_ : x_1 : \varphi_1, \ldots, x_n : \varphi_n \vdash t : \varphi$ this lets us associate $\llbracket t \rrbracket : \otimes_{i \subseteq \Sigma \varphi_i} \llbracket \varphi_i \rrbracket \rightarrow \llbracket \varphi \rrbracket$ in such a way that this is invariant under reduction – note however 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 \( f: M^O_A \rightarrow M^P_A \) 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 \( f: M^O_A \rightarrow M^P_A \), known as its history-free skeleton. For instance, the strategy \( \lambda f^o \circ o. \lambda x^o. fx \) with a unique maximal play in Figure 2, has history-free skeleton \( \{ o_4 \mapsto o_2, o_1 \mapsto o_3 \} \).

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 \( \sigma: A \rightarrow B \) and \( \tau: B \rightarrow C \) are history-free, their history-free skeletons, which have the types

\[
\sigma: M^P_A + M^O_B \rightarrow M^P_B + M^P_B \\
\tau: M^P_B + M^O_C \rightarrow M^P_C + M^P_C,
\]

may be composed via feedback on \( B \), pictured in Figure 3. For any Opponent move in \( A \rightarrow C \), we apply the corresponding function \( f_\sigma \) or \( f_\tau \). As long as the response is in \( B \), we keep applying \( f_\sigma \) and \( f_\tau \) alternately. This process may stay in \( B \) forever (a livelock, in which case the composition \( f_\tau \circ f_\sigma \) is undefined), but otherwise we eventually get a Player move in \( A \rightarrow C \) as required; defining a partial function \( f_{\tau \circ \sigma}: M^O_{A \rightarrow C} \rightarrow M^P_{A \rightarrow C} \). 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 \( \lambda \)-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 \( \lambda \)-terms with products are not in general history-free. For instance, Figure 4 displays the two maximal plays of a contraction/duplication strategy \( \lambda f^o \circ o. (f, f): (o \rightarrow o) \rightarrow ((o \rightarrow o) \circ (o \circ o)) \). It reacts to \( o_1 \) differently depending on the history. To account for this, one may replace partial functions \( f: M^O_A \rightarrow M^P_A \) with \( f: M^O_A \times M \rightarrow M^P_A \times M \), i.e. transducers, where \( M \), 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 \( \Sigma \) (the multiplicative/additive distinction plays no role here).

**Definition 2.** A transducer \( T \) on a game \( A \), written \( T: A \), is

\[
\mathcal{T} = (\mathcal{M}_- \cup \mathcal{M}_+, m_0, \delta_-, \delta_+)
\]

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:

\[
\delta_-: \mathcal{M}_- \times M^O_A \rightarrow \mathcal{M}_+
\]
\[
\delta_+: \mathcal{M}_+ \rightarrow \mathcal{M}_+ \cup \mathcal{M}_- \times M^P_A + \{ b(m_1, \ldots, m_{|b|}) | m_i \in \mathcal{M}_+, b \in \Sigma \}.
\]
On the Expressivity of Linear Recursion Schemes

50:6

The details, though rather direct, are too lengthy for the paper, so we instead present the \( \Sigma / \text{divides}.alt0 \) semantics is invariant under reduction, \( \delta \) in Figure 4 is generated using history-free strategies recovering \( \sigma \) connection ignoring the terminal symbols.

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

\[\text{Proposition 4. For any } \Sigma \mid - \vdash t : a, \ \text{Tree}(\{t\}) = \text{BT}(t).\]

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 \( m^+ \xrightarrow{\delta} n^+ \) when \( \delta \cdot (m^+, a) = n^+ \) and \( m^+ \xrightarrow{n^+} m^- \) when \( \delta \cdot (m^+, a) = m^- \) and \( m^+ \xrightarrow{b} m^- \) when \( \delta \cdot (m^+, b) = (m^-, b) \); the set \( \text{Traces} (\mathcal{T}) \) comprises all sequences \( s_1 \ldots s_{2n} \in M^*_A \) such that (with \( m_0, \ldots, m_n \in M_- \))

\[m_0 \xrightarrow{s_1} \xrightarrow{s_2} \ldots \xrightarrow{s_{2n-1}} \xrightarrow{s_{2n}} m_n.

