Safety is Not a Restriction at Level 2 for String Languages*

K. Aehlig**, J. G. de Miranda, and C.-H. L. Ong

Oxford University Computing Laboratory

Abstract. Recent work by Knapik, Niwiński and Urzyczyn (in FOSSACS 2002) has revived interest in the connexions between higher-order grammars and higher-order pushdown automata. Both devices can be viewed as definitions for term trees as well as string languages. In the latter setting we recall the extensive study by Damm (1982), and Damm and Goerdt (1986). There it was shown that a language is accepted by a level-n pushdown automaton if and only if the language is generated by a *safe* level-n grammar. We show that at level 2 the safety assumption may be removed. It follows that there are no *inherently* unsafe string languages at level 2.

1 Introduction

Higher-order pushdown automata and higher-order grammars were originally introduced as definitional devices for string languages by Maslov [10] and Damm [4] respectively. Damm defined an infinite hierarchy of languages, the OI Hierarchy, the nth level of which is generated by level-n grammars that satisfy a syntactic constraint called *safety*[1]. Similarly, Maslov defined an infinite hierarchy, the nth level of which is generated by level-n pushdown automata (or nPDA). It was then shown [5] that the OI and Maslov hierarchies coincide: a language is generated by a level-n *safe* grammar if and only if it is accepted by a level-n pushdown automaton.

Recently, Knapik *et al.* [7, 8] have re-introduced higher-order grammars and higher-order pushdown automata as definitional devices for *term trees*. Not surprisingly, safety is, again, key to connecting the two. They show that a term tree is generated by a safe level-n grammar if and only if it is accepted by a level-n pushdown automaton. Furthermore, if a term tree is generated by a safe grammar it enjoys a decidable monadic second order (MSO) theory. This latter result has sparked much interest among communities interested in the verification of infinite-state systems.

* This is an extended abstract of a longer paper [2] complete with proofs, which is downloadable from the authors' web pages.

** On leave from Mathematisches Institut, Ludwig-Maximilians-Universität München. Supported by a postdoctoral fellowship of the German Academic Exchange Organisation (DAAD).

[1] Formerly referred to as the restriction of "derived types".

In light of Knapik *et al.*'s result, it seems important to investigate why safety *appears* to be key to such good algorithmic behaviour and desirable properties. To date, no results concerning unsafe grammars (whether in the string-language or term-tree setting) exist. We recall two questions raised by Knapik *et al.* [8]. First, is safety required to guarantee MSO decidability of term trees? Secondly, is safety required (whether in the string-language or term-tree setting) for the equivalence between higher-order grammars and higher-order pushdown automata?

In this paper we make a first attempt at tackling the second of the above problems. We analyse the string-language case and show that at level 2 the restriction is redundant. Precisely we show that every string language generated by a level-2 *unsafe* grammar can be generated by a level-2 *safe* grammar. Hence we arrive at the title of our paper. We conjecture that this is not the case for the term-tree setting.

We briefly sketch a proof of our main result (Theorem 1). By examining why Knapik *et al.*'s translation [8] of higher-order grammars to PDAs fails for unsafe grammars, we discover an important relationship between items in the higher-order store of the PDA in question. To formalise the idea, we introduce a new kind of machine, called *2PDA with links* (2PDAL for short), which is just a 2PDA such that each 1-store that is pushed has a *fresh* link to the item below that has "caused" the push_2 action. When performing a pop_2 subsequently, these links serve as a means of determining the number of 1-stores to pop off. We show that a 2PDAL can implement (i.e. accept the same language as that generated by) a level-2 grammar, whether safe or not. Unfortunately there is no a priori bound on the number of links required, so it is not obvious how a 2PDAL can be directly translated to a 2PDA. However, by a careful analysis of the way links behave, one can use a *non-deterministic* 2PDA to simulate a 2PDAL. Thus we have a way of transforming a (possibly unsafe) level-2 grammar to an equivalent 2PDA.

Related Work In [1] we address another question concerning safety: we show that the MSO theory for all string and tree-languages defined by level-2 grammars is decidable (previously this could only be asserted if the grammar was safe). An independent proof of the same decidability result has also been given by Knapik, Niwiński, Urzyczyn and Walukiewicz [9].

2 Definitions

In this section we introduce higher-order grammars and higher-order pushdown automata as definitional devices for string languages. However, in Section 6 we will relate our result to the term-tree setting [7, 8].

2.1 Higher-Order Grammars and Safety

Types and Terms *Simple types* (ranged over by A, B, etc.) are defined by the grammar: $A ::= o \mid (A \to B)$. Each type A can be uniquely written as

$(A_1, \cdots, A_n, o)$ for some $n \geq 0$, which is a shorthand for $A_1 \to \cdots \to A_n \to o$ (by convention $\to$ associates to the right). We define the *level* of a type by $\mathsf{level}(o) = 0$ and $\mathsf{level}(A \to B) = \max(\mathsf{level}(A)+1, \mathsf{level}(B))$. We say that $A = (A_1, \cdots, A_n, o)$ is *homogeneous* just if $\mathsf{level}(A_1) \geq \mathsf{level}(A_2) \geq \cdots \geq \mathsf{level}(A_n)$, and each A_i is homogeneous.

A *typed* alphabet is a set Δ of simply-typed symbols. We denote by Δ^A the subset of Δ containing precisely those elements of type A. The set of *applicative terms of type* A over Δ, denoted by $\mathcal{T}^A(\Delta)$, is defined by induction over the rules: (1) $\Delta^A \subseteq \mathcal{T}^A(\Delta)$; (2) if $t \in \mathcal{T}^{A \to B}(\Delta)$ and $s \in \mathcal{T}^A(\Delta)$ then $(ts) \in \mathcal{T}^B(\Delta)$. Finally, we write $t : A$ to mean $t \in \mathcal{T}^A$ and we define $\mathsf{level}(t)$ to be $\mathsf{level}(A)$.

