Unboundedness and Downwards Closures of Higher-Order Pushdown Automata

Matthew Hague[1], Jonathan Kochems[2] and C.-H. Luke Ong[2]

[1] Royal Holloway, Univsersity of London
[2] Department of Computer Science, Oxford University

Abstract

We show the diagonal problem for higher-order pushdown automata (HOPDA), and hence the simultaneous unboundedness problem, is decidable. From recent work by Zetzsche this means that we can construct the downwards closure of the set of words accepted by a given HOPDA. This also means we can construct the downwards closure of the Parikh image of a HOPDA. Both of these consequences play an important rôle in verifying concurrent higher-order programs expressed as HOPDA or safe higher-order recursion schemes.

1 Introduction

Recent work by Zetzsche [40] has given a new technique for computing the downwards closure of classes of languages. The downwards closure $\downarrow(\mathcal{L})$ of a language $\mathcal{L}$ is the set of all subwords of words in $\mathcal{L}$ (e.g. *aa* is a subword of *babab*). It is well known that the downwards closure is regular for any language [19]. However, there are only a few classes of languages for which it is known how to compute this closure. In general it is not possible to compute the downwards closure since it would easily lead to a solution to the halting problem for Turing machines.

However, once a regular representation of the downwards closure has been obtained, it can be used in all kinds of analysis, since regular languages are well behaved under all kinds of transformations. For example, consider a system that waits for messages from a complex environment. This complex environment can be abstracted by the downwards closure of the messages it sends or processes it spawns. This corresponds to a lossy system where some messages may be ignored (or go missing), or some processes may simply not contribute to the remainder of the execution. In many settings – e.g. the analysis of safety properties of certain kinds of systems – unread messages or unscheduled processes do not effect the precision of the analysis. Since many types of system permit synchronisation with a regular language, this environment abstraction can often be built into the system being analysed.

Many popular languages such as JavaScript, Python, Ruby, and even C++, include higher-order features – which are increasingly important given the popularity of event-based programs and asynchronous programs based on a continuation or callback style of programming. Hence, the modelling of higher-order function calls is becoming key to analysing modern day programs.

A popular approach to verifying higher-order programs is that of *recursion schemes* and several tools and practical techniques have been developed [23, 38, 26, 24, 30, 5, 6, 34]. Recursion schemes have an automaton model in the form of collapsible pushdown automata (CPDA) [18] which generalises an order-2 model called 2-PDA with links [1] or, equivalently, panic automata [22]. When these recursion schemes satisfy a syntactical condition called *safety*, a restriction of CPDA called *higher-order pushdown automata (HOPDA or n-PDA for order-n HOPDA)* is sufficient [29, 21]. HOPDA can be considered an extension of pushdown automata to a "stack of stacks" structure. It remains open as to whether CPDA are strictly more expressive than nondeterministic HOPDA when generating languages of words. It is known that, at order 2, nondeterministic HOPDA and CPDA generate the same word languages [1]. However, there exists a language generated by a deterministic order-2 CPDA that cannot be generated by a deterministic HOPDA of any order [31].

It is well known that concurrency and first-order recursion very quickly leads to undecidability (e.g. [33]). Hence, much recent research has focussed on decidable abstractions and restrictions (e.g. [14, 4, 20, 27, 13, 37, 28, 10, 16]). Recently, these results have been extended to concurrent versions of CPDA and recursion schemes (e.g. [35, 25, 15, 32]). Many approaches rely on combining representations of the Parikh image of individual automata (e.g. [13, 17, 16]). However, combining Parikh images of HOPDA quickly leads to undecidability (e.g. [17]). In many cases, the downwards closure of the Parikh image would be an adequate abstraction.

Computing downwards closures appears to be a hard problem. Recently Zetzsche introduced a new general technique for classes of automata effectively closed under rational transductions – also referred to as a *full trio*. For these automata the downwards closure is computable iff the *simultaneous unboundedness problem (SUP)* is decidable.

Definition 1.1 (SUP [40]). *Given a language* $\mathcal{L} \subseteq a_1^* \dots a_\alpha^*$ *does* $\downarrow(\mathcal{L}) = a_1^* \dots a_\alpha^*$*?*

Theorem 1.1. *[40, Theorem 1] Let* $\mathcal{C}$ *be class of languages that is a full trio. Then downward closures are computable for* $\mathcal{C}$ *if and only if the SUP is decidable for* $\mathcal{C}$*.*

Zetzsche used this result to obtain the downwards closure of languages definable by 2-PDA, or equivalently, languages definable by *indexed grammars* [2]. Moreover, for classes of languages closed under rational transductions, Zetzsche shows that the simultaneous unboundedness problem is decidable iff the *diagonal problem* is decidable. The diagonal problem was introduced by Czerwiński and Martens [11]. Intuitively, it is a relaxation of simultaneous unboundedness that is insensitive to the order the characters are output. For a word w, let $|w|_a$ be the number of occurrences of a in w.

Definition 1.2 (Diagonal Problem [11]). *Given language* $\mathcal{L}$ *we define*

$$\mathit{Diagonal}_{a_1,\dots,a_\alpha}(\mathcal{L}) = \forall m.\exists w \in \mathcal{L}.\forall 1 \leq i \leq \alpha.|w|_{a_i} \geq m \ .$$

The diagonal problem asks whether $\mathit{Diagonal}_{a_1,\dots,a_\alpha}(\mathcal{L})$ *holds of* $\mathcal{L}$*.*

In this work, we generalise Zetzsche's result for 2-PDA to the general case of n-PDA. We show that the diagonal problem is decidable. Since HOPDA are closed under rational transductions, we obtain decidability of the simultaneous unboundedness problem, and hence a method for constructing the downwards closure of a language defined by a HOPDA.

Corollary 1.1 (Downwards Closures). *Let* P *be an* n*-PDA. The downwards closure* $\downarrow(\mathcal{L}(P))$ *is computable.*

Proof. From Theorem 6.2 (proved in the sequel), we know that the diagonal problem for HOPDA is decidable. Thus, using Zetzsche, we can construct the downwards closure of P. □

This result provides an abstraction upon which new results may be based. It also has several immediate consequences:

1. decidability of separability by piecewise testable languages, from Czerwiński and Martens [11],
2. decidability of reachability in parameterised systems of HOPDA communicating asynchronously via a shared global register, from La Torre *et al.* [36],
3. decidability of finiteness of a language defined by a HOPDA, and

4. computability of the downwards closure of the Parikh image of a HOPDA.

We present our decidability proof in two stages. First we show how to decide $\text{Diagonal}_a(P)$ for a single character and HOPDA P in Sections 3 and 4. In Sections 5, 6, and 7 we generalise our techniques to the full diagonal problem.

In Section 3.1 we give an outline of the proof techniques for deciding $\text{Diagonal}_a(P)$. In short, the outermost stacks of an n-PDA are created and destroyed using $push_n$ and pop_n operations. These $push_n$ and pop_n operations along a run of an n-PDA are "well-bracketed" (each $push_n$ has a matching pop_n and these matchings don't overlap). The essence of the idea is to take a standard tree decomposition of these well-bracketed runs and observe that each branch of such a tree can be executed by an $(n-1)$-PDA. We augment this $(n-1)$-PDA with "regular tests" that allow it to know if, each time a branch is chosen, the alternative branch could have output some a characters. If this is true, then the $(n-1)$-PDA outputs a single a to account for these missed characters. We prove that, although the $(n-1)$-PDA outputs far fewer characters, it can still output an unbounded number iff the n-PDA could. Hence, by repeating this reduction, we obtain a 1-PDA, for which the diagonal problem is decidable since it is known how to compute their downwards closures [39, 9].

In Section 6.1 we give an outline of the generalisation of the proof to the full problem $\text{Diagonal}_{a_1,\ldots,a_\alpha}(P)$. The key difficulty is that it is no longer enough for the $(n-1)$-PDA to follow only a single branch of the tree decomposition: it must follow one branch for each of the $a_1, \ldots, a_\alpha$. Hence, we define HOPDA that can output trees with a bounded number (α) of branches. We then show that our reduction can generalise to HOPDA outputting trees (relying essentially on the fact that the number of branches is bounded).

2 Preliminaries

2.1 Downwards Closures

Given two words $w = \gamma_1 \ldots \gamma_m \in \Sigma^*$ and $w' = \sigma_1 \ldots \sigma_l \in \Sigma^*$ for some alphabet Σ, we write $w \leq w'$ iff there exist $i_1 < \ldots < i_m$ such that for all $1 \leq j \leq m$ we have $\gamma_j = \sigma_{i_j}$. Given a set of words $\mathcal{L} \subseteq \Sigma^*$, we denote its downwards closure $\downarrow(\mathcal{L}) = \{w \mid w \leq w' \in \mathcal{L}\}$.

2.2 Trees

A Σ-labelled finite tree is a tuple $T = (D, \lambda)$ where Σ is a set of node labels, and $D \subset \mathbb{N}^*$ is a finite set of nodes that is prefix-closed, that is, $\eta\,\delta \in D$ implies $\eta \in D$, and $\lambda : D \to \Sigma$ is a function labelling the nodes of the tree.

We write ε to denote the root of a tree (the empty sequence). We also write

$$a[T_1, \ldots, T_m]$$

to denote the tree whose root node is labelled a and has children $T_1, \ldots, T_m$. That is, we define $a[T_1, \ldots, T_m] = (D', \lambda')$ when for each δ we have $T_\delta = (D_\delta, \lambda_\delta)$ and $D' = \{\delta\eta \mid \eta \in D_\delta\} \cup \{\varepsilon\}$ and

$$\lambda'(\eta) = \begin{cases} a & \eta = \varepsilon \\ \lambda_\delta(\eta') & \eta = \delta\eta' \end{cases} .$$

Also, let $T[a]$ denote the tree $(\{\varepsilon\}, \lambda)$ where $\lambda(\varepsilon) = a$. A *branch* in $T = (D, \lambda)$ is a sequence of nodes of T, $\eta_1 \cdots \eta_n$, such that $\eta_1 = \epsilon$, $\eta_n = \delta_1\,\delta_2 \cdots \delta_{n-1}$ is maximal in D, and $\eta_{j+1} = \eta_j\,\delta_j$ for each $1 \leq j \leq n-1$.

2.3 HOPDA

HOPDA are a generalisation of pushdown systems to a stack-of-stacks structure. An order-n stack is a stack of order-$(n-1)$ stacks. An order-n push operation pushes a new order-$(n-1)$ stack onto the stack that is a copy of the existing topmost order-$(n-1)$ stack. Rewrite operations update the character that is at the top of the topmost stacks.

Definition 2.1 (Order-n Stacks)**.** *The set of order-n stacks $\mathcal{S}^{\Gamma}_n$ over a given stack alphabet Γ is defined inductively as follows.*

$$\begin{array}{rcl} \mathcal{S}^{\Gamma}_0 & = & \Gamma \\ \mathcal{S}^{\Gamma}_{k+1} & = & \left\{ [s_1 \dots s_m]_{k+1} \mid \forall i. s_i \in \mathcal{S}^{\Gamma}_k \right\} \; . \end{array}$$

Stacks are written with the top part of the stack to the left. We define several operations.

$$\begin{array}{rcll} \mathrm{top}_k([s_1 \dots s_m]_k) & = & s_1 & \\ \mathrm{top}_k([s_1 \dots s_m]_n) & = & \mathrm{top}_k(s_1) & n > k \\ & & & \\ \mathrm{rew}_\gamma([\gamma_1 \dots \gamma_m]_1) & = & [\gamma\, \gamma_2 \dots \gamma_m]_1 & \\ \mathrm{rew}_\gamma\,([s_1 \dots s_m]_n) & = & [\mathrm{rew}_\gamma(s_1)\; s_2 \dots s_m]_n & n > 1 \\ & & & \\ \mathrm{push}_k([s_1 \dots s_m]_k) & = & [s_1\; s_1 \dots s_m]_k & \\ \mathrm{push}_k([s_1 \dots s_m]_n) & = & [\mathrm{push}_k(s_1)\; s_2, \dots, s_m]_n & n > k \\ & & & \\ \mathrm{pop}_k([s_1 \dots s_m]_k) & = & [s_2 \dots s_m]_k & \\ \mathrm{pop}_k([s_1 \dots s_m]_n) & = & [\mathrm{pop}_k(\, s_1)\; s_2, \dots, s_m]_n & n > k \end{array}$$

and set

$$\mathrm{Ops}_n = \{\mathrm{rew}_\gamma \mid \gamma \in \Gamma\} \cup \{\mathrm{push}_k, \mathrm{pop}_k \mid 1 \leq k \leq n\}$$

to be the set of order-n stack operations.

For example

$$\begin{array}{rcl} \mathrm{push}_2\big([[\gamma\; \sigma]_1]_2\big) & = & [[\gamma\; \sigma]_1\, [\gamma\; \sigma]_1]_2 \\ \mathrm{rew}_\sigma\big([[\gamma\; \sigma]_1\, [\gamma\; \sigma]_1]_2\big) & = & [[\sigma\; \sigma]_1\, [\gamma\; \sigma]_1]_2 \; . \end{array}$$

Definition 2.2 (HOPDA or n-PDA)**.** *An order-n* higher order pushdown automaton (HOPDA or n-PDA) *is given by a tuple $(\mathcal{P}, \Sigma, \Gamma, \mathcal{R}, \mathcal{F}, p_{in}, \gamma_{in})$ where $\mathcal{P}$ is a finite set of control states, Σ is a finite output alphabet (that contains the empty word character ϵ), Γ is a finite stack alphabet, $\mathcal{R} \subseteq \mathcal{P} \times \Gamma \times \Sigma \times Ops_n \times \mathcal{P}$ is a set of transition rules, $\mathcal{F}$ is a set of accepting control states, $p_{in} \in \mathcal{P}$ is the initial control state, and $\gamma_{in} \in \Gamma$ is the initial stack character.*

We write $(p, \gamma) \xrightarrow{a} (p', o)$ for a rule $(p, \gamma, a, o, p') \in \mathcal{R}$.

