On the Power of Additive Branching in Affine Higher-Order Recursion Schemes

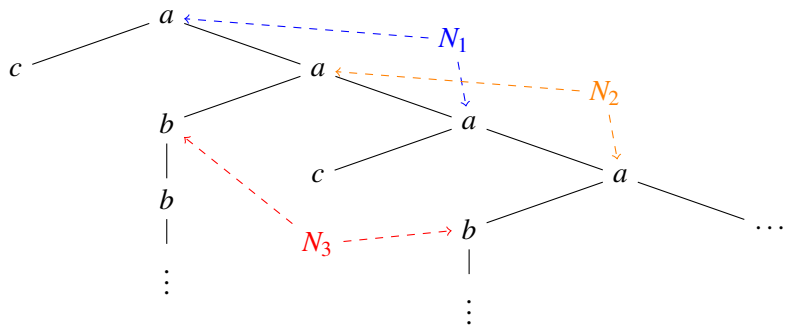
Lê Thành Dũng (Tito) Nguyễn
Laboratoire d'Informatique et des Systèmes (LIS)
CNRS & Aix-Marseille University, France
nltd@nguyentito.eu

Higher-order recursion schemes (HORS) are a formalism based on the λ-calculus for describing infinite trees; their study has been motivated by model-checking recursive functional programs. We investigate the expressive power of HORS subject to an affine typing discipline — more precisely, we revisit Clairambault and Murawski's multiplicative-additive HORS. Our contrasting results, which focus on understanding the expressiveness afforded by the use of the additive conjunction of linear logic, are:

- Every MAHORS with a multiplicatively typed tree alphabet generates a regular tree, i.e. is equivalent to some order-0 HORS. This generalizes a theorem of Clairambault and Murawski concerning multiplicative HORS, by allowing the types of non-terminals to contain additives. Unlike their proof, ours is purely syntactic and does not involve an automaton model.
- There is some MAHORS with an additively typed alphabet that is not equivalent to any safe HORS (in fact, it generates a known example of unsafe tree, related to Urzyczyn's language).

1 Introduction

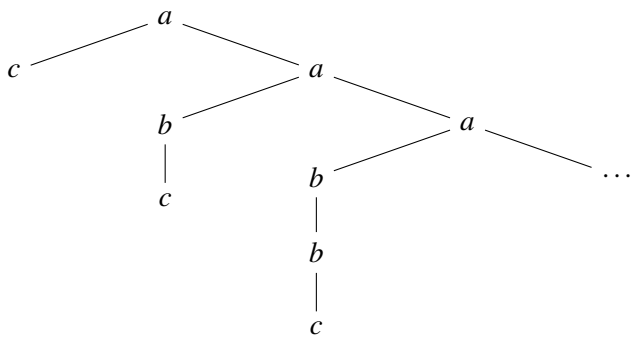
This paper is about possibly infinite trees defined by recursive systems of equations. By trees, we mean ranked trees with node labels in a ranked alphabet — in the case of finite trees, these are the same thing as first-order terms over a signature of function symbols. Here is an infinite example over the alphabet $\{a : 2,\ b : 1,\ c : 0\}$ (where, following a functional programming convention, $a(x,y)$ is written as $a\,x\,y$):



$$N_1 = a\,c\,N_2$$
$$N_2 = a\,N_3\,N_1$$
$$N_3 = b\,N_3$$

In general, the trees generated by systems of equations of the form "non-terminal symbol = first-order term over alphabet and non-terminals" are precisely the *regular trees*, i.e. those that have finitely many subtrees up to isomorphism (in the above example: N_1, N_2, N_3 and c). Some non-regular trees can be defined using equations that involve non-terminals with *higher-order* types (built from a base type o of trees). In these *higher-order recursion schemes* (HORS, see e.g. [CS21]), the right-hand side of equations are simply typed λ-terms rather than first-order terms. A standard example is:

Submitted to:
FICS 2026



$N_1 = N_2\, c : o$
$N_2 = \lambda x.\, a\, x\, (N_2\, (b\, x)) : o \to o$

In the past two decades, the study of HORS has been mainly motivated by the decidability of the monadic second-order (MSO) theory of the trees that they generate [Ong06].[1] This has led to the *higher-order model checking* approach to automated verification for functional programming languages (cf. e.g. [Kob19]): to check a property of interest of the program, express it as an MSO property of the tree defined by some HORS derived from the program.

Affine types, in the sense of linear logic, turn out to be useful to analyse the complexity of higher-order model checking [CGM17]: the idea is that affine functions, which can "use their argument at most[2] once", are less costly to handle in model-checking algorithms than unrestricted functions. This has led Clairambault and Murawski to study the case where all functions are required to be affine [CM19].