Higher-Order Grammars and Safety A *higher-order grammar* is a tuple $G = \langle N, V, \Sigma, \mathcal{R}, S, e \rangle$ such that N is a finite set of homogeneously-typed non-terminals, and S, the *start symbol*, is a distinguished element of N of type o; V is a finite set of typed variables; Σ is a finite alphabet; $\mathcal{R}$ is a finite set of triples, called *rewrite rules* (also referred to as production rules), of the form

$$F x_1 \cdots x_m \stackrel{\alpha}{\longrightarrow} E$$

where $\alpha \in (\Sigma \cup \{\epsilon\})$, $F : (A_1, \cdots, A_m, o) \in N$, each $x_i : A_i \in V$, and E is either a term in $\mathcal{T}^o(N \cup \{x_1, \cdots, x_m\})$ or is $e : o$. We say that F has *formal parameters* $x_1, \cdots, x_m$. In the case where the grammar has two or more rules with the non-terminal F on the lefthand side, then we assume (w.l.o.g.) both rules have the same formal parameters in the same order. Following Knapik *et al.* [7] we assume that if $F \in N$ has type $(A_1, \cdots, A_m, o)$ and $m \geq 1$, then $A_m = o$. Thus, each non-terminal has at least one level-0 variable. Note that this is not really a restriction – as this variable need not occur on the righthand side.

We say that G is a *level-n grammar* (n-grammar for short) just in case n is the level of the non-terminal that has the highest level. We say that G is *deterministic* just if whenever $F x_1 \cdots x_m \stackrel{\alpha}{\longrightarrow} E$ and $F x_1 \cdots x_m \stackrel{\alpha'}{\longrightarrow} E'$ are both in $\mathcal{R}$, then (1) if $\alpha = \alpha'$ then $E = E'$ and (2) if $\alpha = \epsilon$ and $E \neq e$ then $\alpha' = \epsilon$ and $E = E'$.

We extend $\mathcal{R}$ to a family of binary relations $\stackrel{\alpha}{\longrightarrow}$ over $\mathcal{T}^o(N) \cup \{e\}$, where α ranges over $\Sigma \cup \{\epsilon\}$, by the rule: if $F x_1 \cdots x_m \stackrel{\alpha}{\longrightarrow} E$ is a rule in $\mathcal{R}$ where $x_i : A_i$ then for each $M_i \in \mathcal{T}^{A_i}(N)$ we have $F M_1 \cdots M_m \stackrel{\alpha}{\longrightarrow} E[\overline{M_i / x_i}]$.

A *derivation* of $w \in \Sigma^*$ is a sequence $P_1, P_2, \cdots, P_k$ of terms in $\mathcal{T}^o(N)$, and a corresponding sequence $\alpha_1, \cdots, \alpha_k$ of elements in $\Sigma \cup \{\epsilon\}$ such that

$$S = P_1 \stackrel{\alpha_1}{\longrightarrow} P_2 \stackrel{\alpha_2}{\longrightarrow} P_3 \stackrel{\alpha_3}{\longrightarrow} \quad \cdots \quad \stackrel{\alpha_{k-1}}{\longrightarrow} P_k \stackrel{\alpha_k}{\longrightarrow} e$$

and $w = \alpha_1 \cdots \alpha_k$. The *language* generated by G, written $L(G)$, is the set of words over Σ that have derivations in G. We say that two grammars are *equivalent* if they generate the same language.

A grammar is said to be *unsafe* if there exists a rewrite rule $F x_1 \cdots x_m \stackrel{\alpha}{\longrightarrow} E$ such that E contains a subterm t (say) in an operand position (i.e. (st) is a subterm of E, for some s), and t contains an occurrence of x_i for some $1 \leq i \leq n$ such that $\mathsf{level}(t) > \mathsf{level}(x_i)$. Otherwise, the grammar is *safe*. It follows from the definition that all grammars of levels 0 and 1 are safe. This definition of safety follows the one presented by Knapik *et al.* [7,8]. In our technical report [2] we present an alternative definition of safety based on the *safe λ-calculus*. We make no use of this alternative characterisation here, but offer it to the interested reader as a natural way to understand the restriction and how it arises. For an example of an unsafe grammar, see Example 1.

The OI Hierarchy Damm [4] introduced the OI Hierarchy. The nth level of the hierarchy is generated by level-n grammars (defined differently from our grammars). Furthermore, each level is strictly contained in the one above it. The first three levels correspond to the regular, the context-free, and the indexed languages [3]. Damm's grammars are rewrite relations over expressions that are required to be objects of "derived types". An analysis of his definition reveals that the constraint of "derived types" is equivalent to the requirement that all types be *homogeneous* and the grammar be *safe*, both in the sense of Knapik *et al.* Assuming the grammar makes use of only homogeneous types (which all definitions in the literature do), it follows that safety and derived types are equivalent. In particular, it is routine to show that a level-n grammar using his definition corresponds to a *safe* n-grammar in our definition (and the converse holds too). For a comparison of the two (ours and Damm's) we point the reader to a note [6]. This note also motivates our preference for our definition.

Example 1. Consider the following *deterministic* and unsafe (because of the underlined expressions) grammar, where $\Sigma = \{ h_1, h_2, h_3, f_1, f_2, g_1, a, b \}$, with typed non-terminals $D : ((o,o),o,o,o)$, $H : ((o,o),o,o)$, $F : (o,o,o)$, $G : (o,o)$, $A, B : o$, variables φ, x, y and with rules:

$$\begin{array}{rlrlrl}
S & \xrightarrow{\epsilon} DGAB & H\varphi x & \xrightarrow{\epsilon} \varphi x & A & \xrightarrow{a} e \\
D\varphi xy & \xrightarrow{h_1} D(\underline{D\varphi x})y(\varphi y) & Gx & \xrightarrow{g_1} x & B & \xrightarrow{b} e \\
D\varphi xy & \xrightarrow{h_2} H(\underline{Fy})x & Fxy & \xrightarrow{f_1} x & & \\
D\varphi xy & \xrightarrow{h_3} \varphi B & Fxy & \xrightarrow{f_2} y & &
\end{array}$$

As this grammar is deterministic [6] each word in the language has a unique derivation. Hence, the reader can easily verify that the word $h_1 h_3 h_2 f_1 b$ is in the language, whereas $h_1 h_3 h_2 f_1 a$ is not.