A configuration of an n-PDA is a tuple $\langle p, s \rangle$ where $p \in \mathcal{P}$ and s is an order-n stack over Γ. We have a transition $\langle p, s \rangle \xrightarrow{a} \langle p', s' \rangle$ whenever we have $(p, \gamma) \xrightarrow{a} (p', o)$, $\mathrm{top}_1(s) = \gamma$, and $s' = o(s)$.

A run over a word $w \in \Sigma^*$ is a sequence of configurations $c_0 \xrightarrow{a_1} \cdots \xrightarrow{a_m} c_m$ such that the word $a_1 \dots a_m$ less the empty characters ε is w. It is an accepting run if $c_0 = \langle p_{\mathrm{in}}, [\![\gamma_{\mathrm{in}}]\!]_n \rangle$ — where we write $[\![\gamma]\!]_n$ for $[\cdots [\gamma]_1 \cdots]_n$ — and where $c_m = \langle p, s \rangle$ with $p \in \mathcal{F}$. Furthermore, for a set of configurations C, we define

$$\mathrm{Pre}^*_{\mathcal{P}}(C)$$

$$q_f \overset{[_2}{—} q_{13} \overset{[_1}{—} q_{12} \overset{\gamma}{—} q_{11} \overset{\sigma}{—} q_{10} \overset{]_1}{—} q_9 \overset{[_1}{—} q_8 \overset{\sigma}{—} q_7 \overset{]_1}{—} q_6 \overset{]_2}{—} q_5 \overset{[_2}{—} q_4 \overset{[_1}{—} q_3 \overset{\sigma}{—} q_2 \overset{]_1}{—} q_1 \overset{]_2}{—} q_{\text{in}}$$

Figure 1: A run over $\left[[[\gamma\ \sigma]_1\ [\sigma]_1]_2\ [[\sigma]_1]_2\right]_3$

to be the set of configurations c such that there is a run over some word from c to $c' \in C$. When C is defined as the language of some automaton A accepting configurations, we abuse notation and write $\text{Pre}^*_P(A)$ instead of $\text{Pre}^*_P(\mathcal{L}(A))$.

For convenience, we sometimes allow a set of characters to be output instead of only one. This is to be interpreted as outputing each of the characters in the set once (in some arbitrary order). We also allow sequences of operations $o_1; \ldots; o_m$ in the rules instead of single operations. When using sequences we allow a test operation $\gamma?$ that only allows the sequence to proceed if the top_1 character of the stack is γ. All of these extensions can be encoded by introducing intermediate control states.

2.3.1 Regular Sets of Stacks

We will need to represent sets of stacks. To do this we will use automata to recognise stacks. We define the stack automaton model of Broadbent *et al.* [8] restricted to HOPDA rather than CPDA. We will sometimes call these *bottom-up stack automata* or simply *automata*. The automata operate over stacks interpreted as words, hence the opening and closing braces of the stacks appear as part of the input. We annotate these braces with the order of the stack the braces belong to. Let $\Gamma_{[]} = \{[_{n-1}, \ldots, [_1,]_1, \ldots,]_{n-1}\} \uplus \Gamma$. Note, we don't include $[_n,]_n$ since these appear exclusively at the start and end of the stack.

Definition 2.3 (Bottom-up Stack Automata)**.** *A bottom-up stack automaton A is a tuple $\left(\mathcal{Q}, \Gamma_{[]}, q_{in}, \mathcal{Q}_F, \Delta\right)$ where $\mathcal{Q}$ is a finite set of states, $\Gamma_{[]}$ is a finite input alphabet, $q_{in} \in \mathcal{Q}$ is the initial state and $\Delta : \left(\mathcal{Q} \times \Gamma_{[]}\right) \to \mathcal{Q}$ is a deterministic transition function.*

Representing higher order stacks as a linear word graph, where the start of an order-k stack is an edge labelled $[_k$ and the end of an order-k stack is an edge labelled $]_k$, a run of a bottom-up stack automaton is a labelling of the nodes of the graph with states in $\mathcal{Q}$ such that

1. the rightmost (final) node is labelled by q_{in}, and
2. whenever we have for any $\gamma \in \Gamma_{[]}$, and pair of labelled nodes with an edge $q \xrightarrow{\gamma} q'$ then $q = \Delta(q', \gamma)$.

The run is accepting if the leftmost (initial) node is labelled by $q \in \mathcal{Q}_F$. An example run over the word graph representation of $\left[[[\gamma\ \sigma]_1\ [\sigma]_1]_2\ [[\sigma]_1]_2\right]_3$ is given in Figure 1.

Let $\mathcal{L}(A)$ be the set of stacks with accepting runs of A. Sometimes, for convenience, if we have a configuration $c = \langle p, s\rangle$ of a HOPDA, we will write $c \in \mathcal{L}(A)$ when $s \in \mathcal{L}(A)$.

3 The Single Character Case

We assume $\Sigma = \{a, \varepsilon\}$ and use b to range over Σ. This can be obtained by simply replacing all other characters with ε. We also assume that all rules of the form $(p, \gamma) \xrightarrow{b} (p', o)$ with $o = \text{push}_n$ or $o = \text{pop}_n$ have $b = \varepsilon$. We can enforce this using intermediate control states to first apply o in one step, and then in another output b (the stack operation on the second step will be rew_γ where γ is the current top character). We start with an outline of the proof, and then explain each step in detail.

For convenience, we assume acceptance is by reaching a unique control state in $\mathcal{F}$ with an empty stack (i.e. the lowermost stack was removed with a pop_n and $\mathcal{F} = \{p_f\}$). This can easily be obtained by adding a rule to a new accepting state whenever we have a rule leading to a control state in $\mathcal{F}$. From this new state we can loop and perform pop_n operations until the stack is empty.

3.1 Outline of Proof

The approach is to take an n-PDA P and produce an $(n-1)$-PDA P_{-1} that satisfies the diagonal problem iff P does. The idea behind this reduction is that an (accepting) run of P can be decomposed into a tree with out-degree at most 2: each push_n has a matching pop_n that brings the stack back to be the same as it was before the push_n; we cut the run at the pop_n and hang the tail next to the push_n and repeat this to form a tree from a run. This is illustrated in Figure 2 where nodes are labelled by their configurations, and the push_n and pop_n points are marked. The dotted arcs connect nodes matched by their pushes and pops – these nodes have the same stacks. Notice that at each branching point, the left and right subtrees start with the same order-$(n-1)$ stacks on top. Notice also that for each branch, none of its transitions remove the topmost order-$(n-1)$ stack. Hence, we can produce an $(n-1)$-PDA that picks a branch of this tree decomposition to execute and only needs to keep track of the topmost order-$(n-1)$ stack of the n-PDA it is simulating. When picking a branch to execute, the $(n-1)$-PDA outputs a single a if the branch not chosen could have output a number of a characters. We prove that this is enough to maintain unboundedness.

In more detail, we perform the following steps.

1. Instrument P to record whether an a character has been output and then, using known reachability results, obtain regular sets of configurations from which the current top_n stack can be popped, and moreover, we can know whether an a is output on the way. Note, these tests can be seen as a generalisation of pushdown systems with regular tests introduced by Esparza *et al.* [12].

2. From an n-PDA P, we define an $(n-1)$-PDA with tests P_{-1} and then an $(n-1)$-PDA P' such that

$$\mathrm{Diagonal}_a(P) \iff \mathrm{Diagonal}_a(P') \ .$$

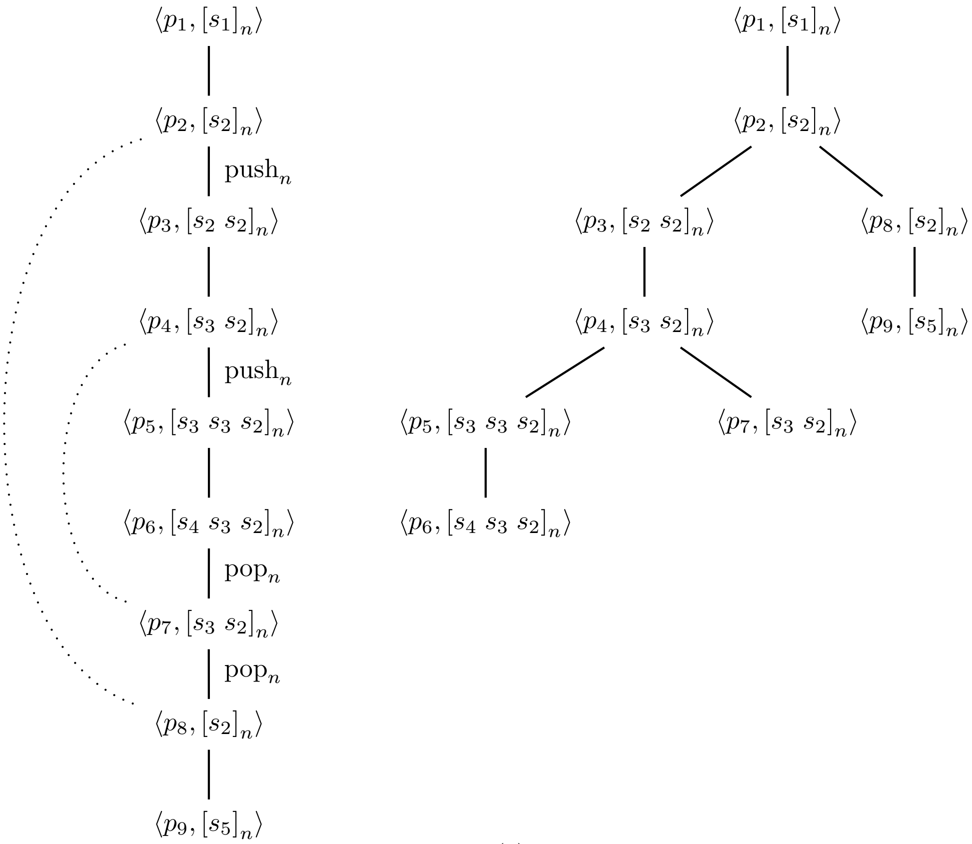
   The tests will be used to check the branches of the tree decomposition not explored by P_{-1}.

3. By repeated applications of the above reduction, we obtain an 1-PDA P for which $\mathrm{Diagonal}_a(P)$ is decidable since the downwards closure of a context-free grammar (equivalent to 1-PDA) is computable [39, 9] and this is equivalent to the diagonal problem.

The $(n-1)$-PDA with tests P_{-1} will simulate the n-PDA P in the following way.

- All operations except for push_n and pop_n will be simulated directly.

- In lieu of performing a push_n operation, P_{-1} will choose to simulate the run of P between the push and its corresponding pop_n, or the run of P after the corresponding pop_n has taken place.



(a) A run of P with push_ns and pop_ns marked.

(b) The tree decomposition of the run

Figure 2: Tree decompositions of runs.

- Tests will be used to determine which control state could appear after the corresponding pop_n.
- If the part of the run not being simulated output some as, then P will output a single a in place of the omitted as.

Although P_{-1} will output far fewer a characters than P (since it does not execute the full run), we show that it still outputs enough as for the language to remain unbounded.

We thus have the following theorem.

Theorem 3.1 (Decidability of the Diagonal Problem)**.** *For a given n-PDA P and output character a, whether $\text{Diagonal}_a(P)$ holds is decidable.*

Proof. We construct via Lemma 3.1 an $(n-1)$-PDA P' such that $\text{Diagonal}_a(P)$ iff $\text{Diagonal}_a(P')$. We repeat this step until we have a 1-PDA. It is known that $\text{Diagonal}_a(P)$ for an 1-PDA is decidable since it is possible to compute the downwards closure [39, 9]. ☐

3.2 HOPDA with Tests

When executing a branch of the tree decomposition, to be able to ensure the branch is correct and whether we should output an extra a we need to know how the system could have behaved on the skipped branch. To do this we add tests to the HOPDA that allow it to know if the current stack belongs to a given regular set. We show in the following sections that the properties required for our reduction can be represented as regular sets of stacks. Although we take Broadbent *et al.*'s logical reflection as the basis of our proof, HOPDA with tests can be seen as a generalisation of pushdown systems with regular valuations due to Esparza *et al.* [12].

Definition 3.1 (n-PDA with Tests)**.** *Given automata $A_1, \ldots, A_m$ recognising regular sets of stacks, a n-*PDA with tests *is a tuple $P = (\mathcal{P}, \Sigma, \Gamma, \mathcal{R}, \mathcal{F}, p_{in}, \gamma_{in})$ where $\mathcal{P}, \Sigma, \Gamma, \mathcal{F}, p_{in}$, and γ_{in} are as in HOPDA, and*

$$\mathcal{R} \subseteq \mathcal{P} \times \Gamma \times \{A_1, \ldots, A_m\} \times \Sigma \times Ops_n \times \mathcal{P}$$

is a set of transition rules.