A key point in [CM19] is the difference between the *multiplicative* conjunction $A \otimes B$ and the *additive* conjunction $A \,\&\, B$ of linear logic (that is, the two pair types, via the proofs-as-programs correspondence). Their standard introduction rules are, respectively:

$$\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B} \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}$$

Intuitively, the first rule says that the resources required to build $A \otimes B$ are a disjoint sum of the resources required for A and those for B. On the other hand, the resources used to build $A \,\&\, B$ are used jointly by both sides of the pair, and this counts as a single use in the eyes of the type system: morally, having an $A \,\&\, B$ is a superposition between having an A and having a B, but not both at the same time.

The affine type system of [CM19] contains '&' but not '$\otimes$'.[3] That said, it has a function arrow '$\multimap$'; and in linear logic, $(A \otimes B) \multimap C$ is isomorphic to $A \multimap B \multimap C$. Now, observe in the above example of HORS that the letter a of rank 2 in the tree alphabet is encoded as a simply typed constant $a : o \to o \to o$. There are two possible affine versions of this type:

$$a : o \multimap o \multimap o \text{ (multiplicative)} \qquad a : (o \,\&\, o) \multimap o \text{ (additive)}$$

The above example of HORS can be typed with the additive version, by taking $N_2 = \lambda x.\, a\, \langle x, N_2\, (b\, x)\rangle$ — since the two occurrences of x are on two sides of an additive pair $\langle -, - \rangle$, they count as a single use. From the point of view of typing, a tree encoded with an additively typed alphabet may be thought as a superposition of its branches: when building this tree, the affineness restriction applies branchwise.

Clairambault and Murawski [CM19] then distinguish two classes of affine recursion schemes:

[1] This decidability theorem has many different proofs, mentioned for instance in the introduction to [Par21]. In particular, see [Gre16, Mel17] for a proof based on a denotational semantics coming from linear logic.

[2] A *linear* function would use its argument *exactly* once.

[3] One motivation for excluding both the multiplicative conjunction $\otimes$ and the additive disjunction $\oplus$ is that their *elimination* rules involve some complications, as often happens for positive connectives in natural deduction / λ-calculus.

- *Multiplicative* HORS cannot use '&' anywhere in the typing of their λ-terms. Necessarily, the alphabet must then be typed multiplicatively.
- *Multiplicative-Additive* HORS — abbreviated as MAHORS — are allowed to use both '$\multimap$' and '&'. Each letter in the tree alphabet can be typed either multiplicatively or additively, though one never loses any expressive power by opting for the latter. (Indeed, for $a : (o \,\&\, o) \multimap o$, one can use $\lambda x.\, \lambda y.\, a\,\langle x, y\rangle : o \multimap o \multimap o$ as a multiplicative version of a.)

Contribution 1: MAHORS with multiplicative alphabet. Clairambault and Murawski have shown that multiplicative HORS can only define regular trees [CM19, Section 7]. We prove that allowing additives *without using them for the tree alphabet* does not increase the expressive power:

Theorem 1.1 (proved in §3)**.** *All MAHORS with multiplicatively typed alphabets generate regular trees.*

Our proof only involves elementary syntactic manipulations on λ-terms, drawing inspiration from order reduction on linear higher-order tree transducers [GLS20, Section 3]. By contrast, the original arguments of [CM19, §7] combine a translation to linear pushdown automata with a reachability analysis; it is not clear whether these arguments could be adapted to prove Theorem 1.1.

Contribution 2: an intrinsically unsafe MAHORS. A historically important syntactic restriction on HORS is *safety*. It was introduced[4] by Knapkik, Niwiński and Urzyczyn [KNU02], who proved the decidability of MSO in this special case, and connected safe HORS to a natural automaton model.

While Clairambault and Murawski do not explicitly look at safety, they provide an example of a HORS that is not equivalent to any MAHORS [CM19, Example 11] — and one can check from the definitions that this example is safe. Conversely, is there some MAHORS that is not equivalent to any safe HORS? We provide an affirmative answer:

Theorem 1.2 (proved in §4)**.** *There is a MAHORS generating the intrinsically unsafe tree from [Par20].*

Of course, this MAHORS uses an additively typed alphabet. According to Theorem 1.1, this has to be the case, since all regular trees can be generated by safe schemes.

Warning. While the arguments sketched in this paper should be convincing enough to experts, they are not yet fully detailed and rigorous, due both to the deadline and the page limit. The missing details will be filled in the upcoming version submitted for the post-proceedings.

2 Definitions

Affine λ-calculus. The grammars of types and terms are

$$A, B ::= o \mid A \multimap B \mid A \,\&\, B \qquad t, u ::= x \mid a \mid \lambda x.\, t \mid t\, u \mid \langle t, u\rangle \mid \pi_1\, t \mid \pi_2\, t$$

where $x, y, \ldots$ are affine variables and $a, b, \ldots$ are reusable constants. Typing judgments have the form $\Theta \mid \Delta \vdash t : A$ where Θ (resp. Δ) lists the constants (resp. variables) with their types. The typing rules are the usual ones, cf. e.g. [CM19, Figure 1]; in particular, all rules handle Θ additively.