2.2 Higher-Order Pushdown Automata

Fix a finite set Γ of *store symbols*, including a distinguished bottom-of-store symbol $\bot$. A *1-store* is a finite non-empty sequence $[a_1, \cdots, a_m]$ of Γ-symbols such that $a_i = \bot$ iff $i = m$. For $n \geq 1$, an $(n+1)$*-store* is a non-empty sequence of n-stores. Inductively we define the *empty* $(n+1)$*-store* $\bot_{n+1}$ to be $[\bot_n]$ where we set $\bot_0 = \bot$. (Note that n-store is sometimes called n-stack in the literature.) Recall the following standard operations on 1-stores:

– $\mathsf{push}_1(a)\ [a_1, \cdots, a_m] = [a, a_1, \cdots, a_m]$ for $a \in \Gamma - \{\perp\}$
– $\mathsf{pop}_1\ [a_1, a_2, \cdots, a_m] = [a_2, \cdots, a_m]$

For $n \geq 2$, the following set Op_n of *level-n operations* are defined over n-stores:

– $\mathsf{push}_n\ [s_1, \cdots, s_l] = [s_1, s_1, \cdots, s_l]$
– $\mathsf{push}_k\ [s_1, \cdots, s_l] = [\mathsf{push}_k\, s_1, s_2, \cdots, s_l], \quad 2 \leq k < n$
– $\mathsf{push}_1(a)\ [s_1, \cdots, s_l] = [\mathsf{push}_1(a)\, s_1, s_2, \cdots, s_l]$ for $a \in \Gamma - \{\perp\}$
– $\mathsf{pop}_n\ [s_1, \cdots, s_l] = [s_2, \cdots, s_l]$
– $\mathsf{pop}_k\ [s_1, \cdots, s_l] = [\mathsf{pop}_k\, s_1, s_2, \cdots, s_l], \quad 1 \leq k < n$

In addition we define $\mathsf{top}_n\ [s_1, \cdots, s_l] = s_1$ and $\mathsf{top}_k\ [s_1, \cdots, s_l] = \mathsf{top}_k\, s_1, 1 \leq k < n$. Note that $\mathsf{pop}_k\, s$ is undefined if the top k-store consists of only one element.

A *level-n pushdown automaton* (nPDA for short) is a tuple $\langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ where Q is a finite set of states; $q_0 \in Q$ is the start state; $F \subseteq Q$ is a set of accepting states; Σ the finite input alphabet; Γ the finite store alphabet (which is assumed to contain $\perp$); and $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \times Q \times Op_n$ is the transition relation.

A *configuration* of an nPDA is given by a triple (q, w, s) where q is the current state, $w \in \Sigma^*$ is the remaining input, and s is an n-store over Γ.

Given a configuration (q, aw, s) (where $a \in \Sigma$ or $a = \epsilon$, and $w \in \Sigma^*$), we say that $(q, aw, s) \to (p, w, s')$ if $(q, a, \mathsf{top}_1(s), p, \theta) \in \delta$ and $s' = \theta(s)$. The transitive closure of $\to$ is denoted by $\to^+$, whereas the reflexive and transitive closure is denoted by $\to^*$. We say that the input w is *accepted* by the above nPDA if $(q_0, w, \perp_n) \to^* (q_f, \epsilon, s)$ for some pushdown store s and some $q_f \in F$.

3 Relating nPDAs and n-grammars

3.1 The main result

Damm and Goerdt [5] showed that a string language is generated by a safe n-grammar if and only if it is accepted by an nPDA. To our knowledge, no results exist for *unsafe* n-grammars. In particular if G is an unsafe n-grammar, it is not known whether $L(G)$ is accepted by an nPDA, or perhaps a PDA of a higher level. Our main result is a first step towards solving this problem.

Theorem 1. *For any 2-grammar that is not assumed to be safe, there exists a non-deterministic 2PDA the accepts the language generated by the grammar. Moreover the conversion is effective.*

Our proof is split into two parts. Given a 2-grammar we first show that it can be implemented by a 2PDAL, where 2PDAL is a machine that has yet to be introduced (Section 4); we then show that a 2PDAL can be simulated by a non-deterministic 2PDA (Section 5). Combining our result with Damm and Goerdt's, we have:

Corollary 1. *Every string language that is generated by an unsafe 2-grammar can also be generated by some safe (non-deterministic) 2-grammar.*

3.2 An Example: Urzyczyn's Language

Before we explain our result and sketch a proof, we present an example of a deterministic but unsafe 2-grammar that generates a string language, which we shall call Urzyczyn's language, or simply U. We then show, via a "bespoke" proof, that U can be accepted by a 2PDA. We shall have occasion to revisit U later in Section 6 in the form of a conjecture.

The language U consists of words of the form $w *^n$ where w is a proper prefix of a well-bracketed word such that no prefix of w is a well-bracketed expression; each parenthesis in w is implicitly labelled with a number, and n is the label of the last parenthesis. The two labelling rules are:

I. The label of the opening **(** is one; the label of any subsequent **(** is that of the preceding **(** plus one.
II. The label of **)** is the label of the parenthesis that precedes the matching **(**.

For example, the following word is in U:

$$\begin{array}{cccccccccccccccccccc} (& (& (& (&) &) & (& (&) & (& (&) &) &) & (& (&) &) & * & * \\ 1 & 2 & 3 & 4 & 3 & 2 & 5 & 6 & 5 & 7 & 8 & 7 & 5 & 2 & 9 & 10 & 9 & 2 & & \end{array}$$

We shall first give an unsafe 2-grammar – call it G_U – that generates the language and then show that it is accepted by a 2PDA.

$$\begin{array}{llll} D\,\varphi\,x\,y\,z \xrightarrow{(} D\,(D\,\varphi\,x)\,z\,(F\,y)\,(F\,y) & & S \xrightarrow{(} D\,G\,E\,E\,E \\ D\,\varphi\,x\,y\,z \xrightarrow{)} \varphi\,y\,x & & F\,x \xrightarrow{*} x \\ D\,\varphi\,x\,y\,z \xrightarrow{*} z & & E \xrightarrow{\epsilon} e \end{array}$$

Remark 1. The language U [12] was motivated by a term tree that is conjectured in [8, p. 213] to be inherently unsafe.