We write $(p, \gamma, A_i) \xrightarrow{b} (p', o)$ for $(p, \gamma, A_i, b, o, p') \in \mathcal{R}$. We have a transition $\langle p, s\rangle \xrightarrow{b} \langle p', s'\rangle$ whenever $(p, \gamma, A_i) \xrightarrow{b} (p', o) \in \mathcal{R}$ and $top_1(s) = \gamma$, $s \in \mathcal{L}(A_i)$, and $s' = o(s)$.

We know from Broadbent *et al.* that these tests do not add any extra power to HOPDA. Intuitively, we can embed runs of the automata into the stack during runs of the HOPDA.

Theorem 3.2 (Removing Tests)**.** *[8, Theorem 3 (adapted)] For every n-PDA with tests P, there exists an n-PDA P' with*

$$\mathcal{L}(P) = \mathcal{L}(P') \ .$$

Proof. This is a straightforward adaptation of Broadbent *et al.* [8]. A more general theorem is proved in Theorem 6.1. □

3.2.1 Marking Outputs

When the HOPDA is in a configuration of the form $\langle p, [s]_n\rangle$ – i.e. the outermost stack contains only a single order-$(n-1)$ stack – we require the HOPDA to be able to know whether,

- for a given p_1 and p_2, there is a run from $\langle p_1, [s]_n\rangle$ to $\langle p_2, []_n\rangle$ (that is, the HOPDA empties the stack), and
- whether, during the run, an a is output.

Given P, we first augment P to record whether an a has been produced. This can be done simply by recording in the control state whether a has been output.

Definition 3.2 (P_γ)**.** *Given $P = (\mathcal{P}, \Sigma, \Gamma, \mathcal{R}, \mathcal{F}, p_{in}, \gamma_{in})$ we define*

$$P_\gamma = (\mathcal{P} \cup \mathcal{P}_\gamma, \Sigma, \Gamma, \mathcal{R} \cup \mathcal{R}_\gamma, \mathcal{F} \cup \mathcal{F}_\gamma, p_{in}, \gamma_{in})$$

where

$$\begin{array}{rcl} \mathcal{P}_\gamma & = & \{p_\gamma \mid p \in \mathcal{P}\} \\ \mathcal{R}_\gamma & = & \left\{(p_\gamma, \gamma) \xrightarrow{b} (p'_\gamma, o) \;\middle|\; (p, \gamma) \xrightarrow{b} (p', o) \in \mathcal{R}\right\} \cup \\ & & \left\{(p, \gamma) \xrightarrow{a} (p'_\gamma, o) \;\middle|\; (p, \gamma) \xrightarrow{a} (p', o) \in \mathcal{R}\right\} \\ \mathcal{F}_\gamma & = & \{p_\gamma \mid p \in \mathcal{F}\} \end{array}$$

It is easy to see that P and P_γ accept the same languages, and that P_γ is only in a control state p_γ if an a has been output.

3.2.2 Building the Automata

Fix some $P = (\mathcal{P}, \Sigma, \Gamma, \mathcal{R}, \mathcal{F})$ and $P_\gamma = (\mathcal{P}_\gamma, \Sigma, \Gamma, \mathcal{R}_\gamma, \mathcal{F}_\gamma)$. To obtain a HOPDA with tests, we need, for each $p_1, p_2 \in \mathcal{P}$ the following automata. Note, we define these automata to accept order-$(n-1)$ stacks since they will be used in an $(n-1)$-PDA with tests.

1. A_{p_1,p_2} accepting all stacks s such that there is a run of P from $\langle p_1, [s]_n \rangle$ to $\langle p_2, []_n \rangle$,
2. $A^a_{p_1,p_2}$ accepting all stacks s such that there is a run of P from $\langle p_1, [s]_n \rangle$ to $\langle p_2, []_n \rangle$ that outputs at least one a.

To do this we will use a reachability result due to Broadbent *et al.* that appeared in ICALP 2012 [7]. This result uses an automata representation of sets of configurations. However, these automata are slightly different in that they read full configurations "top down", whereas the automata of Theorem 3.2 (Removing Tests) read only stacks "bottom up".

It is known that these two representations are effectively equivalent, and that both form an effective boolean algebra [8, 7]. In particular, for a top-down automaton A and a control state p we can build a bottom-up stack automaton B such that $\langle p, s \rangle \in \mathcal{L}(A)$ iff $s \in \mathcal{L}(B)$ and vice versa. We recall the reachability result.

Theorem 3.3. *[7, Theorem 1 (specialised)] Given an HOPDA P and a top-down automaton A, we can construct an automaton A' accepting $Pre^*_P(A)$.*

Let $A_{p,\gamma}$ be a top-down automaton accepting configurations of the form $\langle p, [s]_n \rangle$ where $\mathrm{top}_1(s) = \gamma$. Next, let

$$A_p = \bigcup_{(p',\gamma) \xrightarrow{\varepsilon} (p, \mathrm{pop}_n) \in \mathcal{R}} A_{p',\gamma}$$

and

$$A^a_p = \left(\bigcup_{(p',\gamma) \xrightarrow{\varepsilon} (p, \mathrm{pop}_n) \in \mathcal{R}} A_{p'_\gamma,\gamma} \right)$$

I.e. A_p and A^a_p accept all configurations of P_γ from which it is possible to perform a pop_n operation to p and reach the empty stack.

Definition 3.3 (A_{p_1,p_2} and $A^a_{p_1,p_2}$). *Using the preceding notation, given p_1 and p_2 we define bottom-up automata*

- A_{p_1,p_2} *such that* $\mathcal{L}(A_{p_1,p_2}) = \{ s \mid \langle p_1, [s]_n \rangle \in Pre^*_P(A_{p_2}) \}$.
- $A^a_{p_1,p_2}$ *such that* $\mathcal{L}\big(A^a_{p_1,p_2}\big) = \left\{ s \;\middle|\; \langle p_1, [s]_n \rangle \in Pre^*_{P_\gamma}\big(A^a_{p_2}\big) \right\}$.

It is easy to see both A_{p_1,p_2} and $A^a_{p_1,p_2}$ are regular and representable by bottom-up automata since both

$$\mathrm{Pre}^*_P(A_{p_2}) \quad \text{and} \quad \mathrm{Pre}^*_{P_\gamma}\big(A^a_{p_2}\big)$$

are regular from Theorem 3.3, and bottom-up and top-down automata are effectively equivalent. To enforce only stacks of the form $[s]_n$ we can intersect with an automaton A_1 that accepts all stacks containing a single order-$(n-1)$ stack (this is clearly regular).

3.3 Reduction to Lower Orders

We are now ready to complete the reduction. Correctness is shown in Section 4. Let $A_{\texttt{tt}}$ be the automaton accepting all stacks. In the following definition, a control state (p_1, p_2) means that we are currently in control state p_1 and are aiming to empty the stack on reaching p_2, and the rules $\mathcal{R}_{\text{sim}}$ simulate all operations apart from push$_n$ and pop$_n$ directly, $\mathcal{R}_{\text{fin}}$ detect when the run is accepting, $\mathcal{R}_{\text{push}}$ follow the push branch of the tree decomposition, using tests to ensure the existence of the pop branch, and $\mathcal{R}_{\text{pop}}$ follow the pop branch of the tree decomposition, also using tests to check the existence of the push branch.

Definition 3.4 (P_{-1})**.** *Given an n-PDA P described by the tuple $(\mathcal{P}, \Sigma, \Gamma, \mathcal{R}, \{p_f\}, p_{in}, \gamma_{in})$ as well as families of automata $(A_{p_1,p_2})_{p_1,p_2 \in \mathcal{P}}$ and $\left(A^a_{p_1,p_2}\right)_{p_1,p_2 \in \mathcal{P}}$ we define an $(n-1)$-PDA with tests*

$$P_{-1} = (\mathcal{P}_{-1}, \Sigma, \Gamma, \mathcal{R}_{-1}, \mathcal{F}_{-1}, (p_{in}, p_f), \gamma_{in})$$

where

$$\begin{array}{rcl} \mathcal{P}_{-1} & = & \{(p_1, p_2) \mid p_1, p_2 \in \mathcal{P}\} \uplus \{f\} \\ \mathcal{R}_{-1} & = & \mathcal{R}_{sim} \cup \mathcal{R}_{fin} \cup \mathcal{R}_{push} \cup \mathcal{R}_{pop} \\ \mathcal{F}_{-1} & = & \{f\} \end{array}$$

and we define

- *$\mathcal{R}_{sim}$ is the smallest set containing all rules of the form*

  $$((p_1, p_2), \gamma, A_{\texttt{tt}}) \xrightarrow{b} ((p'_1, p_2), o)$$

  for all $(p_1, \gamma) \xrightarrow{b} (p'_1, o) \in \mathcal{R}$ with $o \notin \{push_n, pop_n\}$ and $p_2 \in \mathcal{P}$, and

- *$\mathcal{R}_{fin}$ is the smallest set containing all rules of the form*

  $$((p_1, p_2), \gamma, A_{\texttt{tt}}) \xrightarrow{\varepsilon} (f, rew_\gamma)$$

  for all $(p_1, \gamma) \xrightarrow{\varepsilon} (p_2, pop_n) \in \mathcal{R}$, and

- *$\mathcal{R}_{push}$ is the smallest set of rules containing all rules of the form*

  $$((p_1, p_2), \gamma, A_{p,p_2}) \xrightarrow{\varepsilon} ((p'_1, p), rew_\gamma)$$

  for all $(p_1, \gamma) \xrightarrow{\varepsilon} (p'_1, push_n) \in \mathcal{R}$ and $p, p_2 \in \mathcal{P}$, and all rules of the form

  $$\left((p_1, p_2), \gamma, A^a_{p,p_2}\right) \xrightarrow{a} ((p'_1, p), rew_\gamma)$$

  for all $(p_1, \gamma) \xrightarrow{\varepsilon} (p'_1, push_n) \in \mathcal{R}$ and $p, p_2 \in \mathcal{P}$, and

- *$\mathcal{R}_{pop}$ is the smallest set containing all rules of the form*

  $$\left((p_1, p_2), \gamma, A_{p'_1,p}\right) \xrightarrow{\varepsilon} ((p, p_2), rew_\gamma)$$

  for all $(p_1, \gamma) \xrightarrow{\varepsilon} (p'_1, push_n) \in \mathcal{R}$ and $p, p_2 \in \mathcal{P}$ and all rules of the form

  $$\left((p_1, p_2), \gamma, A^a_{p'_1,p}\right) \xrightarrow{a} ((p, p_2), rew_\gamma)$$

  for all $(p_1, \gamma) \xrightarrow{\varepsilon} (p'_1, push_n) \in \mathcal{R}$ and $p, p_2 \in \mathcal{P}$.

To complete the reduction, we convert the HOPDA with tests into a HOPDA without tests.

Lemma 3.1 (Reduction to Lower Orders)**.** *For every n-PDA P we can construct an $(n-1)$-PDA P' such that*

$$Diagonal_a(P) \iff Diagonal_a(P') \ .$$

Proof. From Definition 3.4 (P_{-1}) and Lemma 4.1 (below), we obtain from P an $(n-1)$-PDA with tests P_{-1} satisfying the conditions of the lemma. To complete the proof, we invoke Theorem 3.2 (Removing Tests) to find P' as required. □

4 Correctness of Reduction

This section is dedicated to the proof of the following lemma.

Lemma 4.1 (Correctness of P_{-1})**.**

$$Diagonal_a(P) \iff Diagonal_a(P_{-1})$$

The idea of the proof is that each run of P can be decomposed into a tree: each push_n operation creates a node whose left child is the run up to the matching pop_n, and whose right child is the run after the matching pop_n. All other operations create a node with a single child which is the successor configuration.

We then observe that each branch of such a tree corresponds to a run of P_{-1}. To prove that P_{-1} can output an unbounded number of as we prove that any tree containing m edges outputting a must have a branch along which P_{-1} would output $\log(m)$ a characters. Thus, if P can output an unbounded number of a characters, so can P_{-1}.

4.1 Tree Decomposition of Runs

Given a run

$$\rho = c_0 \xrightarrow{b_1} c_1 \xrightarrow{b_2} \cdots \xrightarrow{b_m} c_m$$

of P where each push_n operation has a matching pop_n, we can construct a tree representation of the run inductively as follows. We define $\text{Tree}(c) = T[\varepsilon]$ for the single-configuration run c, and, when

$$\rho = c \xrightarrow{b} \rho'$$

where the first rule applied does not contain a push_n operation, we have

$$\text{Tree}(\rho) = b[\text{Tree}(\rho')]$$

and, when

$$\rho = c_0 \xrightarrow{\varepsilon} \rho_1 \xrightarrow{\varepsilon} \rho_2$$

with c_1 being the first configuration of ρ_2 and where the first rule applied in ρ contains a push_n operation, $c_0 = \langle p, s \rangle$ and $c_1 = \langle p', s \rangle$ for some p, p', s and there is no configuration in ρ_1 of the form $\langle p'', s \rangle$, then

$$\text{Tree}(\rho) = \varepsilon[\text{Tree}(\rho_1), \text{Tree}(\rho_2)] \ .$$

An accepting run of P has the form $\rho \xrightarrow{\varepsilon} c$ where ρ has the property that all push_n operations have a matching pop_n and the final transition is a pop_n operation to $c = \langle p, []_n \rangle$ for some $p \in \mathcal{F}$. Hence, we define the tree decomposition of an accepting run to be

$$\text{Tree}\Big(\rho \xrightarrow{\varepsilon} c\Big) = \varepsilon[\text{Tree}(\rho), T[\varepsilon]] \ .$$

4.2 Scoring Trees

In the above tree decomposition of runs, the tree branches at each instance of a push_n operation. This mimics the behaviour of P_{-1}, which performs such branching non-deterministically. Hence, given a run ρ of P, each branch of $\text{Tree}(\rho)$ corresponds to a run of P_{-1}.