[4] In fact, the safety restriction is already implicit in the "derived types" that appear in the works of the 1980s on higher-order grammars [Dam82] and higher-order tree transducers [EV88] — for the relationship with the modern notion of safety, see e.g. [Par18]. Both [Dam82, EV88] include an equivalence with some kind of higher-order pushdown automaton, as in [KNU02]. There are also later characterisations of the trees generated by safe HORS using iterated graph transformations [Cau02, CW03], which are analogous to the decomposition of safe transducers into a sequence of macro tree transducers [EV88, Theorem 8.1].

Alphabets. A ranked alphabet is a finite set Σ endowed with a *rank* (or *arity*) map $\mathrm{rk}\colon \Sigma \to \mathbb{N}$. We have seen examples of possibly infinite trees over ranked alphabets in the introduction.

The alphabet Σ may be represented by either multiplicatively typed or additively typed constants:

$$\Sigma^{\otimes} = \{a : \underbrace{o \multimap \cdots \multimap o}_{\mathrm{rk}(a) \text{ times}} \multimap o \mid a \in \Sigma\} \qquad \Sigma^{\&} = \{a : \underbrace{(o \,\&\, \cdots \,\&\, o)}_{\mathrm{rk}(a) \text{ times}} \multimap o \mid a \in \Sigma\}$$

Recursion schemes. A MAHORS consists of a ranked alphabet Σ, a finite set of typed non-terminals $\{N_1 : A_1, \dots, N_n : A_n\}$, and a list of equations $N_i \doteq t_i$ with ($\Sigma^{\&}$ or $\Sigma^{\otimes}$), $N_1 : A_1,\ \dots,\ N_n : A_n \mid \varnothing \vdash t_i : A_i$ — the choice of $\Sigma^{\&}$ vs $\Sigma^{\otimes}$ should be consistent for all $i \in \{1, \dots, n\}$.

It gives rise to a notion of Böhm tree for any term u with $\Sigma^{\&\text{ or }\otimes}$, $N_1 : A_1,\ \dots,\ N_n : A_n \mid \Delta \vdash u : B$. More specifically, we take the definition of typed η-long[5] Böhm trees with pairs from [CGM17, Section 2.2.2], replacing the fixpoint unfolding rule $\to_\delta$ by the rules $N_i \to_\delta t_i$ for non-terminals. In particular, when $\Delta = \varnothing$ and $B = o$, the Böhm tree of u represents an infinite ranked tree over the alphabet $\Sigma \cup \{\bot : 0\}$, with the symbol $\bot$ indicating a diverging sub-computation.

We furthermore assume $A_1 = o$, and we say that the infinite ranked tree represented by the Böhm tree of the starting symbol N_1 is *the tree generated by the recursion scheme*.

Remark 2.1. Our definition of MAHORS is less restrictive than in [CM19] since we do not require the λs in t_i to be prenex. This makes no difference regarding the expressive power. (For instance, the translation to the equivalent $\lambda\ell\mathbf{Y}$-calculus given in [CGM17, Section 3.2.1] does not require prenex λs; alternatively, one can directly convert to an MAHORS in prenex form by introducing new non-terminals for inner λ-abstractions, a form of "defunctionalisation".)

Remark 2.2. We will also define recursion schemes recognising some word languages in Section 4.1.

3 MAHORS with multiplicative alphabet

3.1 Finiteness properties of normal affine λ-terms (unrelated to recursion schemes)

To prove Theorem 1.1, we start by some properties of independent interest. Let $\#_\odot(A)$ be the number of occurrences of the connective $\odot \in \{\multimap, \&\}$ in the type A. For a context $\Delta = x_1 : A_1,\ \dots,\ x_n : A_n$, we write $\#_\odot(\Delta) = \#_\odot(A_1) + \cdots + \#_\odot(A_n)$. We also use the notations $\#_{\langle,\rangle}(t)$ and $\#_\lambda(t)$ for the number of occurrences of additive pairs and λ-abstractions in a term t, respectively.

Lemma 3.1. *For t in normal form with $\Theta \mid \Delta \vdash t : A$ and $\#_\&(\Theta) = 0$, we have $\#_{\langle,\rangle}(t) \leqslant 2^{\#_\&(\Delta)+\#_\&(A)} - 1$.*

Proof idea. To use the assumption that t is normal, we leverage a direct mutually inductive description of norMal and Neutral terms in presence of pair types (see e.g. [Fio22, Proposition 7]):

$$M ::= \lambda x.M \mid \langle M, M\rangle \mid N \qquad N ::= x \mid a \mid NM \mid \pi_i N$$

where x ranges over variables and a ranges over constants. We then reason by structural induction. For neutral terms N with $\Theta \mid \Delta' \vdash N : A'$, our induction hypothesis is that $\#_{\langle,\rangle}(N) \leqslant 2^{\#_\&(\Delta') - \#_\&(A')} - 1$ — note the minus sign in the exponent! The case $N = a$ uses the assumption $\#_\&(\Theta) = 0$. □

We now assume that the constants are taken in $\Theta = \Sigma^{\otimes}$ for some fixed ranked alphabet Σ.