Accepting *U* with a 2PDA In order to show that U is accepted by a 2PDA we make use of the following observation.

Proposition 1. *Let $y \in \{(,),*\}^*$. Then $y \in U$ if an only if it has a unique decomposition into $wx{*}^n$ where w is a proper prefix of a well-bracketed word such that no prefix of it (including itself) is well-bracketed and w ends in (; x is a (possibly empty) well-bracketed word; and n (the number of stars) is the number of ('s in w.*

In the preceding example, $w =$ **((** and $x =$ **(( )) (( ) (( ))) (( ))**.

Thanks to the decomposition in Proposition 1, the construction of a 2PDA that accepts U is very simple. We guess the prefix of the input that constitutes w and process w as though checking for a proper prefix of a well-bracketed expression (using the power of a 1PDA). At the same time we perform a push_2 for every **(** found. Thus, the number of 1-stores is equal to the number of **(**'s in w. After reading w we check that x is well-bracketed. When we first meet a $*$, if x was indeed well-bracketed, then we perform a pop_2 for each $*$ found.

$$(q_0, a, Dt_1 \cdots t_n) \to (q_0, \mathsf{push}_1(E)) \text{ if } Dx_1 \cdots x_m \xrightarrow{a} E \text{ and } n \le m \quad \text{(R1)}$$

$$(q_0, \epsilon, e) \to \text{accept} \quad \text{(R2)}$$

$$(q_0, \epsilon, x_j) \to (q_j, \mathsf{pop}_1) \text{ if } x_j : o \quad \text{(R3)}$$

$$(q_0, \epsilon, x_j t_1 \cdots t_n) \to (q_j, \mathsf{push}_2 \,;\, \mathsf{pop}_1) \text{ if } x_j \text{ has level} > 0 \quad \text{(R4)}$$

$$_{1 \le j \le n,}(q_j, \epsilon, \$t_1 \cdots t_n) \to (q_0, \mathsf{pop}_1 \,;\, \mathsf{push}_1(t_j)) \quad \text{(R5)}$$

$$_{j > n,}(q_j, \epsilon, \$t_1 \cdots t_n) \to (q_{j-n}, \mathsf{pop}_2) \quad \text{(R6)}$$

Fig. 1. Adapted transition rules from Knapik *et al.* [8]
Convention. In the Figure x_j means the j-th formal parameter of the relevant non-terminal. Furthermore, $\$ \in N \cup V$.

4 Simulating Higher-Order Grammars by 2PDALs

4.1 Understanding KNU's Proof

Knapik *et al.* [8] have shown that a term tree generated by a safe n-grammar is accepted by an nPDA. Their proof, based on a transformation of n-grammars to their corresponding nPDAs, can easily be adapted to work in the string-language setting.

Theorem 2. *Let G be a safe 2-grammar that generates a string language. Then $L(G)$ is accepted by some 2PDA.*

Proof. We use the same setup as Knapik *et al.* [8, Sect. 5.2], but now we incorporate an input string over the alphabet Σ. The transition function is given in Fig. 1.

Let us examine why the construction fails if we attempt to apply it (blindly) to an unsafe 2-grammar. As an example, we consider the grammar given in Example 1. Recall that the word $h_1 h_3 h_2 f_1 a$ is *not* in the language.

The automaton starts off in the configuration $(q_0, h_1 h_3 h_2 f_1 a, [[S]])$, after a few steps we reach the following configuration:

$$(q_0,\, h_2 f_1 a,\, [[\varphi B, D(D\varphi x)y(\varphi y), DGAB, S]])$$

As the topmost item, φB, is headed by a level-1 variable, we need to find out what φ is in order to proceed. Note that φ is the 1st formal parameter of the preceding item: $D(D\varphi x)y(\varphi y)$, i.e., it refers to $D\varphi x$. To this end, we perform a push_2 and then perform a pop_1, and replace the topmost item with $D\varphi x$. In other words, we have applied rule R4 followed by R5 to arrive at:

$$\begin{aligned}(q_0,\, h_2 f_1 a,\, [&[(D\varphi x)^{\langle 1- \rangle}, DGAB, S],\\ &[\varphi B^{\langle 1+ \rangle}, D(D\varphi x)y(\varphi y), DGAB, S]])\end{aligned}$$

Here we have labelled two store items, one with a $1-$ and the other with a $1+$. These labels are not part of the store alphabet, they have been added so that we may identify these two store items later on.

The crux behind their construction is the following. Suppose we meet the item $D\varphi x^{\langle 1-\rangle}$ later on in the computation, and suppose that we would like to request its third argument, meaning we would be in state q_3. Note, however, that D in $D\varphi x^{\langle 1-\rangle}$ has only 2 arguments. The missing argument can be found by visiting the item $\varphi B^{\langle 1+\rangle}$. Hence the labelling. We need to ensure that there is a systematic way to get from $D\varphi x^{\langle 1-\rangle}$ to $\varphi B^{\langle 1+\rangle}$ whenever we are in a state q_j for $j > 2$ and we have $D\varphi x^{\langle 1-\rangle}$ as our topmost symbol. This systematic way suggested by Knapik *et al.* is embodied by rule R6 of Fig. 1. It says that all we need to do is perform a pop_2, followed by a change in state to q_{j-2}, and to repeat if necessary.