We formalise this intuition in the following section. In this section, we assign scores to each subtree T of $\text{Tree}(\rho)$. These scores correspond directly to the largest number of a characters that P_{-1} can output while simulating a branch of T.

Note, in the following definition, we exploit the fact that only nodes with exactly one child may have a label other than ε. We also give a general definition applicable to trees with out-degree larger than 2. This is needed in the simultaneous unboundedness section. For the moment, we only have trees with out-degree at most 2.

Let

$$\overline{b} = \begin{cases} 0 & b = \varepsilon \\ 1 & b = a \end{cases} \quad \text{and} \quad \overline{m} = \begin{cases} 0 & m = 0 \\ 1 & m > 0 \end{cases} .$$

Then,

$$\text{Score}(T) = \begin{cases} 0 & T = T[\varepsilon] \\ \text{Score}(T_1) + \overline{b} & T = b[T_1] \\ \max\limits_{1 \leq i \leq m} \left(\text{Score}(T_i) + \overline{\sum\limits_{j \neq i} \text{Score}(T_j)} \right) & T = \varepsilon[T_1, \dots, T_m] \end{cases}$$

We then have the following lemma for trees with out-degree 2.

Lemma 4.2 (Minimum Scores). *Given a tree T containing m nodes labelled a, we have*

$$\mathit{Score}(T) \geq \log(m)$$

Proof. The proof is by induction over m. In the base case $m = 1$ and there is a single node η in T labelled a. By definition, the subtree T' rooted at η has $\text{Score}(T') = 1$. Since the score of a tree is bounded from below by the score of any of its subtrees, we have $\text{Score}(T) \geq \log(1)$ as required.

Now, assume $m > 1$. Find the smallest subtree T' of T containing m nodes labelled a. We necessarily have either

1. $T' = a[T_1]$, or
2. $T' = \varepsilon[T_1, T_2]$ where T_1 and T_2 each have at least one node each labelled a.

In case (1) we have by induction

$$\text{Score}(T') = 1 + \log(m - 1) \geq \log(m)$$

In case (2) we have

$$\text{Score}(T') = \max \begin{pmatrix} \text{Score}(T_1) + \overline{\text{Score}(T_2)}, \\ \text{Score}(T_2) + \overline{\text{Score}(T_1)} \end{pmatrix} .$$

We pick whichever of T_1 and T_2 has the most nodes labelled a. This tree has at least $\lceil m/2 \rceil$ nodes labelled a. Note, since both trees contain nodes labelled a, the right-hand side of the addition is always 1. Hence, we need to show

$$\log(\lceil m/2 \rceil) + 1 \geq \log(m)$$

which follows from

$$\begin{aligned}\log(m) - \log(\lceil m/2 \rceil) = \log\left(\frac{m}{\lceil m/2 \rceil}\right) \\ \leq \\ \log\left(\frac{m}{m/2}\right) = \log(2) = 1 \ .\end{aligned}$$

By our choice of T' we thus have $\mathrm{Score}(T) = \mathrm{Score}(T') \geq \log(m)$ as required. ☐

4.3 From Branches to Runs

Lemma 4.3 (Scores to Runs)**.** *Given an accepting run ρ of P, if $Score(Tree(\rho)) = m$ then $a^m \in \mathcal{L}(P_{-1})$.*

Proof. Let p_f be the final (accepting) control state of P and let $T = \mathrm{Tree}(\rho)$. We begin at the root node of T, which corresponds to the initial configuration of ρ. Let $\langle p, [s]_n \rangle$ be this initial configuration and let $\langle (p, p_f), s \rangle$ be the initial configuration of P_{-1}.

Thus, assume we have a node η of T, with a corresponding configuration $c = \langle p, s \rangle$ of P and configuration $c_{-1} = \langle (p, p_{\mathrm{pop}}), \mathrm{top}_n(s) \rangle$ of P_{-1} and a run ρ_{-1} of P_{-1} ending in c_{-1} and outputting $(m - \mathrm{Score}(T'))$ a characters where T' is the subtree of T rooted at η. The subtree T' corresponds to a sub-run ρ' of ρ where the transition immediately following ρ' is a pop_n transition to a control state p_{pop}.

There are two cases when we are dealing with internal nodes.

- $T' = b[T_1]$.

  In this case there is a transition $c \xrightarrow{b} c'$ via a rule $(p, \gamma) \xrightarrow{b} (p', o)$ where $o \notin \{\mathrm{push}_n, \mathrm{pop}_n\}$. Hence, we have the rule $((p, p_{\mathrm{pop}}), \gamma, A_{\mathtt{tt}}) \xrightarrow{b} ((p', p_{\mathrm{pop}}), o)$ in P_{-1} and thus we can extend ρ_{-1} with a transition $c_{-1} \xrightarrow{b} c'_{-1}$ via this rule where ρ_{-1}, c' and c'_{-1} maintain the assumptions above.

- $T' = \varepsilon[T_1, T_2]$.

  In this case we have that T' corresponds to a sub-run

  $$c \xrightarrow{\varepsilon} \rho_1 \xrightarrow{\varepsilon} \rho_2$$

  of ρ. The transition from c to the beginning of ρ_1 is via a rule $r_1 = (p, \gamma) \xrightarrow{\varepsilon} (p_1, \mathrm{push}_n)$ and the transition from the end of ρ_1 to the start of ρ_2 is via a rule $r_2 = (p_2, \gamma_1) \xrightarrow{\varepsilon} (p_3, \mathrm{pop}_n)$. Moreover, from the definition of the decomposition, the final configuration in ρ_2 is followed in ρ by a pop rule $r_3 = (p_4, \gamma_2) \xrightarrow{\varepsilon} (p_{\mathrm{pop}}, \mathrm{pop}_n)$.

  There are two further cases depending on whether the score of T' is derived from the score of T_1 or T_2.

  – In the case of T_1, then, first observe that ρ_2 followed by an application of r_3 is a run from $\langle p_3, s \rangle$ to $\langle p_{\mathrm{pop}}, \mathrm{pop}_n(s) \rangle$ where the stack $\mathrm{pop}_n(s)$ does not appear in ρ_2. Thus, there is a run of P from $\langle p_3, [\mathrm{top}_n(s)]_n \rangle$ to $\langle p_{\mathrm{pop}}, []_n \rangle$ and moreover, this run outputs an a whenever the original run does. Hence, there is also a corresponding run of P from which outputs an a whenever the original run does.

    If an a is output, we have $c_{-1} \in \mathcal{L}\left(A^a_{p_3, p_{\mathrm{pop}}}\right)$ and $\mathrm{Score}(T') - \mathrm{Score}(T_1) = 1$. We can extend ρ via an application of the rule $\left((p, p_{\mathrm{pop}}), \gamma, A^a_{p_3, p_{\mathrm{pop}}}\right) \xrightarrow{a} ((p_1, p_3), \mathrm{rew}_\gamma)$ that exists in P_{-1} since $c_{-1} \in \mathcal{L}\Big(A^a_{p_3,p_{\mathrm{pop}}}\Big)$. This transition maintains the property on the stacks since the push_n copies the topmost stack, hence P_{-1} does not need to change its stack. It maintains the property on the scores since it outputs a, accounting for the part of the score contributed by T_2. Finally, the condition on control states is satisfied since the second component is set to p_2.

If an a is not output, then the case is similar to the above, except T_2 does not contribute to the score, we have $c_{-1} \in \mathcal{L}(A_{p_3,p_{\mathrm{pop}}})$, and the transition of P_{-1} is labelled ε instead of a.

– The case of T_2 is almost symmetric to T_1. Observe that ρ_1 followed by an application of r_2 is a run from $\langle p_1, \mathrm{push}_n(s)\rangle$ to $\langle p_3, s\rangle$ where the stack s does not appear in ρ_1. Thus, there is a run of P from $\langle p_1, [\mathrm{top}_n(s)]_n\rangle$ to $\langle p_3, []_n\rangle$ and moreover, this run outputs an a whenever the original run does. Hence, there is also a corresponding run of P from which outputs an a whenever the original run does.

If an a is output, we have $c_{-1} \in \mathcal{L}(A^a_{p_1,p_3})$ and $\mathrm{Score}(T') - \mathrm{Score}(T_2) = 1$. We can extend ρ via an application of the rule $\big((p, p_{\mathrm{pop}}), \gamma, A^a_{p_1,p_3}\big) \xrightarrow{a} ((p_3, p_{\mathrm{pop}}), \mathrm{rew}_\gamma)$ that exists in P_{-1} since $c_{-1} \in \mathcal{L}(A^a_{p_1,p_3})$ This transition maintains the property on the stacks since the stack after the pop_n is identical to the stack before the push_n, hence P_{-1} does not need to change its stack. It maintains the property on the scores since it outputs a, accounting for the part of the score contributed by T_1. Finally, the condition on control states is satisfied since the second component is unchanged.

If an a is not output, then the case is similar to the above, except T_1 does not contribute to the score, we have $c_{-1} \in \mathcal{L}(A_{p_1,p_3})$ and the transition of P_{-1} is labelled ε instead of a.

Finally, we reach a leaf node η with a run outputting the required number of as. We need to show that the run constructed is accepting. Let η' be the first ancestor of η that contains η in its leftmost subtree. Let T' be the subtree rooted at η'. This tree corresponds to a sub-run ρ' of ρ that is followed immediately by a pop_n rule $(p, \gamma) \xrightarrow{\varepsilon} (p_{\mathrm{pop}}, \mathrm{pop}_n)$. Moreover, we have $((p, p_{\mathrm{pop}}), \gamma, A_{\mathtt{tt}}) \xrightarrow{\varepsilon} (f, \mathrm{rew}_\gamma)$ with which we can complete the run of P_{-1} as required. ☐

4.4 The Other Direction

Finally, we need to show that each accepting run of P_{-1} gives rise to an accepting run of P containing at least as many as.

Lemma 4.4 (P_{-1} to P)**.** *We have Diagonal$_a(P_{-1})$ implies Diagonal$_a(P)$.*

Proof. Let p_f be the unique accepting conrol state of P. Take an accepting run ρ_{-1} of P_{-1}. We show that there exists a corresponding run ρ of P outputting at least as many as.

Let

$$c_0 \xrightarrow{b} \cdots \xrightarrow{b} c_m \xrightarrow{\varepsilon} \langle f, s\rangle$$

for some s be the accepting run of P_{-1}. We define inductively for each $0 \leq i \leq m$ a pair of runs ρ^i_1, ρ^i_2 of P such that

1. ρ^i_2 ends in a configuration $\langle p_f, []_n\rangle$ (i.e. is accepting), and
2. if $c_i = \langle (p, p_{\mathrm{pop}}), s\rangle$ then
   (a) the final configuration of ρ^i_1 is $\langle p, [ss_1 \ldots s_l]_n\rangle$, for some $s_1, \ldots, s_l$, and

   (b) the first configuration of ρ_2^i is $\langle p_{\text{pop}}, [s_1 \dots s_l]_n \rangle$, and

3. the sum of the number of a characters output by ρ_1^i and ρ_2^i is at least the number of a characters output by $c_0 \xrightarrow{b_1} \cdots \xrightarrow{b_i} c_i$.

Initially we have $c_0 = \langle (p_{\text{in}}, p_f), s \rangle$ and $s = [\![\gamma_{\text{in}}]\!]_{n-1}$. We define $\rho_1^0 = \langle p_{\text{in}}, [s]_n \rangle$ and $\rho_2^0 = \langle p_f, []_n \rangle$ which immediately satisfy the required conditions.

Assume we have ρ_1^i and ρ_2^i as required. We show how to obtain ρ_1^{i+1} and ρ_2^{i+1}. There are several cases depending on the rule used on the transition $c_i \xrightarrow{b_{i+1}} c_{i+1}$. Let $c_i = \langle (p, p_{\text{pop}}), s \rangle$, the final configuration of ρ_1^i be $\langle p, [ss_1 \dots s_l]_n \rangle$ and the first configuration of ρ_2^i be $\langle p_{\text{pop}}, [s_1 \dots s_l]_n \rangle$.

- If the rule was $((p, p_{\text{pop}}), \gamma, A_{\texttt{tt}}) \xrightarrow{b} ((p', p_{\text{pop}}), o)$ with $o \notin \text{push}_n, \text{pop}_n$ then we have $(p, \gamma) \xrightarrow{b} (p', o) \in \mathcal{R}$ and we define ρ_1^{i+1} to be ρ_1^i extended by an application of this rule. We also define $\rho_2^{i+1} = \rho_2^i$.

  The required conditions are inherited from ρ_1^i and ρ_2^i since o only changes the top_n stack, the final configuration of ρ_2^{i+1} is the same as ρ_2^i, p_{pop} is not changed, and the rule of P outputs an a iff the rule of P_{-1} does.