[5] These are not Nakajima trees (or similar) because the types bound the amount of η-expansion.

Lemma 3.2. *For t* in normal form *with $\Sigma^{\otimes} \mid \Delta \vdash t : A$, we have $\#_\lambda(t) \leqslant (\#_{\multimap}(\Delta) + \#_{\multimap}(A)) \times 2^{\#_{\&}(\Delta)+\#_{\&}(A)}$.*
Furthermore, the number of variable occurrences of t is also bounded by a function of Δ and A.

Proof idea. The bound on $\#_\lambda(t)$ can be established by structural induction, similarly to the previous lemma. To perform this induction, it is best to distinguish between neutrals with a head variable and neutrals with a head constant. In the latter case, i.e. $t = aM_1 \dots M_n$, the total number of occurrences of $\multimap$ in the typing judgments for $M_1, \dots, M_n$ is at most $\#_{\multimap}(\Delta) + \#_{\multimap}(A)$ *thanks to the fact that the constant $a \in \Sigma^{\otimes}$ has a type of the form $o \multimap \dots \multimap o \multimap o$.* Indeed, with a higher-order constant $\ell : (o \multimap o) \multimap o$ one could build terms such as $\ell(\lambda x_1.\ell(\lambda x_2.\ell(\dots(\lambda x_n.x_i)\dots)))$.

As for the bound on variable occurrences, it reduces to those on $\#_{\langle,\rangle}(t)$ and $\#_\lambda(t)$, thanks to the following affineness property: if we rewrite t by replacing each additive pair by one of its components, then in the resulting untyped term, each free variable from Δ appears at most once, and each λ-abstraction binds at most one variable occurrence. □

3.2 A purely syntactic proof of Theorem 1.1

Let us now fix a MAHORS *with multiplicatively typed alphabet*, given by the non-terminals $N_i : A_i$ with the equations $N_i \doteq t_i$ for $i = 1, \dots, n$, following the notations of Section 2. By reasoning on the finite approximants of these Böhm trees, the lemmas from the previous subsection give us:

Lemma 3.3. *The Böhm tree of any term u with $\Sigma^{\otimes}$, $N_1 : A_1, \dots, N_n : A_n \mid \Delta \vdash u : B$ has finitely many pairs, λ-abstractions and variable occurrences.*

Now, let us hereditarily head-normalize some N_i in a "breadth-first" fashion. Thanks to this lemma, we will get $N_i \to^* t'_i$ (in finite time) such that "every variable occurrence, λ and $\langle -,- \rangle$ is already in its eventual right place in the Böhm tree". That is, t'_i admits a decomposition $t'_i = u_i[v_{i,1}/x_1, v_{i,2}/x_2, \dots]$ such that the $\lambda\bot$-term $u_i[\bot/x_1, \bot/x_2, \dots]$ is an approximant of the Böhm tree of N_i *that contains all variable occurrences, λs and $\langle -,- \rangle$s in that Böhm tree*. Note that approximants are $\lambda\bot$-terms in normal form, with constants from $\Sigma^{\otimes}$ but *without non-terminals*. Thus, u_i is in normal form and non-terminal-free.

Meanwhile, each $v_{i,j}$ is an affine λ-term with potentially both constants and non-terminals, but *whose Böhm tree cannot contain any variable,*[6] *λ or $\langle -,- \rangle$*. This has two consequences:

- such a Böhm tree must have type o (because we work with η-long Böhm trees), so $v_{i,j} : o$;
- by substituting all free variables in $v_{i,j}$ by terms built using λ-abstractions, pairs and a new symbol $\Omega : o$ — to be used later as a non-terminal — we get a *closed* term $\hat{v}_{i,j} : o$ with *the same Böhm tree*.

Each $\hat{v}_{i,j}$ either head-normalizes (perhaps in 0 steps) to some term $v'_{i,j}$ which starts with a head constant, or diverges non-productively; in that second case, we take $v'_{i,j} = \Omega$. We can now define a new MAHORS using the non-terminals S (start symbol), $N'_{i,j}$ and Ω:

$$S \doteq u_1[N'_{1,1}/x_1, \dots] \qquad N'_{i,j} \doteq v'_{i,j}[(u_1[N'_{1,1}/x_1, \dots])/N_1, (u_2[N'_{2,1}/x_1, \dots])/N_2, \dots] \qquad \Omega \doteq \Omega$$

This new recursion scheme is designed to be equivalent to the original one. Since the left-hand side of the $N'_{i,j}$ rule has a head constant whenever $\hat{v}_{i,j}$ does not diverge non-productively, the recursion is productive when it should be. Finally, all non-terminals have type o, so the new HORS generates a regular tree — which finishes the proof of Theorem 1.1.