We say that \( \mathcal{T} \) is a \textbf{strategic transducer} if for all \( s \in \text{Traces}(\mathcal{T}) \cap P_A \), if \( sa \in P_A \) and \( sb \in \text{Traces}(\mathcal{T}) \), then \( sab \in P_A \). Then, \( \text{Traces}(\mathcal{T}) \cap P_A \) is a strategy written \( \text{Strat}(\mathcal{T}) \). We say that \( \sigma : A \) has \textbf{finite memory} if \( \sigma = \text{Strat}(\mathcal{T}) \) for a strategic transducer \( \mathcal{T} \). We also recover \textbf{history-free strategies} as those for which \( M_- \) is a singleton. For instance, the strategy in Figure 4 is generated using \( M_- = \{m_0, m_1\} \) and \( M_+ = M_- \times M_0 \delta \cdot (m, a) = (m, a), \delta \cdot (\_, \sigma_4) = (m_0, \bullet_2), \delta \cdot (\_, \sigma_6) = (m_1, \bullet_2), \delta \cdot (m_0, \bullet_1) = (m_0, \bullet_4) \) and \( \delta \cdot (m_1, \bullet_1) = (m_1, \bullet_5) \).

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, \( \text{Strat}(\mathcal{T} \circ \mathcal{S}) = \text{Strat}(\mathcal{T}) \circ \text{Strat}(\mathcal{S}) \). This entails that for all \( t \), \( \text{Strat}(\{t\}) = [t] \). But for closed transducers \( \{t\} \) and tree-generating game semantics, \( \text{Tree}(\{t\}) = \text{Strat}(\{t\}) \). Since game semantics is invariant under reduction, \( [t] = [\text{BT}(t)] = \text{BT}(t) \), and Proposition 4 follows.
4 Game Semantics to TSA

The previous section lets us associate, to any \( \Sigma \vdash t : o \), 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 \( \mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S) \). By definition, for each \( F \in \mathcal{N} \) we have \( \Sigma, \mathcal{N} \vdash \mathcal{R}(F) : \mathcal{N}(F) \). 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 \( \Sigma \vdash N_1, \ldots, N_N \vdash \mathcal{R}'(F) : \mathcal{N}(F) \) obtained by giving different names \( G_1, \ldots, G_p \) (\( p \leq N \)) 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 \( \mathcal{B}(\mathcal{G}) \). For each \( F \in \mathcal{N} \) and \( n \in \mathbb{N} \), we redefine \( \Sigma \vdash \mathcal{N}(F) \) by setting \( \mathcal{N}(F) = 1 \), and \( \mathcal{N}(F) = \mathcal{R}'(F)[\mathcal{N}(G)/G_1 | G \in \mathcal{N}, 1 \leq i \leq N] \). 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:

\[
(\mathcal{R}'(F)) = \bigotimes_{1 \leq i \leq N} [\mathcal{N}(G)] \rightarrow [\mathcal{N}(F)].
\]

The unfolding above can then be replicated as follows.

\[\textbf{Proposition 5.} \] Setting \( T^0_F = 1 \) with all positive transitions undefined, and \( T^{n+1}_F = (\mathcal{R}'(F)) \otimes (\bigotimes_{1 \leq i \leq N} [\mathcal{N}(G)]) \), for all \( n \in \mathbb{N} \), we have \( \text{Tree}(T^n_S) = \mathcal{B}(\mathcal{N}(S)) \).

\[\textbf{Proof.}\] 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 \( n \in \mathbb{N} \), \( \mathcal{N}(F) \) and \( T^n_F \) are transducers generating the same finite memory strategy. By Proposition 4, \( \text{Tree}(T^n_S) = \text{Tree}(\mathcal{N}(S)) = \mathcal{B}(\mathcal{N}(S)) \).

Figure 6 displays the structure of transducer compositions arriving at the finite tree automaton \( T^3_F \). 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 $T_S^n$ 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 $[R(G)]$, 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 previously visited stack state/node, its local memory, accessed or updated only when visiting that node. The (local) transducer memory must be $m_0$. The green node is active, and all others passive.