- If the rule was $\left((p, p_{\text{pop}}), \gamma, A_{p'_{\text{pop}}, p_{\text{pop}}}\right) \xrightarrow{\varepsilon} \left((p', p'_{\text{pop}}), \text{rew}_\gamma\right)$ then we have a rule $r = (p, \gamma) \xrightarrow{\varepsilon} (p', \text{push}_n) \in \mathcal{R}$. Moreover, from the test $A_{p'_{\text{pop}}, p_{\text{pop}}}$ we know there is a run of P from $\langle p'_{\text{pop}}, [s]_n \rangle$ to $\langle p_{\text{pop}}, []_n \rangle$ and hence there is also a run ρ from $\langle p'_{\text{pop}}, [ss_1 \dots s_l]_n \rangle$ to $\langle p_{\text{pop}}, [s_1 \dots s_l]_n \rangle$. We set $\rho_2^{i+1} = \rho\rho_2^i$ and ρ_1^{i+1} to be ρ_1^i extended by an application of r.

  Since the final configuration of ρ_1^{i+1} is $\langle p', [sss_1 \dots s_l]_n \rangle$ it is easy to check the required correspondence with the first configuration $\langle p'_{\text{pop}}, [ss_1 \dots s_l]_n \rangle$ of ρ_2^{i+1}.

  The remaining conditions are immediate since no a is output and the final configuration of ρ_2^{i+1} is the same as ρ_2^i.

- The case of $\left((p, p_{\text{pop}}), \gamma, A^a_{p'_{\text{pop}}, p_{\text{pop}}}\right) \xrightarrow{a} \left((p', p'_{\text{pop}}), \text{rew}_\gamma\right)$ is almost identical to the previous case. To adapt the proof, one needs only observe that since $c_i \in \mathcal{L}\left(A^a_{p'_{\text{pop}}, p_{\text{pop}}}\right)$ the run ρ used to extend ρ_2^i also outputs at least one a character.

- If the rule was $((p, p_{\text{pop}}), \gamma, A_{p_1, p_2}) \xrightarrow{\varepsilon} ((p_2, p_{\text{pop}}), \text{rew}_\gamma)$ then there is also a rule $r = (p, \gamma) \xrightarrow{\varepsilon} (p_1, \text{push}_n) \in \mathcal{R}$ and from the test A_{p_1, p_2} we know there is a run of P from $\langle p_1, [s]_n \rangle$ to $\langle p_2, []_n \rangle$ and therefore there is also a run ρ that goes from $\langle p_1, [sss_1 \dots s_l]_n \rangle$ to $\langle p_2, [ss_1 \dots s_l]_n \rangle$. We set ρ_1^{i+1} to be ρ_1^i extended with an application of r and then the run ρ. We also set $\rho_2^{i+1} = \rho_2^i$.

  To verify that the properties hold, we observe that $c_{i+1} = \langle (p_2, p_{pop}), s \rangle$, and ρ_1^{i+1} ends with $\langle p_2, [ss_1 \dots s_l]_n \rangle$ and ρ_2^{i+1} still begins with $\langle p_{\text{pop}}, [s_1 \dots s_l]_n \rangle$ and has the required final configuration. The property on the number of as holds since the rule of P_{-1} did not output an a.

- The case of $\left((p, p_{\text{pop}}), \gamma, A^a_{p_1, p_2}\right) \xrightarrow{a} ((p_2, p_{\text{pop}}), \text{rew}_\gamma)$ is almost identical to the previous case. To adapt the proof, one needs only observe that since $c_i \in \mathcal{L}\left(A^a_{p_1, p_2}\right)$ the run ρ used to extend ρ_1^i also outputs at least one a character.

Finally, when we reach $i = m$ we have from the final transition of the run of P_{-1} that there is a rule $(p, \gamma) \xrightarrow{\varepsilon} (p_{\text{pop}}, \text{pop}_n)$. We combine ρ_1^m and ρ_2^m with this pop transition, resulting in an accepting run of P that outputs at least as many a characters as the run of P_{-1}. $\square$

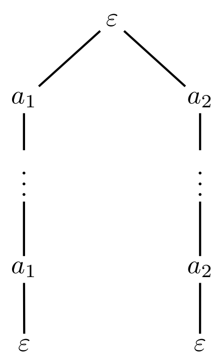


Figure 3: An example showing that following a single branch does not work for simultaneous unboundedness.

5 Multiple Characters

We generalise the previous result to the full diagonal problem. Naïvely, the previous approach cannot work. Consider the HOPDA executing

$$\text{push}_1^m; \text{push}_n; \text{pop}_1^m; \text{pop}_n; \text{pop}_1^m$$

where the first sequence of pop_1 operations output a_1 and the second sequence output a_2.

The corresponding run trees are of the form given in Figure 3. In particular, P_{-1} can only choose one branch, hence all runs of P_{-1} produce a bounded number of a_1s or a bounded number of a_2s. They cannot be simultaneously unbounded.

For P_{-1} to be able to output both an unbounded number of a_1 and a_2 characters, it must be able to output two branches of the tree. To this end, we define a notion of α-branch HOPDA, which output trees with up to α branches. We then show that the reduction used for reducing n-PDA to $(n-1)$-PDA can be generalised to α-branch HOPDA.

5.1 Branching HOPDA

We define n-PDA outputting trees with at most α branches, denoted (n, α)-PDA. Note, an n-PDA that outputs a word is an $(n, 1)$-PDA. Indeed, any (n, α)-PDA is also an (n, α')-PDA whenever $\alpha \leq \alpha'$.

Definition 5.1 ((n, α)-PDA)**.** *We define an* order-n α-branch pushdown automaton ((n, α)-PDA) *to be given by a tuple $P = (\mathcal{P}, \Sigma, \Gamma, \mathcal{R}, \mathcal{F}, p_{in}, \gamma_{in}, \theta)$ where $\mathcal{P}$, Σ, Γ, $\mathcal{F}$, p_{in}, and γ_{in} are as in HOPDA. The set of rules $\mathcal{R} \subseteq \bigcup_{1 \leq m \leq \alpha} \mathcal{P} \times \Gamma \times \Sigma \times Ops_n \times \mathcal{P}^m$ together with a mapping $\theta : \mathcal{P} \to \{1, \ldots, \alpha\}$ such that for all $(p, \gamma, b, o, p_1, \ldots, p_m) \in \mathcal{R}$ we have $\theta(p) \geq \theta(p_1) + \cdots + \theta(p_m)$.*

We use the notation $(p, \gamma) \xrightarrow{b} (p_1, \ldots, p_m, o)$ to denote a rule $(p, \gamma, b, o, p_1, \ldots, p_m) \in \mathcal{R}$. Intuitively, such a rule generates a node of a tree with m children. The purpose of the mapping θ is to bound the number of branches that this tree may have. Hence, at each branching rule, the quota of branches is split between the different subtrees. The existence of such a mapping implies this information is implicit in the control states and an (n, α)-PDA can only output trees with at most α branches.

From the initial configuration $c_0 = \langle p_{\text{in}}, [\![\gamma_{\text{in}}]\!]_n \rangle$ a run of an (n, α)-PDA is a tree $T = (D, \lambda)$ whose nodes are labelled with n-PDA configurations, and generates an output tree $T' = (D, \lambda')$ whose nodes are labelled with symbols from the output alphabet. Precisely

- $\lambda(\varepsilon) = c_0$, and
- for a node η with children $\eta_1, \ldots, \eta_m$ and $\lambda(\eta) = \langle p, s \rangle$ there is a rule $(p, \gamma) \xrightarrow{b} (p_1, \ldots, p_m, o)$ such that for all $1 \leq i \leq m$ we have $\lambda(\eta_i) = \langle p_i, s' \rangle$ where $\text{top}_1(s) = \gamma$, $s' = o(s)$. Moreover we have $\lambda'(\eta) = b$.
- For all leaf nodes η we have $\lambda'(\eta) = \varepsilon$.

The run is accepting if for all leaf nodes η we have $\lambda(\eta) = \langle p, []_n \rangle$ and $p \in \mathcal{F}$. Let $\mathcal{L}(P)$ be the set of output trees of P.

Given an output tree T we write $|T|_a$ to denote the number of nodes labelled a in T. For a (n, α)-PDA P, we define

$$\text{Diagonal}_{a_1,\ldots,a_\alpha}(P) = \forall m.\exists T \in \mathcal{L}(P).\forall 1 \leq i \leq \alpha.|T|_{a_i} \geq m \ .$$

6 Reduction For Simultaneous Unboundedness

Given an (n, α)-PDA P we construct an $(n-1, \alpha)$-PDA P_{-1} such that

$$\text{Diagonal}_{a_1,\ldots,a_\alpha}(P) \iff \text{Diagonal}_{a_1,\ldots,a_\alpha}(P_{-1}) \ .$$

Moreover, we show $\text{Diagonal}_{a_1,\ldots,a_\alpha}(P)$ is decidable for a $(0, \alpha)$-PDA (i.e. a regular automaton outputting an α-branch tree) P.

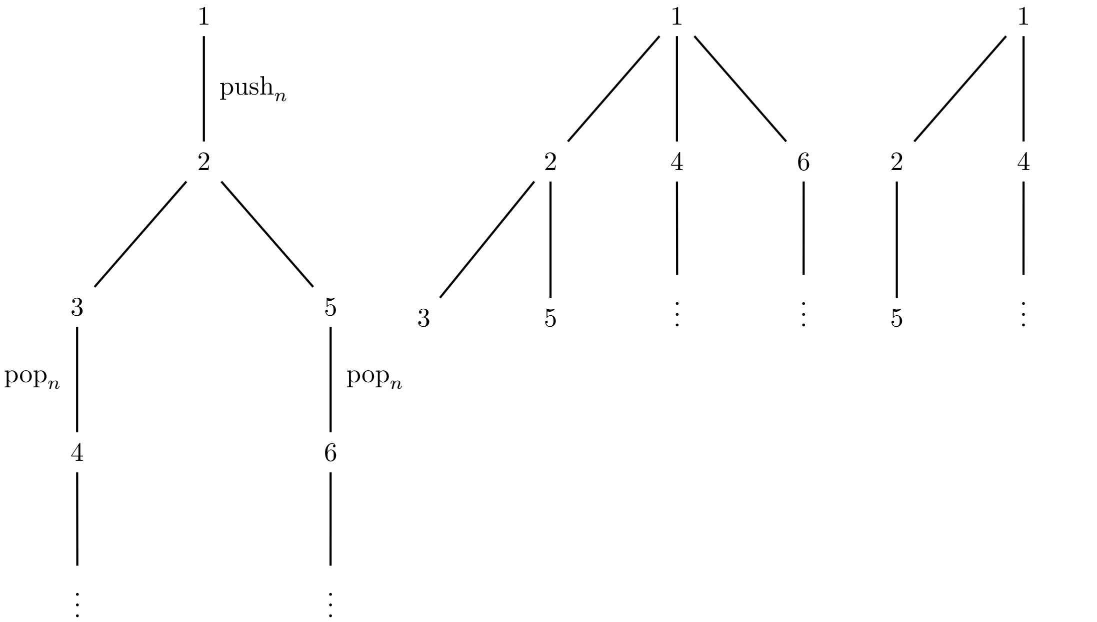
For simplicity, we assume for all rules $(p, \gamma) \xrightarrow{b} (p_1, \ldots, p_m, o)$ if $m > 1$ then $o = \text{rew}_\gamma$ (i.e. the stack is unchanged). Additionally we have $b = \varepsilon$.

We also make analogous assumptions to the single character case. That is, we assume $\Sigma = \{a_1, \ldots, a_\alpha, \varepsilon\}$ and use b to range over Σ. We also assume that all rules of the form $(p, \gamma) \xrightarrow{b} (p', o)$ with $o = \text{push}_n$ or $o = \text{pop}_n$ have $b = \varepsilon$. Finally, we assume acceptance is by reaching a unique control state in $\mathcal{F}$ with an empty stack.

6.1 Some Intuition

We briefly sketch the intuition behind the algorithm. We illustrate the reduction from (n, α)-PDA to $(n-1, \alpha)$-PDA in Figure 4.

- We begin with an n-PDA which we first interpret as an (n, α)-PDA. This is possible because an (n, α)-PDA can produce *at most* α branches. Thus, an (n, α)-PDA – which produces a single branch – is also a (n, α)-PDA. We work with HOPDA producing α branches because, after each reduction step, we will need to output one branch for each character in $a_1, \ldots, a_\alpha$.
- We have an (n, α)-PDA P that outputs a tree with at most α branches. In Figure 4 we show part of a run tree with 2 branches. The push_n and pop_n operations are shown on the edges of the tree. Nodes are numbered to help identify them during the different transformations.



(a) An α-branch run tree of P

(b) The decomposition of the run tree

(c) An α-branch subtree of the decomposition generated by P_{-1}.

Figure 4: Illustrating the reduction steps.

- We "decompose" this tree into another tree where the branches appearing after the pop_n operations are hung from the same parent as their matching push_n. This is shown in the middle of Figure 4. Notice that this tree has an unbounded number of branches (it branches at each push_n). However, we know that the maximum out-degree of any of its nodes is $(\alpha + 1)$ since the source of a push_n-labelled edge has one child, and we add at most α extra children corresponding to the pop_n on each of its at most α branches.

- We prove a generalisation of Lemma 4.2 (Minimum Scores) that shows a run tree with at least m instances of a character a has a branch with a score of at least $\log_{(\alpha+1)}(m)$. Thus, we need to select one branch for each a we wish to output.

- We build an $(n-1,\alpha)$-PDA P_{-1} that non-deterministically picks out the highest scoring branches for each a. This is shown on the right of Figure 4.

6.2 Branching HOPDA with Regular Tests

As before, we instrument our HOPDA with tests. Removing these tests requires a simple adaptation of Broadbent *et al.* [8].