[6] Note that $v_{i,j}$ may contain variable occurrences that are "lost" by being "sent to infinity" during the reduction. This is one reason we prefer to work with affine types than linear types. See the recent "Ohana trees" [CMS25] for another take on this phenomenon in an untyped and relevant (as in relevance logic) setting.

4 An intrinsically unsafe MAHORS

The main example tree that separates safe and unsafe HORS, due to Parys [Par20], is best presented by taking a detour through word languages. It is a variant of an earlier example, attributed to Urzyczyn, of an unsafe scheme that was conjectured not to be equivalent to any safe scheme [KNU02, Example in §3]. While the HORS given in [KNU02] is somewhat enigmatic, its true explanation (found in later sources, e.g. [CS21, p. 1312]) was in terms of a word language.

We begin with a few general considerations on recognising (some) word languages with (MA)HORS, before looking at the intrinsically unsafe language that we are interested in.

4.1 HORS recognising prefix codes

Let Γ be a finite alphabet (without arities). We call a language $L \subset \Gamma^*$ a *precode* when no word in L has a strict prefix in L. (When $L \notin \{\varnothing, \{\varepsilon\}\}$, we get the standard notion of *prefix code* [BPR10, Chapter 3], which our terminology alludes to.) We present a notion of MAHORS serving as recognisers of precodes, which is a variant of the "labelled recursion schemes" from [CS21].

Definition 4.1. A *precode-recognising MAHORS* over Γ consists of $n \in \mathbb{N}$ and, for each $i \leqslant n$,

- a typed non-terminal $N_i : A_i$ whose type has the form $A_i = B_1 \multimap \cdots \multimap B_{k_i} \multimap o$, and
- either a rule $N_i\, x_1 \ldots x_{k_i} \to t_i$ such that $\top : o,\ N_1 : A_1,\ \ldots,\ N_n : A_n \mid x_1 : B_1,\ \ldots,\ x_{k_i} : B_{k_i} \vdash t_i : o$,
- or (exclusive) some labelled rules $N_i\, x_1 \ldots x_{k_i} \xrightarrow{\gamma} t_{i,\gamma}$ such that $t_{i,\gamma}$ satisfies the same typing judgment as t_i in the previous item, with at most one such rule for each $\gamma \in \Gamma$.

This defines a labelled head reduction relation on the terms u such that $\top : o,\ N_1 : A_1,\ \ldots \mid \varnothing \vdash u : o$, by combining head β-reduction (which is silent, i.e. labelled by the empty word) with the rules for non-terminals in head position. The language recognised by the MAHORS is the set of accepted words, which are the labels of reduction sequences from N_1 to $\top$.

Example 4.2. The following MAHORS recognises words of the form (w) where $w \in \{(,)\}^*$ is well-bracketed — the terms reached by head reduction are of the form $N\,(N\,(\ldots\,\top)\ldots)$, corresponding to the stack of the usual pushdown automaton for this language:

$$S : o \qquad S \xrightarrow{(} N\,\top \qquad\qquad N : o \multimap o \qquad N\,x \xrightarrow{(} N\,(N\,x) \qquad N\,x \xrightarrow{)} x$$

Remark 4.3. If $\top : o,\ N_1 : A_1,\ \ldots \mid \varnothing \vdash u : o$ then either u is normal, or there is a single head reduction for u which is silent, or all head reductions are labelled by letters and for each letter there is at most one. This is a form of determinism of the recognition process. Since $\top$ is normal, this entails that the language recognised by a recursion scheme (according to our definition) is a precode.

We now explain how to relate certain ranked trees to precodes.

Definition 4.4. Let $\Gamma = \{\gamma_1, \ldots, \gamma_{|\Gamma|}\}^*$ be an ordered alphabet and T be a tree over the ranked alphabet $\{\bullet : |\Gamma|,\ \top : 0,\ \bot : 0\}$. Each edge of T (if there is any) connects some node $\bullet$ to one of its children; let us label it by γ_i if it goes to the i-th child. The *branch language* $\mathscr{L}(T) \subset \Gamma^*$ consists of the labels of the paths from the root of T to its $\top$-labeled leaves; it is a precode.

Claim 4.5. *From any MAHORS recognising a precode $L \subset \Gamma^*$, one can build a MAHORS using the* additively typed *constants $\{\bullet : |\Gamma|,\ \top : 0\}^{\&}$ that generates a tree T such that $\mathscr{L}(T) = L$. (Note that $\bot : 0$ may also appear in T, as the leaf symbol indicating a divergence in the Böhm tree computation.)*

Proof. Here is one construction. Let $N_i : B_1 \multimap \cdots \multimap B_{k_i} \multimap o$ for $i \in \{1,\dots,n\}$ be the non-terminals of the precode-recognising MAHORS. We take the same non-terminals, plus $\Omega : o$, for the new MAHORS.