After a few more steps of the 2PDA we will arrive at another configuration where the topmost symbol is headed by a level-1 variable:

$$\begin{aligned}(q_0,\, f_1 a,\, &\texttt{[[}\varphi x, H(Fy)x, (D\varphi x)^{\langle 1-\rangle}, DGAB, S\texttt{]},\\ &\texttt{[}\varphi B^{\langle 1+\rangle}, D(D\varphi x)y(\varphi y), DGAB, S\texttt{]]})\end{aligned}$$

Therefore, we next get:

$$\begin{aligned}(q_0,\, f_1 a,\, &\texttt{[[}Fy^{\langle 2-\rangle}, (D\varphi x)^{\langle 1-\rangle}, DGAB, S\texttt{]},\\ &\texttt{[}\varphi x^{\langle 2+\rangle}, H(Fy)x, (D\varphi x)^{\langle 1-\rangle}, DGAB, S\texttt{]},\\ &\texttt{[}\varphi B^{\langle 1+\rangle}, D(D\varphi x)y(\varphi y), DGAB, S\texttt{]]})\end{aligned}$$

Again we have labelled a new pair of store items, so that the same principle applies: if we want the missing argument of $Fy^{\langle 2-\rangle}$, then we will be able to find it at $\varphi x^{\langle 2+\rangle}$. After a few more steps we eventually reach the following crucial configuration:

$$\begin{aligned}(q_3,\, a,\, &\texttt{[[}(D\varphi x)^{\langle 1-\rangle}, DGAB, S\texttt{]},\\ &\texttt{[}\varphi x^{\langle 2+\rangle}, H(Fy)x, (D\varphi x)^{\langle 1-\rangle}, DGAB, S\texttt{]},\\ &\texttt{[}\varphi B^{\langle 1+\rangle}, D(D\varphi x)y(\varphi y), DGAB, S\texttt{]]})\end{aligned} \tag{1}$$

Intuitively, here we want the third argument of D in the expression $D\varphi x$. By rule R6 we arrive at: (in the following $\to_n$ means n steps of $\to$)

$$\begin{aligned}&(q_1,\, a,\, \texttt{[[}\varphi x^{\langle 2+\rangle}, H(Fy)x, (D\varphi x)^{\langle 1-\rangle}, DGAB, S\texttt{]},\\ &\qquad\qquad \texttt{[}\varphi B^{\langle 1+\rangle}, D(D\varphi x)y(\varphi y), DGAB, S\texttt{]]})\\ \to_2\, &(q_2,\, a,\, \texttt{[[}H(Fy)x, (D\varphi x)^{\langle 1-\rangle}, DGAB, S\texttt{]},\\ &\qquad\qquad \texttt{[}\varphi B^{\langle 1+\rangle}, D(D\varphi x)y(\varphi y), DGAB, S\texttt{]]})\\ \to_2\, &(q_2,\, a,\, \texttt{[[}(D\varphi x)^{\langle 1-\rangle}, DGAB, S\texttt{]},\\ &\qquad\qquad \texttt{[}\varphi B^{\langle 1+\rangle}, D(D\varphi x)y(\varphi y), DGAB, S\texttt{]]})\end{aligned}$$

$$\begin{aligned}
\rightarrow_2 (q_2,\, a,\, &\texttt{[[}DGAB, S\texttt{]},\\
&\qquad \texttt{[}\varphi B^{\langle 1+\rangle}, D(D\varphi x)y(\varphi y), DGAB, S\texttt{]]})\\
\rightarrow_2 (q_0,\, \epsilon,\, &\texttt{[[}e, A, S\texttt{]},\\
&\qquad \texttt{[}\varphi B^{\langle 1+\rangle}, D(D\varphi x)y(\varphi y), DGAB, S\texttt{]]})
\end{aligned}$$

Note that we have accepted $h_1 h_3 h_2 f_1 a$ which is incorrect! The construction only works under the assumption that the grammar is safe. However, the labels we have used lead us to the construction of a new kind of machine which can remedy this problem.

Provided that each time we create a new pair of labels (the $+$ and $-$ part), we ensure they are unique, then these labels provide a way of always jumping to the correct 1-store when we are looking for missing arguments. Why? Because each time we want the missing argument of an item labelled with $n-$, we would simply perform as many pop_2's as necessary until our topmost symbol was labelled with the corresponding $n+$. To see how this would work, let us backtrack to configuration (1) in the above example. Applying this idea of a parameterised pop_2, this brings us to:

$$(q_1,\, a,\, \texttt{[[}\varphi B^{\langle 1+\rangle}, D(D\varphi x)y(\varphi y), DGAB, S\texttt{]]})$$

which is indeed what we wanted, and it is easy to see the word will be rejected. This idea of using pairs of labels, which we call *links* is formalised in a new kind of machine called level-2 *pushdown automaton with links*, or simply 2PDAL.

4.2 Formal Definition of 2PDAL

Formally, a 2PDAL is a 2PDA with the added feature that each item can be decorated with labels from the set $\{n+ : n \geq 1\} \cup \{n- : n \geq 1\}$. It is possible for an item to have zero, one or two labels – no other possibilities exist. We write labels as superscripts, as in $a^{\langle\rangle}$ (or simply a), $a^{\langle 3+\rangle}$ and $a^{\langle 3+,4-\rangle}$. These superscripts are sets of at most two elements, ranged over by λ; thus we have $\langle 3+\rangle \cup \langle 4-\rangle = \langle 3+, 4-\rangle = \langle 4-, 3+\rangle$. In the case where an item has two labels, one of these will always be a $+$ and the other a $-$. These labels come in matching pairs. Thus, if there exists an item in the store labelled by $m-$ and another labelled by $m+$, together they are said to form an *instance* of the *link* m. We refer to the item that gains the $-$ as the *start point* and that which gains the $+$, the *end point*.