Running a TSA 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 previously visited stack state/node, its local memory, accessed or updated only when visiting that node. The (local) transducer memory must be $m_0$. The green node is active, and all others passive.

Informally, the transitions operate as follows. Initially, only $\gamma_0$ is on the stack. Subsequently, given state $q$, local memory $m$, and top of the stack $\gamma \in \Gamma^*$:

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

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

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 $G$ with a restricted TSA $A(G)$ with states

$$Q = \left( \sum_{F \in \mathcal{N}} \mathcal{M}_{\vartheta_{1 \leq i < N} \mathcal{N}_{G\mathcal{N}}[\mathcal{N}(G)] \rightarrow [\mathcal{N}(F)]} \right) + \left( \sum_{F \in \mathcal{N}} \mathcal{M}_{\vartheta_{1 \leq i < N} \mathcal{N}_{G\mathcal{N}}[\mathcal{N}(G)] \rightarrow [\mathcal{N}(F)]} \right).$$
We use constructors Move and State to refer to elements from the left and right components of \( Q \) respectively. The memory alphabet is \( M = \sum_{F \in \mathcal{N}} M^{\mathcal{R}(F)}/\equiv \), 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 \( q_0 = \text{Move}(S,\varepsilon) \) and the transition function is given in Figure 7.

The TSA \( A(G) \) is designed so that a run of stack size bounded by \( n \) simulates a run of \( \mathcal{T}^n_\mathcal{S} \). When in state \( \text{State}(F,m) \), the automaton is currently operating in a \( F \) node of \( \mathcal{T}^n_\mathcal{S} \) (as in Figure 6), performing internal computation following \( \delta^F_n \). 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 up), or an Opponent move in \( \bigotimes_{1 \leq i \leq n} \bigotimes_{\mathcal{G} \in \mathcal{N}} \llbracket \mathcal{N}(G) \rrbracket \) (in which case we must move up, passing the control to a recursive call). If the state is \( \text{State}(G,m) \) 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 \( \text{Move}(F,a) \) in which the automaton reads the input move using \( \delta^F \) and resumes as above.

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

**Proof.** For \( n \geq 1 \), write \( \text{Tree}_n(A(G)) \) for the tree obtained from the truncated run-tree where the stack size is bounded by \( n-1 \) (where \( \gamma_0 \) has size 0). By construction, this truncated run-tree is weakly bisimilar to that of \( \mathcal{T}^n_\mathcal{S} \). In particular, \( \text{Tree}_n(A(G)) = \text{Tree}(\mathcal{T}^n_\mathcal{S}) = \text{BT}(\bigcup_{n \in \mathbb{N}} E_n(S)) \) by Proposition 5, so \( \text{Tree}(A(G)) = \text{BT}(G) \) by continuity.

This TSA is restricted: for any type \( \varphi \), 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 \( P_{[\bigotimes_{\mathcal{G} \in \mathcal{N}} \mathcal{N}(G)]} \). Then, \( A(G) \) is \( k \)-restricted. Indeed, fix a stack value \( \gamma_{n+1}\gamma_n\ldots\gamma_0 \) with \( \gamma_{n+1} = (F,i) \). Then, all transitions moving between \( \gamma_{n+1}\gamma_n\ldots\gamma_0 \) carry a move from \( M_{[\bigotimes_{\mathcal{G} \in \mathcal{N}} \mathcal{N}(G)]} \). By construction, the sequence of such moves forms a play in \( P_{[\bigotimes_{\mathcal{G} \in \mathcal{N}} \mathcal{N}(G)]} \). 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: \( M^{\mathcal{R}(F)} \) is a singleton. Consequently, \( A(G) \) 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 \( G \) be an MHORS. Then the tree-generating PDA \( A(G) \) 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 \( a \in M^+_\mathcal{S} \) there is at most one \( m \in M_+ \) such that \( \delta_m(a) = (\_,a) \) and for each \( m \in M_+ \) there is at most one \( (m',a) \in M_+ \times M^+_\mathcal{S} \) such that \( \delta_m(a) = (m',a) \) or at most one \( m' \in M_+ \) such that \( \delta_m(a) = m' \), and the two possibilities are mutually exclusive. Reversible strategic transducers are closed under all operations used in the interpretation, hence if \( \Sigma | \Delta = t : A \) involves no product, \( \{t\} \) is reversible (this phenomenon is well-known in GoI [6]). This entails that \( A(G) \) is linear. ▲
5 TSA to MAHORS