- First, we have the equation $\Omega \doteq \Omega$.
- If there is a silent rule $N_i x_1 \dots x_{k_i} \to t_i$ in the precode-recognising MAHORS, we put an equation $N_i \doteq \lambda x_1. \dots \lambda x_{k_i}. t_i$ in the tree-generating MAHORS.
- Otherwise, we take $N_i \doteq \lambda x_1. \dots \lambda x_{k_i}. \bullet \langle t_{i,\gamma_1}, \dots, t_{i,\gamma_{k_i}} \rangle$ with the convention that $t_{i,\gamma} = \Omega$ when there is no γ-labeled rule for N_i. The use of an additive tuple and the typing $\bullet : (o \,\&\, \cdots \,\&\, o) \multimap o$ are crucial: by definition, all the terms $t_{i,\gamma}$ separately satisfy the same typing judgment. □

Remark 4.6. While it would seem that Example 4.2 uses only multiplicative connectives at first glance, applying the above construction leads to $N \doteq \lambda x. \bullet \langle N(Nx), x \rangle$ which involves an additive pair. This is again an example of the contrapositive of Theorem 1.1: well-bracketed words with no well-bracketed prefixes form a non-regular language, which therefore cannot be the branch language of a regular tree.

Let U be the language introduced by Parys in [Par20, Section 4]. From its definition, given in the next subsection, one can check that it is a prefix code: it contains words of the form $w\#^n$ where $n \geqslant 1$ is determined by the #-free prefix w.

Theorem 4.7 (variant of [Par20, Theorem 1.1])**.** *There is no tree over* $\{\bullet, \top, \bot\}$ *generated by any safe HORS whose branch language is* U.

Proof. Suppose that T were such a tree. Then according to [KNU02, Theorem 5.3], the safe HORS generating T could be translated to a tree-generating higher-order pushdown automaton (HOPDA). In turn, this device could be turned into a deterministic HOPDA that accepts the word language $U = \mathscr{L}(T)$, by adapting arguments from [Par20, Section 3] to our definition of branch language. This would contradict the result that U cannot be recognised by a deterministic HOPDA [Par20, Theorem 1.2]. □

Thus, to prove Theorem 1.2, it suffices to exhibit a MAHORS that recognises U.

4.2 The prefix code U and a MAHORS that recognises it

Let us say that a word in $\{(,),\star\}^*$ is well-bracketed when the parentheses match ignoring the stars. That is, the set of well-bracketed words is the preimage of the Dyck language by the morphism that erases $\star$s.

Definition 4.8 ([Par20, §4])**.** $U = \{w\#^{\mathrm{stars}(w)+1} \mid w \in \{(,),\star\}^*\}$ where:

- if w is a prefix of a well-bracketed word (i.e. has no unmatched closed parenthesis), but is not itself well-bracketed, then $\mathrm{stars}(w)$ is the number of $\star$s before the last unmatched open parenthesis in w;
- otherwise, $\mathrm{stars}(w) = 0$.

Adapting Carayol and Serre's presentation [CS21, p. 1312] of Urzyczyn's precursor to the language U, we give an ad-hoc labelled transition system that recognises U.

Claim 4.9. *In the following LTS over* $\{Z, X\} \cup (\mathbb{N}^* \times \mathbb{N}) \cup \mathbb{N}$*, the language of paths from* Z *to* $\top$ *is* U.

$$Z \to ([],0) \qquad ([],n) \xrightarrow{\#} \top \qquad ([n_1,\dots,n_{\ell+1}],n) \xrightarrow{\#} n_{\ell+1} \qquad n+1 \xrightarrow{\#} n \qquad 0 \to \top$$

$$([],n) \xrightarrow{)} X \qquad X \xrightarrow{(\ or\ )\ or\ \star} X \qquad X \xrightarrow{\#} \top \qquad ([n_1,\dots,n_\ell],n) \xrightarrow{\star} ([n_1,\dots,n_\ell],n+1)$$

$$([n_1,\dots,n_\ell],n) \xrightarrow{(} ([n_1,\dots,n_\ell,n],n) \qquad ([n_1,\dots,n_\ell,n_{\ell+1}],n) \xrightarrow{)} ([n_1,\dots,n_\ell],n)$$

Furthemore, every state of the form $([n_1,\dots,n_\ell],n)$ *reachable from* Z *satisfies* $n_1 \leqslant \cdots \leqslant n_\ell \leqslant n$.

Idea. The states of the form $([n_1,\dots,n_\ell],n)$ represent the situation where:

- the input prefix read until now has n stars and is a prefix of some well-bracketed word;
- there are ℓ unmatched open parentheses, and the i-th one has n_i stars to its left. □

We would now like to design a precode-recognising MAHORS that somehow simulates this LTS. The HORS for Urzyczyn's language from [CS21] can be adapted to an HORS for U, but it does not seem to lend itself to being affinely typed. Instead we use a trick that somehow works: our MAHORS "stores" the difference $n-n_\ell$, i.e. the number of stars seen since the last unmatched open parenthesis.