In addition to the usual operations of a 2PDA, a 2PDAL has an iterated form of pop_2, parameterised over links m, defined as follows: for s ranging over 2-stores

$$\mathsf{pop}_2(m)\, s = \begin{cases} s & \text{if } \mathsf{top}_1(s) \text{ has label } m+ \\ \mathsf{pop}_2(m)(\mathsf{pop}_2(s)) & \text{otherwise} \end{cases}$$

Given a 2-grammar G (not assumed to be safe) transitions of the corresponding 2PDAL, written 2PDAL_G, are defined by induction over the set of rules in

$$
\begin{aligned}
(q_0, a, Dt_1 \cdots t_n{}^{\lambda}) &\to (q_0, \mathsf{push}_1(E)) \text{ if } Dx_1 \cdots x_m \xrightarrow{a} E \text{ and } n \leq m \\
(q_0, \epsilon, e) &\to \text{accept} \\
(q_0, \epsilon, x_j) &\to (q_j, \mathsf{pop}_1) \\
(q_0, \epsilon, \varphi_j t_1 \cdots t_n{}^{\lambda}) &\to (q_0, \mathsf{repl}_1(\varphi_j t_1 \cdots t_n{}^{\lambda \cup \langle m+ \rangle})\,;\, \mathsf{push}_2\,;\, \mathsf{pop}_1\,;\, \mathsf{repl}_1(s_j{}^{\langle m- \rangle})) \\
&\qquad \text{where } m \text{ is fresh and } Ds_1 \cdots s_{n'}{}^{\lambda'} \text{ precedes } \varphi_j t_1 \cdots t_n{}^{\lambda}. \\
{}_{1 \leq j \leq n},\ (q_j, \epsilon, \$t_1 \cdots t_n{}^{\lambda}) &\to (q_0, \mathsf{repl}_1(t_j)) \\
{}_{j > n},\ (q_j, \epsilon, \$t_1 \cdots t_n{}^{\lambda}) &\to (q_{j-n}, \mathsf{pop}_2(m)) \text{ if } m- \in \lambda
\end{aligned}
$$

Fig. 2. Transition rules of the 2PDAL, 2PDAL_G

Fig. 2. For convenience we have written $\mathsf{repl}_1(a)$ as a shorthand for $\mathsf{pop}_1;\mathsf{push}_1(a)$. The store alphabet, Γ, is comprised of the start symbol S (as the bottom-of-store symbol) and a subset of the (finite) set of all subexpressions of the right hand sides of the productions in G. We assume that each production rule of the grammar assumes the following format:

$$F\varphi_1 \cdots \varphi_m x_{m+1} \cdots x_{m+n} \xrightarrow{a} E \tag{2}$$

where the φ's are used for level-1 parameters, and the x's are used for level-0 parameters. As in Knapik *et al.*, the set of states includes $\{q_i : 0 \leq i \leq M\}$, where M is the maximum of the arities[2] of any non-terminal or variable occurring in the grammar. As can be seen from Fig. 2 the automaton works in phases beginning and ending in distinguished states q_i with some auxiliary states in between.

Proposition 2. *The language of a (possibly unsafe) 2-grammar G is accepted by* 2PDAL_G.

Proof. (Sketch) Intuitively the correctness of this proposition should be clear. For a formal proof we find it useful to appeal to a new model of computation for higher-order grammars due to Stirling [11] called *pointer machines*. We show that a pointer machine for a grammar G can be simulated by 2PDAL_G. Unfortunately, owing to space constraints, we cannot give details of pointer machines here, but we point the interested reader to the technical report [2].

Remark 2. In an e-mail [12], Urzyczyn sketches a model of computation for evaluating (possibly unsafe) grammars called a "panic automaton". We understand that it can be shown that a level-2 panic automaton can be simulated by a 3PDA. However, only after our submission to FOSSACS'05 did a full account [9] of this new automaton become available. A preliminary reading of this account suggests that panic automata and PDALs are similar in many respects, but a detailed analysis of their relationships awaits further investigation. It should be mentioned that in [9] panic automata are used to give a proof of the MSO decidability of all *term trees* generated by level-2 grammars (as was mentioned at the end of Section 1).

[2] A term of type $A_1 \to \cdots \to A_n \to o$ is said to have arity n.

5 Simulating 2PDALs by Non-deterministic 2PDAs

The incorporation of labels (as names of links) into the store alphabet will, in general, lead to an infinite alphabet. Here we show how these links and the way in which they are manipulated can be simulated by a *non-deterministic* 2PDA.

5.1 Intuition

Note that in the running example of Section 4, only the link labelled by 1 was "followed", in the sense that we jumped from the $1-$ to the $1+$. The link labelled by 2, on the other hand, did not serve any purpose in this run.

The intuition behind simulating 2PDAL_G with a 2PDA relies on guessing which links are "useful" and *only* labelling those. We will see that "useful" links interact with one another in a very consistent and well-behaved way that will allow us to label them anonymously. We formalise this here.

We say that a link m is *queried* if we are in a configuration (q_i, w, s) where $i > 0$ and $\mathsf{top}_1(s) = \$t_1 \cdots t_n{}^{\lambda}$ with $m- \in \lambda$. Intuitively, querying a link m formalises the notion of "asking for a level-0 argument" from an item labelled with $m-$. We say that a link m is *followed* if the link m is queried (as above) and $i > n$. The following lemma is crucial:

Lemma 1. *Given a link m, m is queried at most once during the run of a* 2PDAL_G *for a 2-grammar G.*

The simulating non-deterministic 2PDA will follow the rules in Fig. 2 almost exactly. The difference is that each time we are about to generate a link we guess whether it will ever be followed in the future or not. We have the luxury of doing this precisely because of Lemma 1. Thus, we label the start and end points of the link if and only if we guess that it will be followed. Furthermore, instead of a fresh label m, we simply mark the start point with a $-$ and the end point with a $+$. Our non-deterministic 2PDA will thus have a finite store alphabet: $\Gamma \cup \{a^+ : a \in \Gamma\} \cup \{a^- : a \in \Gamma\} \cup \{a^{+/-} : a \in \Gamma\}$ where Γ is the store alphabet of the preceding section.

A Controlled Form of Guessing Now this presents a problem of ambiguity. Suppose we find ourselves in a configuration (q, w, s) where $\mathsf{top}_1(s)$ is labelled by $-$, how can we tell which of the store items labelled by a $+$ is the *true* end point of this link? (True in the sense that if we did have the ability to name our links as with 2PDAL_G, the topmost item would have label $m-$ for some m, and the *real* end point would have label $m+$ for the same m.) The answer lies in the use of a *controlled* form of guessing: when guessing whether a link will be followed in the future we require the guess to be subject to some constraints. We shall see that as a consequence the following invariant can be maintained:

> *Assume that the topmost* 1*-store has at least one item labelled by $-$. For the leftmost (closest to the top) of these, the corresponding end point*

can always be found in the first 1*-store beneath it whose topmost item is marked with a* $+$.[3]

Before formalising the controlled form of guessing, we introduce a definition. Let (q_0, w, s) be a reachable configuration of 2PDAL_G such that

$$\mathsf{top}_2(s) = [\varphi_{j_1} t_1 \cdots t_n^{\lambda}, A_1, \cdots, A_k, \cdots, A_N]$$

where $N \geq 2$. We say that φ_{j_1} *ultimately refers to* A_k just if:

(i) For $i = 1, \cdots, k-1$, the j_ith argument of A_i (where A_i is of the form $Ds_1 \cdots s_l$ for some $D \in N$ and some $l \geq j_i$) is a variable $\varphi_{j_{i+1}}$. We remind the reader of the notational convention set out in (2).
(ii) The j_kth argument of A_k is an application or a non-terminal.