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

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

For $1 \leq d \leq k$, $F_{q,m,\gamma}^{d : \delta, u,B}$ has $B + 1$ arguments: the first $B$ arguments are used to simulate $u_{\gamma_j}$ ($1 \leq j \leq B$) 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 ($u_{\gamma_j}$), we call the $j$th 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 $1 \leq d \leq k$ are summarised in the table below, using $\lambda$ notation for brevity (for $F_{q,m,\gamma}^{d : \delta, u,B}$ we set $F_{q,m,\gamma}^{d : \delta, u,B} = 1$).

<table>
<thead>
<tr>
<th>$\delta(q,m,\gamma)$</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q'$</td>
<td>$F_{q,m,\gamma}^{d : \delta, u,B} x_1 \cdots x_B \gamma y = F_{q,m,\gamma}^{d : \delta, u,B} x_1 \cdots x_B \gamma y$</td>
</tr>
<tr>
<td>$b{q_1, \ldots, q_B}$</td>
<td>$F_{q,m,\gamma}^{d : \delta, u,B} x_1 \cdots x_B \gamma y = b{F_{q_1,m,\gamma}^{d : \delta, u,B} x_1 \cdots x_B \gamma y, \ldots, F_{q_B,m,\gamma}^{d : \delta, u,B} x_1 \cdots x_B \gamma y}$</td>
</tr>
<tr>
<td>$(q', m', \text{down})$</td>
<td>$F_{q,m,\gamma}^{d : \delta, u,B} x_1 \cdots x_B \gamma y = (\pi_{q'y}) {F_{q',m',\gamma}^{d : \delta, u,B} x_1 \cdots x_B \gamma y' y \in Q}$</td>
</tr>
<tr>
<td>$(q', m', \text{up}_{\gamma_j})$</td>
<td>$F_{q,m,\gamma}^{d : \delta, u,B} x_1 \cdots x_B \gamma y = (\pi_{q'y}) {\lambda z^{T_{\gamma_j}} F_{q',m',\gamma_j}^{d : \delta, u,B} x_1 \cdots x_{j-1} x_j \cdots x_B \gamma y' y \in Q}$</td>
</tr>
</tbody>
</table>

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 ($\gamma_j$). The use of the same $\gamma$ 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 $j$th 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 $j$th component incremented by 1). In the up case, we use a $\lambda$-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 $S = (F_{q_0,m_0,\gamma_0}^{0,0,1} N_{1} \cdots N_{B} | q \in Q)$. The divergent terms correspond to our convention that the automaton blocks when down is called at the root node. $N_{j} (1 \leq j \leq B)$ stands for $(N_{q,j} | q \in Q)$, where $N_{q,j}$ are auxiliary non-terminals that represent nodes visited for the first time. They are subject to the rule $N_{q,j} y = F_{q,m,\gamma}^{0,0,1} N_{1} \cdots N_{B} y$.

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

$\blacktriangleright$ 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 $a \& \cdots \& a \rightarrow a$. 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: $S : Fb, Ff = a(fS)(F(Gf)), Gfx = f(fx)$, where $a : o \rightarrow o \rightarrow o, b : o \rightarrow o$ and $S : o$ are terminals and $F : (o \rightarrow o) \rightarrow o$ and $G : (o \rightarrow o) \rightarrow o \rightarrow o$ are non-terminals. The scheme generates an infinite tree whose finite branches correspond to the language $L = \{a^nb^{2n-1} \mid n \geq 1\}$. 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 $(D = \epsilon [(D)|D])$, is not indexed [8] (see also page 2 of [16]). The MAHORS given below (using $\lambda$-syntax for brevity) has been obtained by lifting the grammar rules for $D$ to triples of words, encoded with the type $T_3 = ((o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow o) \rightarrow o$. 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 $b : (o \& o) \rightarrow o$ represents rule choice and the other terminals $([,], \# : o \rightarrow o, \$: o)$ are used to build the word. The scheme relies on the following non-terminals: $S : o, D : T_3, K : (o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow (o \rightarrow o)$ and $I : o \rightarrow o$, which are subject to the following rules:

$$
S = D(\lambda x y z. x(\#(y(\#(z\$)))), \quad Kxyv = [(x(\{yv\}))], \quad Iv = v,
Df = b(III, D(\lambda x y z. D(\lambda x_2 y_2 z_2. f(Kx_1x_2)(Ky_1y_2)(Kz_1z_2))))
$$

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. &ilde- and MAHORS. Recall from Lemma 8 that, for any MHORS, there exists an equivalent linear PDA (LPDA) \( (\Sigma, Q, \Gamma, \delta, q_0, \gamma_0) \) with transition function \( \delta : Q \times \Gamma^* \to Q + \{ b(q_1, \ldots, q_n) \mid q_i \in Q, b \in \Sigma \} + Q \times \{ (s, \gamma) \mid \gamma \in \Gamma \} + \{ \text{down} \} \) 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 \( (q, t) \in Q \times (\Gamma^*)^* \). As we do not have the space to review all the necessary definitions, let us just recall that the techniques employ multi-automata over \( \Gamma^* \) 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 \( A \), there exists a multi-automaton \( A_{era} \), 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 \( \text{pre}^*(Q \times \{ \epsilon \}) \). Hence, given \( A \), one can calculate the relation \( R_A = \{(q, \gamma, q') \in Q \times \Gamma \times Q \mid (q, \gamma) \Rightarrow^* (q', \epsilon)\} \) in polynomial time.

- For any LPDA \( A \), there exists a multi-automaton \( A_{era} \), constructible in polynomial time, which represents all configurations reachable from \( (q_0, \gamma_0) \), i.e. all \( (q, t\gamma_0) \) such that \( (q_0, \gamma_0) \Rightarrow^* (q, t\gamma_0) \). This corresponds to representing \( \text{post}^* \{(q_0, \gamma_0)\} \) [9].

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

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

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.

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

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

The above bound depends on \( s \). We show that there is a uniform bound, polynomial with respect to the size of \( A \). First, given \( s \in \Gamma^* \), the multi-automaton \( A_{era} \) 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 \( A_{era} \) (to those from which an original accepting state is reachable via an \( s\gamma_0 \)-labelled path). Let \( A_{era}^s \) be the resultant automaton. Note that the size of \( A_{era}^s \) is bounded by a polynomial in \(|A|\) that is independent of \( s \), because the only difference between \( A_{era}^s \) and \( A_{era} \) is the set of accepting states, and its size bounded by \(|Q|\).
Observe that \((\{q, t\} \mid (q_0, \gamma) \Rightarrow^* (q, ts\gamma), (q, t) \Rightarrow^* (q', \epsilon))\) for some \(q')\) is exactly the set of configurations that are represented by both \(A_{\epsilon o}^s\) and \(A_{\epsilon o}^t\). Consider the product \(A'\) of the two multi-automata. By Lemma 14, \(A'\) 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 \(|A|\). As this reasoning is independent of \(s\), we obtain:

**Lemma 15.** For any LPDA \(A\), there exists a bound \(H\), polynomial in \(|A|\), such that, for any \(s, t \in \Gamma^*\), if \((q, ts\gamma)\) 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 \(a, b: a \rightarrow o, o, F_1: o \rightarrow o\) \((1 \leq i \leq n)\) with \((S \Rightarrow F_n(bS)), F_0(x) = ax\) and \(F_i(x) = F_{i-1}(F_{i-1}x)\) for \(1 \leq i \leq n\) generates an infinite branch \((a^{2^i}b)^i\), which could only be generated by a 0-HORS of exponential size in \(n\).

### References