Using a non-terminal $S : o \multimap o$, we encode each natural number m as the term $\underline{m} = \lambda x.\, S\,(\dots(S\,x)\dots)$ (where m occurs S times) of type $o \multimap o$. Note that $\underline{m} \circ \underline{n} =_\beta \underline{m+n}$ where $f \circ g = \lambda x.\, f\,(g\,x)$. States of the LTS in $\mathbb{N}$ are encoded as $\overline{m} = \underline{m}\ \top : o$.

To encode states in $\mathbb{N}^* \times \mathbb{N}$, we introduce the non-terminals $F : (((o \multimap o) \multimap o)\,\&\,o) \multimap (o \multimap o) \multimap o$ and $G : (o \multimap o) \multimap o$. We take $\overline{([n_1,\dots,n_\ell],n)} = \underline{[n_1,\dots,n_\ell]}\ \underline{(n-n_\ell)} : o$, with the convention $n_0 = 0$ (reused below in the subexpression $n_{\ell+1}-n_\ell$) where, inductively,

$$\underline{[]} = G \qquad \underline{[n_1,\dots,n_{\ell+1}]} = F\ \langle \lambda h.\ \underline{[n_1,\dots,n_\ell]}\ (\underline{(n_{\ell+1}-n_\ell)} \circ h),\ \overline{n_{\ell+1}} \rangle$$

This encoding of $[n_1,\dots,n_\ell]$ is redundant: it stores both $n_1,\dots,n_\ell$ and their successive differences, taking advantage of the monotonicity of the list.

Claim 4.10. *The language U is recognised by the following MAHORS, whose non-terminals are S,F,G typed as above and $Z,X : o$, with starting symbol Z (and where we use the abbreviation $I = \lambda x.x$):*

$$\begin{array}{ll}
Z \to G\,I & S\,x \xrightarrow{\#} x \\
G\,f \xrightarrow{\#} \top & F\,p\,f \xrightarrow{\#} \pi_2\,p \\
G\,f \xrightarrow{\star} G\,(S \circ f) & F\,p\,f \xrightarrow{\star} F\,p\,(S \circ f) \\
G\,f \xrightarrow{)} X & F\,p\,f \xrightarrow{)} (\pi_1\,p)\,f \\
G\,f \xrightarrow{(} F\,\langle \lambda h.\, G\,(f \circ h),\, f\,\top \rangle\,I & F\,p\,f \xrightarrow{(} F\,\langle \lambda h.\, F\,p\,(f \circ h),\, f\,(\pi_2\,p) \rangle\,I \\
X \xrightarrow{\#} \top & X \xrightarrow{\gamma} X \text{ for } \gamma \in \{(,),\star\}
\end{array}$$

Proof idea. One can check that it simulates the above LTS for U, modulo β-conversions. For example:

$$\begin{aligned}
\overline{([n_1,\dots,n_{\ell+1}],n)} &= F\ \langle \lambda h.\ \underline{[n_1,\dots,n_\ell]}\ (\underline{(n_{\ell+1}-n_\ell)} \circ h),\ \overline{n_{\ell+1}} \rangle\ \underline{(n-n_{\ell+1})} \\
&\xrightarrow{)} \pi_1\ \langle \lambda h.\ \underline{[n_1,\dots,n_\ell]}\ (\underline{(n_{\ell+1}-n_\ell)} \circ h),\ \overline{n_{\ell+1}} \rangle\ \underline{(n-n_{\ell+1})} \\
&=_\beta \underline{[n_1,\dots,n_\ell]}\ (\underline{(n_{\ell+1}-n_\ell)} \circ \underline{(n-n_{\ell+1})}) \\
&=_\beta \underline{[n_1,\dots,n_\ell]}\ \underline{(n_{\ell+1}-n_\ell+n-n_{\ell+1})} \\
&= \overline{([n_1,\dots,n_\ell],n)}
\end{aligned}$$

(To be fully rigorous, we would need to show that allowing these silent β-conversions — which are not always in head position — in an accepting run does not change the recognised language.) □

Acknowledgments. Thanks to Alixel Bagnis and Nathan Lhote for their collaboration (not covered here) on an alternative approach to Theorem 1.2 based on tree stack automata; and thanks to Pierre Clairambault, Thomas Colcombet, Axel Kerinec and Vincent Moreau for discussions on MAHORS.

References

[BPR10] Jean Berstel, Dominique Perrin & Christophe Reutenauer (2010): *Codes and Automata*. *Encyclopedia of Mathematics and its Applications* 129, Cambridge University Press, doi:10.1017/CBO9781139195768.

[Cau02] Didier Caucal (2002): *On Infinite Terms Having a Decidable Monadic Theory*. In Krzysztof Diks & Wojciech Rytter, editors: *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*, *Lecture Notes in Computer Science* 2420, Springer, pp. 165–176, doi:10.1007/3-540-45687-2_13.