Suppose that we are in a configuration (q_0, w, s) of the non-deterministic 2PDA where $\mathsf{top}_2(s) = [\varphi t_1 \cdots t_n^?, A_1, \cdots, A_k, \cdots, A_N]$ where ? may either denote a $-$ or no label at all. Furthermore, suppose that φ ultimately refers to A_k. Two possibilities exist:

A. None of the store items $\varphi t_1 \cdots t_n^?, A_1, \cdots, A_k$ are labelled by a $-$; or
B. There exists a store item in $\varphi t_1 \cdots t_n^?, A_1, \cdots, A_k$ labelled by a $-$.

In the first case we leave it up to the 2PDA to guess whether this link will be followed or not. In the second case, we *force* it to label $\varphi t_1 \cdots t_n$ (with $+$) as well as its matching partner (with $-$), thus committing it to following this link in the future.

We illustrate why this maintains the above invariant with an example. Consider:

$$\begin{bmatrix} [\varphi x_1 x_2, D\varphi x^-, F(F\varphi x)y, G\varphi x^-, \cdots] \\ [A^+, \cdots] \\ [B^+, \cdots] \end{bmatrix}$$

Note that the topmost store has two items labelled with a $-$, $D\varphi x$ and $G\varphi x$. By our invariant we know that $D\varphi x$ has end point A^+. And let us suppose that $G\varphi x$ points to B^+. Suppose that the φ of the topmost item ultimately refers to $F(F\varphi x)y$. Furthermore, suppose we go against our controlled form of guessing and allow the machine *not* to label $\varphi x_1 x_2$ and its matching partner. Thus we arrive at

$$\begin{bmatrix} [\varphi, F(F\varphi x)y, G\varphi x^-, \cdots] \\ [\varphi x_1 x_2, D\varphi x^-, F(F\varphi x)y, G\varphi x^-, \cdots] \\ [A^+, \cdots] \\ [B^+, \cdots] \end{bmatrix}$$

Now $G\varphi x$ is the leftmost item labelled with a $-$. Our invariant has been violated as the real end point of $G\varphi x$ is not A^+.

[3] The invariant is actually stronger than this, but this is sufficient to ensure that the simulation works correctly.

$$
\begin{aligned}
(q_0, a, Dt_1\cdots t_n^{\lambda}) &\to (q_0, \mathsf{push}_1(E)) \text{ if } Dx_1\cdots x_m \xrightarrow{a} E \text{ and } n \le m\\
(q_0, \epsilon, e) &\to \text{accept}\\
(q_0, \epsilon, x_j) &\to (q_j, \mathsf{pop}_1)\\
(q_0, \epsilon, \varphi_j t_1\cdots t_n) &\to \begin{cases} (q_0, \mathsf{repl}_1(\varphi_j t_1\cdots {t_n}^{+})\ ;\ \mathsf{push}_2\ ;\ \mathsf{pop}_1\ ;\ \mathsf{repl}_1({s_j}^{-})) \\ (q_0, \mathsf{push}_2\ ;\ \mathsf{pop}_1\ ;\ \mathsf{repl}_1(s_j)) \end{cases}\\
&\quad\ \text{if Situation A holds and } Ds_1\cdots {s_{n'}}^{\lambda'} \text{ precedes } \varphi_j t_1\cdots t_n\\
(q_0, \epsilon, \varphi_j t_1\cdots {t_n}^{\lambda}) &\to (q_0, \mathsf{repl}_1(\varphi_j t_1\cdots {t_n}^{+\cup\lambda})\ ;\ \mathsf{push}_2\ ;\ \mathsf{pop}_1\ ;\ \mathsf{repl}_1({s_j}^{-}))\\
&\quad\ \text{if Situation B holds and } Ds_1\cdots {s_{n'}}^{\lambda'} \text{ precedes } \varphi_j t_1\cdots {t_n}^{\lambda}\\
{}_{1\le j\le n},\ (q_j, \epsilon, \$t_1\cdots t_n^{\lambda}) &\to \begin{cases} (q_0, \mathsf{repl}_1(t_j)) \text{ if } - \notin \lambda \\ \text{abort} \qquad\qquad\ \text{if } - \in \lambda \end{cases}\\
{}_{j>n},\ (q_j, \epsilon, \$t_1\cdots t_n^{\lambda}) &\to \begin{cases} \text{abort} \qquad\qquad\ \text{if } - \notin \lambda \\ (q_{j-n}, \mathsf{pop}_2^{+}) \text{ if } - \in \lambda \end{cases}
\end{aligned}
$$

Fig. 3. Transition rules of the non-deterministic 2PDA, $2PDA_G$

In the above, Situations A and B refer to the two possibilities outlined in the preceding page regarding ultimate referral.

Penalty for Guessing Wrongly The cost of using non-determinism is that we commit ourselves to following our guesses. When we find out that we have guessed wrongly, we shall have to abort the run. There are two cases. Suppose we find ourselves in a configuration (q_j, w, s) where $\mathsf{top}_1(s) = \$x_1\cdots {x_n}^{-}$ and $j \le n$. The fact that the topmost item is labelled by $-$ means that we guessed that we would follow this link. We have guessed wrongly and we abort. Symmetrically if we reach (q_j, w, s) where $\mathsf{top}_1(s) = \$x_1\cdots x_n$ and $j > n$, then we also abort. Why? The absence of a $-$ label means that we guessed that we would *not* follow this link, but we are now about to turn against our original guess.