Definition 6.1 ((n,α)-PDA with Tests)**.** *Given a set $A_1, \ldots, A_m$ of automata, an* (n,α)-PDA with tests *is given by a tuple $P = (\mathcal{P}, \Sigma, \Gamma, \mathcal{R}, \mathcal{F}, p_{in}, \gamma_{in}, \theta)$ where $\mathcal{P}$, Σ, Γ, $\mathcal{F}$, p_{in}, γ_{in} are as in HOPDA. The set of rules $\mathcal{R} \subseteq \bigcup_{1 \leq m \leq \alpha} \mathcal{P} \times \Gamma \times \{A_1, \ldots, A_m\} \times \Sigma \times Ops_n \times \mathcal{P}^m$ together* *with a mapping* $\theta : \mathcal{P} \to \{1, \ldots, \alpha\}$ *such that for all* $(p, \gamma, A, b, o, p_1, \ldots, p_m) \in \mathcal{R}$ *we have* $\theta(p) \geq \theta(p_1) + \cdots + \theta(p_m)$.

We use the notation $(p, \gamma, A) \xrightarrow{b} (p_1, \ldots, p_m, o)$ to denote a rule $(p, \gamma, A, b, o, p_1, \ldots, p_m) \in \mathcal{R}$.

From the initial configuration $c_0 = \langle p_{\text{in}}, [\![\gamma_{\text{in}}]\!]_n \rangle$ a run of an (n, α)-PDA with tests is a tree $T = (D, \lambda)$ and generates an output tree $\rho = (D, \lambda')$ where

- $\lambda(\varepsilon) = c_0$, and
- for a node η with children $\eta_1, \ldots, \eta_m$ and $\lambda(\eta) = \langle p, s \rangle$ there is a rule $(p, \gamma, A) \xrightarrow{b} (p_1, \ldots, p_m, o)$ such that $s \in \mathcal{L}(A)$ and for all $1 \leq i \leq m$ we have $\lambda(\eta_i) = \langle p_i, s' \rangle$ where $\text{top}_1(s) = \gamma$, and $s' = o(s)$. Moreover we have $\lambda'(\eta) = b$.
- For all leaf nodes η we have $\lambda'(\eta) = \varepsilon$.

The run is accepting if for all leaf nodes η we have $\lambda(\eta) = \langle p, []_n \rangle$ and $p \in \mathcal{F}$. Let $\mathcal{L}(P)$ be the set of output trees of P.

Theorem 6.1 (Removing Tests)**.** *[8, Theorem 3 (adapted)] For every* (n, α)*-PDA with tests* P*, there exists an* (n, α)*-PDA* P' *with*

$$\mathcal{L}(P) = \mathcal{L}(P') \ .$$

Proof. This is a straightforward adaptation of Broadbent *et al.* [8]. Let the (n, α)-PDA with tests be $P = (\mathcal{P}, \Sigma, \Gamma, \mathcal{R}, \mathcal{F}, p_{\text{in}}, \gamma_{\text{in}}, \theta)$ with test automata $A_1, \ldots, A_m$. We build an (n, α)-PDA that mimics P almost directly. The only difference is that each character γ appearing in the stack is replaced by

$$(\gamma, \overline{\tau}_1, \ldots, \overline{\tau}_m) \ .$$

For each test A we have a vector of functions

$$\overline{\tau} = (\tau_1, \ldots, \tau_n) \ .$$

The function $\tau_k : \mathcal{Q} \to \mathcal{Q}$ intuitively describes runs of A from the bottom of $\text{top}_{k+1}(s)$ to the top of $\text{pop}_k\big(\text{top}_{k+1}(s)\big)$. Thus, we can reconstruct an entire run over $\text{pop}_1(s)$ from initial state q as

$$q' = \tau_1(\cdots \tau_n(q))$$

and then we can consult Δ to complete the run by adding the effect of reading $\text{top}_1(s)$.

Thus, let $A_i = \big(\mathcal{Q}_i, \Gamma_{[]}, q^i_{\text{in}}, \Delta_i, \mathcal{Q}^i_F\big)$. We define

$$\hat{P} = \Big(\mathcal{P}, \Sigma, \hat{\Gamma}, \hat{\mathcal{R}}, \mathcal{F}, p_{\text{in}}, \hat{\gamma_{\text{in}}}, \theta\Big)$$

where

$$\hat{\Gamma} = \{(\gamma, \overline{\tau}_1, \ldots, \overline{\tau}_m) \ \mid \gamma \in \Gamma \wedge \forall i. \overline{\tau}_i \in (\mathcal{Q}_i \to \mathcal{Q}_i)^n \}$$

and $\hat{\mathcal{R}}$ is the smallest set of rules of the form

$$(p, \hat{\gamma}) \xrightarrow{b} (p_1, \ldots, p_l, \text{Update}(o, \hat{\gamma}))$$

where $\hat{\gamma} = (\gamma, \overline{\tau}_1, \ldots, \overline{\tau}_m)$ and $(p, \gamma, A_i) \xrightarrow{b} (p_1, \ldots, p_l, o) \in \mathcal{R}$ and $\text{Accepts}\big(\gamma, \overline{\tau}_i, \Delta_i, q^i_{\text{in}}, \mathcal{Q}^i_F\big)$ and we define

$$\text{Accepts}(\gamma, \tau_1, \ldots, \tau_n, \Delta, q_{\text{in}}, \mathcal{Q}_F)$$
$$\Longleftrightarrow$$
$$q = \tau_1(\cdots \tau_n(q_{\text{in}})) \wedge \Delta(q, [_n \cdots [_1 \gamma) \in \mathcal{Q}_F$$

where $\Delta(q, [_n \cdots [_1 \gamma)$ is shorthand for the repeated application of Δ on γ then $[_1$, back to $[_n$, and we define $\text{Update}(o, \hat{\gamma}) = \hat{o}$ following the cases below. Let $\hat{\gamma} = (\gamma, \overline{\tau}_1, \ldots, \overline{\tau}_m)$.

- When $o = \text{rew}_\sigma$ then $\hat{o} = (\sigma, \overline{\tau}_1, \ldots, \overline{\tau}_m)$.
- When $o = \text{push}_k$ then $\hat{o} = \text{push}_o; \text{rew}_{(\gamma, \overline{\tau}'_1, \ldots, \overline{\tau}'_m)}$ where for all i we have

  $$\overline{\tau}_i = (\tau_1, \ldots, \tau_{k-1}, \tau'_k, \tau_{k+1}, \ldots \tau_n)$$

  and

  $$\tau'_k(q) = \tau_k(\Delta_i(\tau_1(\cdots \tau_k(q)),]_{k-1}[_{k-1} \cdots [_1 \gamma)).$$

  I.e., we apply the functions to read the whole stack once, and then the correct part of the copy created by the push_k.
- When $o = \text{pop}_k$ then

  $$\hat{o} = \text{pop}_o; (\sigma, \overline{\tau}'_1, \ldots, \overline{\tau}'_m)?; \text{rew}_{(\sigma, \overline{\tau}''_1, \ldots, \overline{\tau}''_m)}$$

  where for all i we have $\overline{\tau}_i = (\tau_1, \ldots, \tau_n)$ and $\overline{\tau}'_i = (\tau'_1, \ldots, \tau'_n)$ and

  $$\overline{\tau}''_i = \left(\tau'_1, \ldots, \tau'_{k-1}, \tau_k, \ldots \tau_n\right) \ .$$

  We can see that this is correct since we do not update the functions that read parts of the stack unchanged (i.e., stacks outside of those changed by the pop_k), and we take the functions that are correct for the newly exposed top parts of the stack for the remaining functions.

Finally, we set $\hat{\gamma_{\text{in}}} = (\gamma_{\text{in}}, \overline{\tau}_1, \ldots, \overline{\tau}_m)$ where for each i we have $\overline{\tau}_i = (\tau_1, \ldots, \tau_n)$ such that for each k we have $\tau_k(q) = \Delta(q,]_k \cdots]_n)$. □

6.3 Building The Automata

Previously we built automata A_{p_1,p_2} to indicate that from p_1, the current top stack could be removed, arriving at p_2. This is fine for words, however, we now have α-branch trees. It is no longer enough to specify a single control state: the top stack may be popped once on each branch of the tree, hence we need the following automata

$$A^O_{p,p_1,\ldots,p_m}$$

where $\theta(p) \geq \theta(p_1) + \cdots + \theta(p_m)$ and $O \subseteq \{a_1, \ldots, a_\alpha\}$. We have $s \in \mathcal{L}\left(A^O_{p,p_1,\ldots,p_m}\right)$ iff there is a run tree T with the root labelled $\langle p, [s]_n \rangle$ and m leaf nodes labelled $\langle p_1, []_n \rangle, \ldots, \langle p_m, []_n \rangle$ respectively. Moreover, we have $a \in O$ iff the corresponding output tree T' has $|T'|_a > 0$.

6.3.1 Alternating HOPDA

To construct the required stack automata, we need to do reachability analysis of (n, α)-PDA. We show that such analyses can be rephrased in terms of alternating higher-order pushdown systems (HOPDS), for which the required algorithms are already known [7]. Note, we refer to these machines as "systems" rather than "automata" because they do not output a language.

Definition 6.2 (Alternating HOPDS). *An* alternating order-n pushdown system *is a tuple* $P = (\mathcal{P}, \Gamma, \mathcal{R})$ *where* $\mathcal{P}$ *is a finite set of control states,* Γ *is a finite stack alphabet, and*

$$\mathcal{R} \subseteq (\mathcal{P} \times \Gamma \times \mathit{Ops}_n \times \mathcal{P}) \cup \left(\mathcal{P} \times \Gamma \times 2^{\mathcal{P}}\right)$$

is a set of transition rules.

We write $(p, \gamma) \rightarrow (p, o)$ to denote $(p, \gamma, o, p) \in \mathcal{R}$ and $(p, \gamma) \rightarrow p_1, \ldots, p_m$ to denote $(p, \gamma, \{p_1, \ldots, p_m\}) \in \mathcal{R}$.

An run of an alternating HOPDS may split into several configurations, each of which must reach a target state. Hence, the branching of the alternating HOPDS mimics the branching of the (n, α)-PDA. Given a set C of configurations, we define $\mathrm{Pre}^*_P(C)$ to be the smallest set C' such that

$$\begin{array}{rcl} C' & = & C \cup \\ & & \left\{ \langle p, s \rangle \;\middle|\; \begin{array}{c} (p, \gamma) \rightarrow (p', o) \in \mathcal{R} \wedge \\ \mathrm{top}_1(s) = \gamma \wedge \\ \langle p', o(s) \rangle \in C' \end{array} \right\} \cup \\ & & \left\{ \langle p, s \rangle \;\middle|\; \begin{array}{c} (p, \gamma) \rightarrow p_1, \ldots, p_m \in \mathcal{R} \wedge \\ \mathrm{top}_1(s) = \gamma \wedge \\ \forall i. \langle p_i, s \rangle \in C' \end{array} \right\} . \end{array}$$

6.3.2 Constructing the Tests

In order to use standard results to obtain $A^O_{p,p_1,\ldots,p_m}$ we construct an alternating HOPDS $P_\diamond$ and automaton A such that

$$c \in \mathrm{Pre}^*_{P_\diamond}(A)$$

iff c satisfies the properties above.

The alternating HOPDS $P_\diamond$ will mimic the branching of P with alternating transitions[1] $(p, \gamma) \rightarrow p_1, \ldots, p_m$ of $P_\diamond$. It will maintain in its control states information about which characters have been output, as well as which control states should appear on the leaves of the branches. This final piece of information prevents all copies of the alternating HOPDS from verifying the same branch of P.

Definition 6.3 ($P_\diamond$)**.** *Given an (n, α)-PDA P described by the tuple $(\mathcal{P}, \Sigma, \Gamma, \mathcal{R}, \mathcal{F}, p_{in}, \gamma_{in})$, of P, we define*

$$P_\diamond = (\mathcal{P}_\diamond, \Gamma, \mathcal{R}_\diamond)$$

where

$$\mathcal{P}_\diamond = \left\{ (p, O, p_1, \ldots, p_m) \;\middle|\; \begin{array}{c} 1 \leq m \leq \alpha \wedge \\ O \subseteq \{a_1, \ldots, a_\alpha\} \wedge \\ p_1, \ldots, p_m \in \mathcal{P} \end{array} \right\}$$

and $\mathcal{R}_\diamond$ is the smallest set of rules containing, for each

$$(p, \gamma) \xrightarrow{b} (p', o) \in \mathcal{R}$$

all rules

$$((p, O, p_1, \ldots, p_i), \gamma) \rightarrow ((p_1, O \setminus \{b\}, p_1, \ldots, p_i), o)$$

and for each

$$(p, \gamma) \xrightarrow{\varepsilon} (p_1, \ldots, p_m, rew_\gamma) \in \mathcal{R}$$

with $m > 1$ all alternating rules

$$((p, O, p'_1, \ldots, p'_i), \gamma) \rightarrow \left(p_1, O_1, p^1_1, \ldots, p^1_{i_1}\right), \ldots \left(p_m, O_m, p^m_1, \ldots, p^m_{i_m}\right)$$

such that $p'_1, \ldots, p'_i$ is a permutation of $p^1_1, \ldots, p^1_{i_1}, \ldots p^m_1, \ldots, p^m_{i_m}$ and $O = O_1 \cup \cdots \cup O_m$.

[1] We slightly alter the alternation rule from ICALP 2012 [7] by matching the top stack character as well as the control state. This is a benign alteration since it is straightforward to track the top of stack character in the control state.