[CGM17] Pierre Clairambault, Charles Grellois & Andrzej Murawski (2017): *Linearity in Higher-order Recursion Schemes*. *Proceedings of the ACM on Programming Languages* 2(POPL), pp. 39:1–39:29, doi:10.1145/3158127.

[CM19] Pierre Clairambault & Andrzej S. Murawski (2019): *On the Expressivity of Linear Recursion Schemes*. In Peter Rossmanith, Pinar Heggernes & Joost-Pieter Katoen, editors: *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 138, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 50:1–50:14, doi:10.4230/LIPIcs.MFCS.2019.50.

[CMS25] Rémy Cerda, Giulio Manzonetto & Alexis Saurin (2025): *Ohana Trees and Taylor Expansion for the λI-Calculus: No variable gets left behind or forgotten!* In Maribel Fernández, editor: *10th International Conference on Formal Structures for Computation and Deduction, FSCD 2025, Birmingham, UK, July 14-20, 2025*, *LIPIcs* 337, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 12:1–12:20, doi:10.4230/LIPICS.FSCD.2025.12.

[CS21] Arnaud Carayol & Olivier Serre (2021): *Higher-order recursion schemes and their automata models*. In Jean-Éric Pin, editor: *Handbook of Automata Theory*, European Mathematical Society Publishing House, Zürich, Switzerland, pp. 1295–1341, doi:10.4171/AUTOMATA-2/13.

[CW03] Arnaud Carayol & Stefan Wöhrle (2003): *The Caucal Hierarchy of Infinite Graphs in Terms of Logic and Higher-Order Pushdown Automata*. In Paritosh K. Pandya & Jaikumar Radhakrishnan, editors: *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science, 23rd Conference, Mumbai, India, December 15-17, 2003, Proceedings*, *Lecture Notes in Computer Science* 2914, Springer, pp. 112–123, doi:10.1007/978-3-540-24597-1_10.

[Dam82] Werner Damm (1982): *The IO- and OI-Hierarchies*. *Theoretical Computer Science* 20, pp. 95–207, doi:10.1016/0304-3975(82)90009-3.

[EV88] Joost Engelfriet & Heiko Vogler (1988): *High Level Tree Transducers and Iterated Pushdown Tree Transducers*. *Acta Informatica* 26(1/2), pp. 131–192, doi:10.1007/BF02915449.

[Fio22] Marcelo Fiore (2022): *Semantic analysis of normalisation by evaluation for typed lambda calculus*. *Mathematical Structures in Computer Science* 32(8), pp. 1028–1065, doi:10.1017/S0960129522000263.

[GLS20] Paul Gallot, Aurélien Lemay & Sylvain Salvati (2020): *Linear High-Order Deterministic Tree Transducers with Regular Look-Ahead*. In Javier Esparza & Daniel Král', editors: *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, *LIPIcs* 170, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 38:1–38:13, doi:10.4230/LIPIcs.MFCS.2020.38.

[Gre16] Charles Grellois (2016): *Semantics of linear logic and higher-order model-checking*. Ph.D. thesis, Université Paris 7. Available at `https://theses.hal.science/tel-01311150/`.

[KNU02] Teodor Knapik, Damian Niwiński & Paweł Urzyczyn (2002): *Higher-Order Pushdown Trees Are Easy*. In Mogens Nielsen & Uffe Engberg, editors: *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, *Lecture Notes in Computer Science* 2303, Springer, pp. 205–222, doi:10.1007/3-540-45931-6_15.

[Kob19] Naoki Kobayashi (2019): *10 Years of the Higher-Order Model Checking Project (Extended Abstract)*. In Ekaterina Komendantskaya, editor: *Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages, PPDP 2019, Porto, Portugal, October 7-9, 2019*, ACM, pp. 2:1–2:2, doi:10.1145/3354166.3354167.

[Mel17] Paul-André Melliès (2017): *Higher-order parity automata*. In: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, IEEE, Reykjavik, Iceland, pp. 1–12, doi:10.1109/LICS.2017.8005077.

[Ong06] C.-H. Luke Ong (2006): *On Model-Checking Trees Generated by Higher-Order Recursion Schemes*. In: *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, IEEE Computer Society, pp. 81–90, doi:10.1109/LICS.2006.38.

[Par18] Paweł Parys (2018): *Homogeneity Without Loss of Generality*. In Hélène Kirchner, editor: *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, *LIPIcs* 108, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 27:1–27:15, doi:10.4230/LIPIcs.FSCD.2018.27.

[Par20] Paweł Parys (2020): *On the Expressive Power of Higher-Order Pushdown Systems*. *Logical Methods in Computer Science* 16(3), doi:10.23638/LMCS-16(3:11)2020.

[Par21] Paweł Parys (2021): *Higher-Order Model Checking Step by Step*. In Nikhil Bansal, Emanuela Merelli & James Worrell, editors: *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, *LIPIcs* 198, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 140:1–140:16, doi:10.4230/LIPIcs.ICALP.2021.140.