5.2 Definition of the Non-deterministic 2PDA, $2PDA_G$

Let G be a (possibly unsafe) 2-grammar. The transition rules of the corresponding non-deterministic 2PDA, $2PDA_G$, are given in Fig. 3. Note that we assume that production rules of the grammar assume the format given in rule (2). Let s range over 2-stores, we define $\mathsf{pop}_2^{+}(s) = p(\mathsf{pop}_2(s))$ where

$$
p(s) = \begin{cases} s & \text{if } \mathsf{top}_1(s) \text{ has label } + \\ p(\mathsf{pop}_2(s)) & \text{otherwise} \end{cases}
$$

Remark 3. In the definition of the transition rules (Fig. 3), in case the top_1 item of the 2-store is headed by a level-1 variable, the 2PDA has to work out whether situation A or B holds. This can be achieved by a little scratch work on the side:

do a push_2, inspect the topmost 1-store for as deep as necessary, followed by a pop_2. Alternatively we could ask the oracle to tell us whether it is A or B, taking care to ensure that a wrong pronouncement will lead to an abort.

Proposition 3. *Given a 2-grammar G, $2\mathrm{PDAL}_G$ can be simulated by $2\mathrm{PDA}_G$.*

Proof. (Sketch) It should be quite clear from Fig. 3 that $2\mathrm{PDA}_G$ behaves like a "crippled" $2\mathrm{PDAL}_G$. Thus, we can expect that if w is accepted by $2\mathrm{PDA}_G$, then w is accepted by $2\mathrm{PDAL}_G$. To show the converse requires a more delicate analysis of the behaviour of 2PDALs which we do not have space to contain here. Roughly, we assume the existence of an all-knowing oracle that can tell us whether or not a link will be followed in the future. All we then need to show is that the controlled form of guessing does not restrict the choices of the oracle – which it does not (i.e the controlled form of guessing is actually "sensible"). Full proofs of both directions are given in the technical report [2].

6 Urzyczyn's Language: a Conjecture about Term Trees

We have shown that the language U is accepted by a non-deterministic 2PDA. Based on the grammar G_U for Urzyczyn's language, we can construct the following term-tree generating grammar[4] over signature $\Sigma = \{\mathbf{(} : (o,o),\ \mathbf{)} : (o,o),\ * : (o,o),\ 3 : (o,o,o,o),\ e : o,\ r : o\}$ and with the following rewrite rules.

$$S \to \mathbf{(}DGEEE \qquad Fx \to *x$$

$$D\varphi xyz \to 3(\mathbf{(}D(D\varphi x)z(Fy)(Fy)\mathbf{)}\mathbf{(}\mathbf{)}\varphi yx)(*z) \qquad E \to e$$

$$G \to r$$

Proposition 4. *Suppose that the term tree generated by the above grammar can be generated by a safe (term-tree generating-) 2-grammar. Then the language U can be accepted by a deterministic 2PDA.*

Conjecture 1. U cannot be accepted by a deterministic 2PDA.

Conjecture 1 is closely related to a conjecture of Knapik *et al.*; see Remark 1. Thanks to Proposition 4, provided Conjecture 1 is true, we will have an example of an *inherently unsafe* term tree i.e. an unsafe 2-grammar whose term tree cannot be generated by a safe 2-grammar.

7 Further Directions

Let us recall our main result. We have shown that the string language of every level-2 grammar (whether safe of unsafe) can be accepted by a 2PDA. Combining this with earlier results [5] we have that there are no *inherently* unsafe string languages at level 2. This was a first attempt at understanding safety. However, our result leaves many questions unanswered:

[4] See [7, 8] for the term-tree definitions of grammars and PDAs.

– Does our result extend to levels 3 and beyond?
– What is the relationship between deterministic unsafe grammars and deterministic safe grammars? In particular, Conjecture 1.
– Is safety a requirement for MSO decidability? (An easy corollary of the result we have presented here is that LTL model-checking [13] is decidable for term trees generated by level-2 unsafe grammars – see technical report [2] for details. This has recently been superseded [1, 9].)
– It would be useful to have a "pumping lemma" for higher-order PDAs. We understand that Blumensath has a promising argument involving intricate surgeries on runs on an automaton; his ideas gives conditions under which such runs can be "pumped".

References

1. K. Aehlig, J. G. de Miranda, and C. H. L. Ong. The monadic second order theory of trees given by arbitrary level-two recursion schemes is decidable. *TLCA'05 (to appear)*.
2. K. Aehlig, J. G. de Miranda, and C. H. L. Ong. Safety is not a restriction at level 2 for string languages. Technical Report PRG-RR-04-23, OUCL, 2004.
3. A. Aho. Indexed grammars - an extension of context-free grammars. *J. ACM*, 15:647–671, 1968.
4. W. Damm. The IO- and OI-hierarchy. *TCS*, 20:95–207, 1982.
5. W. Damm and A. Goerdt. An automata-theoretical characterization of the OI-hierarchy. *Information and Control*, 71:1–32, 1986.
6. J. G. de Miranda and C. H. L. Ong. A note on deterministic pushdown languages. Available at `http://web.comlab.ox.ac.uk/oucl/work/jolie.de.miranda`, 2004.
7. T. Knapik, D. Niwiński, and P. Urzyczyn. Deciding monadic theories of hyperalgebraic trees. In *TLCA'01*, pages 253–267. Springer, 2001. LNCS Vol. 2044.
8. T. Knapik, D. Niwiński, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FOSSACS'02*, pages 205–222. Springer, 2002. LNCS Vol. 2303.
9. T. Knapik, D. Niwiński, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars, panic automata, and decidability. 25 October, 2004.
10. A. N. Maslov. The hierarchy of indexed languages of an arbitrary level. *Soviet Math. Dokl.*, 15:1170–1174, 1974.
11. C. Stirling. Personal email communication. 15 October, 2002.
12. P. Urzyczyn. Personal email communication. 26 July, 2003.
13. M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, pages 238–266. Springer-Verlag, 1995.