In the above definition, the permutation condition ensures that the states to be popped to are properly distributed amongest the newly created branches.

Lemma 6.1. *We have*

$$s \in \mathcal{L}\big(A^{O}_{p,p_1,\ldots,p_m}\big) \iff \langle (p, O, p_1, \ldots, p_m), [s]_n \rangle \in Pre^*_{P_\diamond}(A)$$

where A is such that

$$\mathcal{L}(A) = \{ \langle (p, \emptyset, p), []_n \rangle \mid p \in \{p_1, \ldots, p_m\} \} \ .$$

Proof. First take $s \in \mathcal{L}\big(A^{O}_{p,p_1,\ldots,p_m}\big)$ and the run tree witnessing this membership. We can move down the tree, maintaining a frontier $c_1, \ldots, c_l$ and building a tree witnessing that $\langle (p, O, p_1, \ldots, p_m), [s]_n \rangle \in \mathrm{Pre}^*_{P_\diamond}(A)$. Initially we have the frontier $\langle p, [s]_n \rangle$ and the initial configuration $\langle (p, O, p_1, \ldots, p_m), [s]_n \rangle$.

Hence, take a configuration $c = \langle p', s' \rangle$ from the frontier and corresponding configuration $c' = \langle (p', O', p'_1, \ldots, p'_i), s' \rangle$. If the rule applied to c is not a branching rule, we simply take the matching rule of $P_\diamond$ and apply it to c'. Note, that if the rule output b we remove b from O'. Hence, O' contains only characters that have not been output on the path from the initial configuration.

If the rule applied is branching, that is $(p', \gamma) \xrightarrow{\varepsilon} \big(p''_1, \ldots, p''_j, \mathrm{rew}_\gamma\big)$ then we apply the rule

$$((p', O, p'_1, \ldots, p'_i), \gamma) \to \big(p''_1, O_1, p^1_1, \ldots, p^1_{i_1}\big), \ldots \Big(p''_j, O_j, p^j_1, \ldots, p^j_{i_j}\Big)$$

where $p'_1, \ldots, p'_i$ is a permutation of $p^1_1, \ldots, p^1_{i_1}, \ldots p^j_1, \ldots, p^m_{i_j}$ and $O = O_1 \cup \cdots \cup O_m$. These partitions are made in accordance with the distribution of the leaves and outputs of the run tree of P. I.e. if a control state p'' appears on the i'th subtree, then it should appear in the i'th target state of $P_\diamond$. Similarly, if the i'th subtree outputs an $b \in O$, then b should be placed in $O_{i'}$. Applying this alternating transition creates a matching configuration for each new branch in the frontier.

We continue in this way until we reach the leaf nodes of the frontier. Each leaf $\langle p', s \rangle$ has a matching $\langle (p', \emptyset, p'), s \rangle$ and hence is in $\mathcal{L}(A)$. Thus, we have witnessed $\langle (p, O, p_1, \ldots, p_m), [s]_n \rangle \in \mathrm{Pre}^*_{P_\diamond}(A)$ as required.

To prove the other direction, we mirror the previous argument, showing that the witnessing tree for $P_\diamond$ can be used to build a run tree of P.

□

We can now build $A^{O}_{p,p_1,\ldots,p_m}$ from the control state p and top-down automaton representation of $\mathrm{Pre}^*_{P_\diamond}(A)$ since we can effectively translate from top-down automata to bottom-up stack automata.

6.4 Reduction to Lower Orders

We generalise our reduction to (n, α)-PDA. Let $A_{\mathtt{tt}}$ be the automata accepting all configurations. Note, in the following definition we allow all transitions (including branching) to be labelled by sets of output characters. To maintain our assumed normal form we have to replace these transitions using intermediate control states to ensure all branching transitions are labelled by ε and all transitions labelled O are replaced by a sequence of transitions outputting a single instance of each character in O.

The construction follows the intuition of the single character case, but with a lot more bookkeeping. A control state

$$(p, p_1, \ldots, p_m, O, B)$$

plays the role of a test $A^O_{p,p_1,\ldots,p_m}$ since all branches need to be tested, and if we are running the HOPDA along that branch, the HOPDA should perform the test to ensure that there is no conflict between the branch actually followed and the one used to pass the test. The final component B is a set of output characters such that we have $a \in B$ to indicate that the branch the HOPDA has assigned this control state is the branch responsible for outputting enough a characters. As a sanity condition we enforce $O \cap B = \emptyset$ since a branch outputting a should never run a test on itself.

We explain the rules intuitively. It will be beneficial to refer to the formal definition (below) while reading the explanations. The case for $\mathcal{R}_{\text{push}}$ is illustrated in Figure 5 since it covers most of the situations appearing in the other rules as well.

- The rules in $\mathcal{R}_{\text{init}}$ guess how many branches will be needed to output enough of each a. (This might be less than α since one branch might account for several characters.)

- The rules in $\mathcal{R}_{\text{fin}}$ check whether the run can be finished (always via a pop_n since we are aiming for the empty stack). This is true if we only have one branch to complete (just reach p') and we have no more characters that we're obliged to output.

- The rules in $\mathcal{R}_{\text{sim}}$ simulate a non-branching operation. They do this faithfully, simply passing along all other information (updating the O field if a character is output).

- The rules in $\mathcal{R}_{\text{br}}$ are the first of the complicated rules. This is mainly a matter of notation. The reasoning behind the rules is that we're at a point where the tree splits into l different branches. These have control states $p'_1, \ldots, p'_l$ respectively. We non-deterministically guess which of these branches should output which of the characters in B. Thus, we split B into $B_1, \ldots, B_i$. This means we are exploring i branches. Let $x_1, \ldots, x_i$ be the control states on these branches. The remaining branches we handle using tests on the stack. Let $y_1, \ldots, y_j$ be the control states appearing on these branches. We require that all of $p'_1, \ldots, p'_l$ are accounted for, so we assert that $p'_1, \ldots, p'_l$ is a permutation of $x_1, \ldots, x_i, y_1, \ldots, y_j$.

  Similarly, in the current subtree we are obliged to pop to leaf nodes containing the control states $p_1, \ldots, p_m$. Thus, we split these obligations between the branches we are exploring and the branches we are handling using tests. We use another permutation check to ensure that these obligations have been distributed properly.

  Finally, we are required to output characters in O. We may also, in choosing a particular branch for a character a, need to output a to account for instances appearing on a missed branch. Hence we also output O' to account for these. We distribute the obligations O and O' amongst the different branches using $X_1, \ldots, X_i$ and $Y_1, \ldots, Y_j$.

- The rules in $\mathcal{R}_{\text{push}}$ and $\mathcal{R}_{\text{pop}}$ follow the same intuition as in the single character case, except we have the branching to deal with. In particular, at a push we have one branch corresponding to exploring what happens between the push and the corresponding pops, and a branch for each of the corresponding pops. We choose a selection of these branches to track with the HOPDA and a selection to handle using tests. The difference between $\mathcal{R}_{\text{push}}$ and $\mathcal{R}_{\text{pop}}$ is that the former explores the branch of the push using the HOPDA and the latter uses a test.

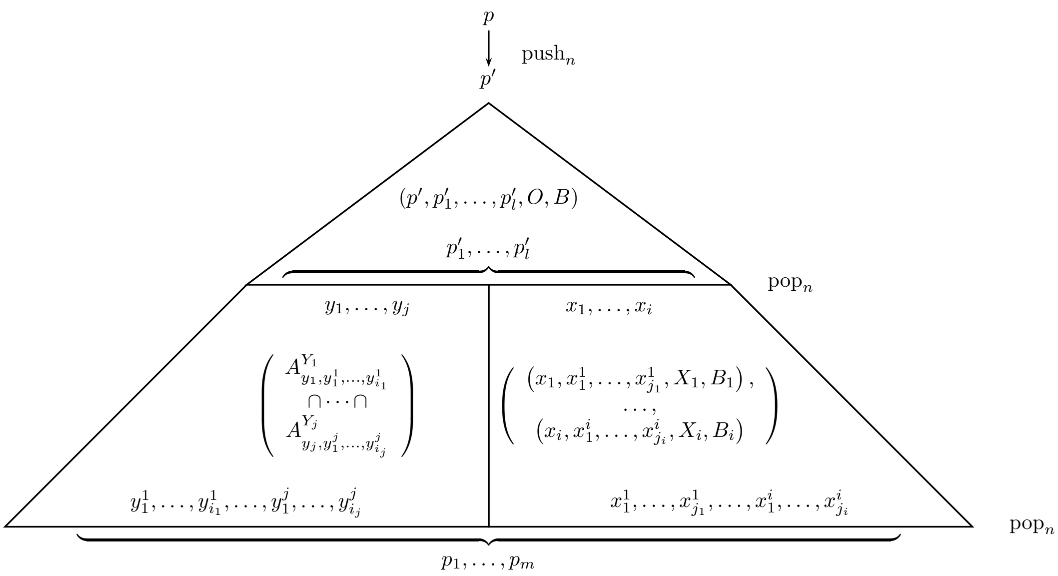


Figure 5: Illustrating the rules in $\mathcal{R}_{\text{push}}$.

In these rules, after the push we're in control state p' and we guess that we will pop to control states $p'_1, \ldots, p'_l$. Hence we have a branch or a test to ensure that this happens. The remaining branches and tests are for what happens after the pops. The start from the states $p'_1, \ldots, p'_l$ and must, in total, pop to the original pop obligation $p_1, \ldots, p_m$. Hence, we distribute these tasks in the same way as the $\mathcal{R}_{\text{br}}$.

Definition 6.4 (P_{-1})**.** *Given an (n, α)-PDA P described by $(\mathcal{P}, \Sigma, \Gamma, \mathcal{R}, \{p_f\}, p_{in}, \gamma_{in}, \theta)$ and automata $A^O_{p,p_1,\ldots,p_m}$ for all $1 \leq m \leq \alpha$, $p, p_1, \ldots, p_m \in \mathcal{P}$, and $O \subseteq \{a_1, \ldots, a_\alpha\}$ we define an $(n-1, \alpha)$-PDA with tests*

$$P_{-1} = \left(\mathcal{P}_{-1}, \Sigma, \Gamma, \mathcal{R}_{-1}, \mathcal{F}_{-1}, p^{-1}_{in}, \gamma_{in}, \theta_{-1}\right)$$

where $\mathcal{P}_{-1}$ is the set

$$\left\{ (p, p_1, \ldots, p_m, O, B) \;\middle|\; \begin{array}{c} 1 \leq m \leq \alpha \;\wedge \\ p, p_1, \ldots, p_m \in \mathcal{P} \;\wedge \\ O, B \subseteq \{a_1, \ldots, a_\alpha\} \;\wedge \\ O \cap B = \emptyset \end{array} \right\} \uplus \left\{p^{-1}_{in}, f\right\}$$

and

$$\begin{array}{rcl} \mathcal{R}_{-1} & = & \mathcal{R}_{init} \cup \mathcal{R}_{sim} \cup \mathcal{R}_{br} \cup \mathcal{R}_{fin} \cup \mathcal{R}_{push} \cup \mathcal{R}_{pop} \\ \mathcal{F}_{-1} & = & \{f\} \end{array}$$

and

$$\theta_{-1}((p, p_1, \ldots, p_m, O, B)) = |B|$$

and is 1 *for all other control states. We define the sets of rules, where in all cases,* $p_1, \ldots, p_m \in \mathcal{P}$ *and* $O, O', B \subseteq \{a_1, \ldots, a_\alpha\}$*, to be as follows:*

- $\mathcal{R}_{init}$ *is the smallest set containing all rules of the form*

$$\left(p_{in}^{-1}, \gamma_{in}\right) \xrightarrow{\varepsilon} ((p_{in}, p_f, \ldots, p_f, \emptyset, \{a_1, \ldots, a_\alpha\}), rew_{\gamma_{in}})$$

  where $|p_f, \ldots, p_f| \leq \alpha$*, and*

- $\mathcal{R}_{fin}$ *is the smallest set containing all rules of the form*

$$((p, p', \emptyset, B), \gamma, A_{\mathtt{tt}}) \xrightarrow{\varepsilon} (f, rew_\gamma)$$

  for all $(p, \gamma) \xrightarrow{\varepsilon} (p', pop_n) \in \mathcal{R}$ *and* $B \subseteq \{a_1, \ldots, a_\alpha\}$*, and*

- $\mathcal{R}_{sim}$ *is the smallest set of rules containing all rules of the form*

$$((p, p_1, \ldots, p_m, O, B), \gamma, A_{\mathtt{tt}}) \xrightarrow{\{b\} \cap B} ((p', p_1, \ldots, p_m, O \setminus \{b\}, B), o)$$

  for $(p, \gamma) \xrightarrow{b} (p', o) \in \mathcal{R}$*, and* $o \notin \{push_n, pop_n\}$*, and*

- $\mathcal{R}_{br}$ *is the smallest set containing all rules of the form*

$$\left((p, p_1, \ldots, p_m, O, B), \gamma, \begin{array}{c} A^{Y_1}_{y_1, y^1_1, \ldots, y^1_{i_1}} \\ \cap \cdots \cap \\ A^{Y_j}_{y_j, y^j_1, \ldots, y^j_{i_j}} \end{array} \right) \xrightarrow{O' \cap B} \left(\begin{array}{c} \left(x_1, x^1_1, \ldots, x^1_{j_1}, X_1, B_1\right), \\ \ldots, \\ \left(x_i, x^i_1, \ldots, x^i_{j_i}, X_i, B_i\right) \end{array}, rew_\gamma \right)$$

  where

$$(p, \gamma) \xrightarrow{\varepsilon} (p'_1, \ldots, p'_l, rew_\gamma) \in \mathcal{R}$$

  and $p'_1, \ldots p'_l$ *is a permutation of*

$$x_1, \ldots, x_i, y_1, \ldots, y_j$$

  and $p_1, \ldots, p_m$ *is a permutation of*

$$x^1_1, \ldots, x^1_{j_1}, \ldots x^i_1, \ldots, x^i_{j_i} y^1_1, \ldots, y^1_{i_1}, \ldots y^j_1, \ldots, y^j_{i_j}$$

  and

$$O \cup O' = X_1 \cup \cdots \cup X_i \cup Y_1 \cup \cdots \cup Y_j$$

  and

$$B = B_1 \cup \cdots \cup B_i \ .$$

- $\mathcal{R}_{push}$ *is the smallest set containing all rules of the form*

$$\left((p, p_1, \ldots, p_m, O, B), \gamma, \begin{array}{c} A^{Y_1}_{y_1, y^1_1, \ldots, y^1_{i_1}} \\ \cap \cdots \cap \\ A^{Y_j}_{y_j, y^j_1, \ldots, y^j_{i_j}} \end{array} \right) \xrightarrow{O' \cap B} \left(\begin{array}{c} (p', p'_1, \ldots, p'_l, X, B_0), \\ \left(x_1, x^1_1, \ldots, x^1_{j_1}, X_1, B_1\right), \\ \ldots, \\ \left(x_i, x^i_1, \ldots, x^i_{j_i}, X_i, B_i\right) \end{array}, rew_\gamma \right)$$

  where

$$(p, \gamma) \xrightarrow{\varepsilon} (p', push_n)$$

and $p'_1, \ldots p'_l$ *is a permutation of*

$$x_1, \ldots, x_i, y_1, \ldots, y_j$$

and $p_1, \ldots, p_m$ *is a permutation of*

$$x^1_1, \ldots, x^1_{j_1}, \ldots x^i_1, \ldots, x^i_{j_i} y^1_1, \ldots, y^1_{i_1}, \ldots y^j_1, \ldots, y^j_{i_j}$$

and

$$O \cup O' = X \cup X_1 \cup \cdots \cup X_i \cup Y_1 \cup \cdots \cup Y_j$$

and

$$B = B_0 \cup \cdots \cup B_i \ .$$

- *we have* $\mathcal{R}_{pop}$ *is the smallest set containing all rules of the form*

$$\left((p, p_1, \ldots, p_m, O, B), \gamma, \begin{array}{c} A^{Y}_{p', p'_1, \ldots, p'_l} \cap \\ A^{Y_1}_{y_1, y^1_1, \ldots, y^1_{i_1}} \\ \cap \cdots \cap \\ A^{Y_j}_{y_j, y^j_1, \ldots, y^j_{i_j}} \end{array} \right) \xrightarrow{O' \cap B} \left(\begin{array}{c} \left(x_1, x^1_1, \ldots, x^1_{j_1}, X_1, B_1\right), \\ \ldots, \\ \left(x_i, x^i_1, \ldots, x^i_{j_i}, X_i, B_i\right) \end{array}, rew_\gamma \right)$$

where

$$(p, \gamma) \xrightarrow{\varepsilon} (p', push_n)$$

and $p'_1, \ldots p'_l$ *is a permutation of*

$$x_1, \ldots, x_i, y_1, \ldots, y_j$$

and $p_1, \ldots, p_m$ *is a permutation of*

$$x^1_1, \ldots, x^1_{j_1}, \ldots x^i_1, \ldots, x^i_{j_i} y^1_1, \ldots, y^1_{i_1}, \ldots y^j_1, \ldots, y^j_{i_j}$$

and

$$O \cup O' = Y \cup X_1 \cup \cdots \cup X_i \cup Y_1 \cup \cdots \cup Y_j$$

and

$$B = B_1 \cup \cdots \cup B_i \ .$$

To complete the reduction, we convert the (n, α)-PDA with tests into a (n, α)-PDA without tests.

Lemma 6.2 (Reduction to Lower Orders)**.** *For every* (n, α)*-PDA* P *we can construct an order-*$(n-1)$ α*-branch HOPDA* P' *such that*

$$Diagonal_{a_1, \ldots, a_\alpha}(P) \iff Diagonal_{a_1, \ldots, a_\alpha}(P') \ .$$

Proof. From Definition 6.4 (P_{-1}) and Lemma 7.1 (below), we obtain from P an $(n-1, \alpha)$-PDA with tests P_{-1} satisfying the conditions of the lemma. To complete the proof, we invoke Theorem 6.1 (Removing Tests) to find P' as required. □

We show correctness of the reduction in Section 7. First we show that we have decidability once we have reduced to order-0.

6.5 Decidability at Order-0

We show that the problem becomes decidable for a 0-PDA P. This is essentially a finite state machine, hence we can linearise the trees generated by saving the list of states that have been branched to in the control state. After one branch has completed, we run the next in the list, until all branches have completed. In this way, a tree of P becomes a run of the linearised HOPDA, and vice-versa. Since each output tree has a bounded number of branches, we do not need an unbounded list in our control states. Thus, we convert the HOPDA into a finite state automaton over words, for which the diagonal problem is decidable.

Definition 6.5 ($\overline{P}$)**.** *Given an $(0, \alpha)$-PDA P described by the tuple $(\mathcal{P}, \Sigma, \Gamma, \mathcal{R}, \mathcal{F}, p_{in}, \gamma_{in}, \theta)$ we define a 0-PDA*

$$\overline{P} = \left(\overline{\mathcal{P}}, \Sigma, \Gamma, \overline{\mathcal{R}}, \mathcal{F}, p_{in}, \gamma_{in}\right)$$

such that

$$\mathcal{P} = \left\{ (p, p_1, \gamma_1, \ldots p_m, \gamma_m) \;\middle|\; \begin{array}{c} p, p_1, \ldots, p_m \in \mathcal{P} \wedge \\ \gamma_1, \ldots, \gamma_m \in \Gamma \wedge \\ 0 \leq m \leq \alpha \end{array} \right\} \cup \{f\}$$

and $\overline{\mathcal{R}}$ is the smallest set containing all rules of the form

$$((p, p_1, \gamma_1, \ldots, p_m, \gamma_m), \gamma) \xrightarrow{b} \left(\left(p_1', \begin{array}{c} p_1, \gamma_1, \ldots, p_m, \gamma_m, \\ p_2', \sigma, \ldots, p_l', \sigma \end{array} \right), rew_\sigma \right)$$

for each

$$(p, \gamma) \xrightarrow{b} (p_1', \ldots, p_l', rew_\sigma) \in \mathcal{R}$$

and all rules

$$((p, p_1, \gamma_1, \ldots, p_m, \gamma_m), \gamma) \xrightarrow{\varepsilon} ((p_1, p_2, \gamma_2, \ldots, p_m, \gamma_m), rew_{\gamma_1})$$

whenever $p \in \mathcal{F}$.

Lemma 6.3 (Decidability at Order-0)**.** *We have*

$$Diagonal_{a_1,\ldots,a_\alpha}(P) \iff Diagonal_{a_1,\ldots,a_\alpha}\left(\overline{P}\right)$$

and hence $Diagonal_{a_1,\ldots,a_\alpha}(P)$ is decidable.

Proof. Take an accepting run tree ρ of P. If this tree contains no branching, then it is straightforward to construct an accepting run of $\overline{P}$. Hence, assume all trees with fewer than α branches have a corresponding run of $\overline{P}$. At a subtree $c[T_1, \ldots, T_m]$ we take the run trees $\rho_1, \ldots, \rho_m$ corresponding to the subtrees. Let $c = \langle p, \gamma \rangle$ and $c_1 = \langle p_1, \gamma \rangle, \ldots, c_m = \langle p_m, \gamma \rangle$ be the configurations at the roots of the subtrees. We build a run beginning at c and transitioning to $\langle (p_1, p_2, \gamma, \ldots, p_m, \gamma), \gamma \rangle$. The run then follows ρ_1 with the extra information in its control state. After ρ_1 accepts, we transition to $\langle (p_2, p_3, \gamma, \ldots, p_m, \gamma), \gamma \rangle$ and then replay ρ_2. We repeat until all subtrees have been dispatched. This gives an accepting run of $\overline{P}$ outputting the same number of each a.

In the other direction, we replay the accepting run ρ of $\overline{P}$ until we reach a configuration $\langle (p_1, p_2, \gamma, \ldots, p_m, \gamma), \gamma \rangle$ via a rule

$$(p, \sigma) \xrightarrow{\varepsilon} ((p_1, p_2, \gamma, \ldots, p_m, \gamma), \mathrm{rew}_\gamma)$$

At this point we apply

$$(p, \sigma) \xrightarrow{\varepsilon} (p_1, \ldots, p_m, \mathrm{rew}_\gamma)$$

of P. We obtain runs for each of the new children as follows. We split the remainder of the run ρ' into m parts $\rho'_1, \ldots, \rho'_m$ where the break points correspond to each application of a rule of the second kind. For each i we replay the transitions of ρ'_1 from $\langle p_i, \gamma \rangle$ to obtain a new run of $\overline{P}$ with fewer applications of the second rule. Inductively, we obtain an accepting run of P that we plug into the ith child. This gives us an accepting run of P outputting the same number of each a. □

6.6 Decidability of The Diagonal Problem

We thus have the following theorem.

Theorem 6.2 (Decidability of the Diagonal Problem). *For a given n-PDA P and output characters $a_1, \ldots, a_\alpha$, it is decidable whether $Diagonal_{a_1,\ldots,a_\alpha}(P)$.*

Proof. We first interpret P as an (n, α)-PDA and then construct via Lemma 6.2 (Reduction to Lower Orders) an $(n-1, \alpha)$-PDA P' such that $\mathrm{Diagonal}_{a_1,\ldots,a_\alpha}(P)$ iff $\mathrm{Diagonal}_{a_1,\ldots,a_\alpha}(P')$. We repeat this step until we have an $(0, \alpha)$-PDA. Then, from Lemma 6.3 (Decidability at Order-0) we obtain decidability as required. □

7 Correctness for Simultaneous Unboundedness

In this section we prove the following lemma.

Lemma 7.1 (Correctness of P_{-1}).

$$Diagonal_{a_1,\ldots,a_\alpha}(P) \iff Diagonal_{a_1,\ldots,a_\alpha}(P_{-1})$$

Proof. The proof follows the same outline as the single character case. To show there is a run with at least m of each character, we take via Lemma 7.2 (below), $m' = (\alpha+1)^m$, and a run of P outputting at least this many of each character. Then from Lemma 7.3 (below) a run of P_{-1} outputting at least m of each character as required. The other direction is shown in Lemma 7.4 (below). □

We first generalise our tree decomposition and notion of scores. We then show that every α-branch subtree of a tree decomposition generates a run tree of P_{-1} matching the scores of the tree. Finally we prove the opposite direction.

7.1 Tree Decomposition of Output Trees

Given an output tree T of P where each $push_n$ operation has a matching pop_n on all branches, we can construct a decomposed tree representation of the run inductively as follows. We define $\mathrm{Tree}(T[\varepsilon]) = T[\varepsilon]$ and, when

$$T = b[T_1, \ldots, T_m]$$

where the rule applied at the root does not contain a $push_n$ operation, we have

$$\mathrm{Tree}(T) = b[\mathrm{Tree}(T_1), \ldots, \mathrm{Tree}(T_m)] \ .$$

In the final case, let

$$T = \varepsilon[T']$$

where the rule applied at the root contains a push_n operation and the corresponding pop_n operations occur at nodes $\eta_1, \ldots, \eta_m$.

Note, in the case of output trees containing an arbitrary number of branches, m may be unbounded. In our case, $m \leq \alpha$, without which our reduction would fail since P_{-1} would not be able to accurately count the number of pop_n nodes. In fact, our trees would have unbounded out degree, which would prevent Lemma 4.2 (Minimum Scores) from generalising.

Let $T_1, \ldots, T_m$ be the output trees rooted at $\eta_1, \ldots, \eta_m$ respectively and let T' be T with these subtrees removed. Observe all branches of T are cut by this operation since the push_n must be matched on all branches. We define

$$\text{Tree}(T) = \varepsilon[\text{Tree}(T'), \text{Tree}(T_1), \ldots, \text{Tree}(T_m)] \ .$$

An accepting run of P has an extra pop_n operation at the end of each branch leading to the empty stack. Let T' be the tree obtained by removing the final pop_n-induced edge leading to the leaves of each branch. We define the tree decomposition of an accepting run to be

$$\text{Tree}(T) = \varepsilon[\text{Tree}(T'), T[\varepsilon], \ldots, T[\varepsilon]]$$

where there are as many $T[\varepsilon]$ as there are leaves of ρ.

Notice that our trees have out-degree at most $(\alpha + 1)$.

7.2 Scoring Trees

We score branches in the same way that we scored trees for the single character case. We simply define $\text{Score}_a(\rho)$ to be $\text{Score}(\rho)$ when a is considered as the only output character (all others are replaced with ε).

We have to slightly modify our minimum score lemma to accommodate the increased out-degree of the nodes in the trees.

Lemma 7.2 (Minimum Scores)**.** *Given a tree T with maximum out-degree $(\alpha+1)$, containing, for each $a \in \{a_1, \ldots, a_\alpha\}$, at least m nodes labelled a, for each $a \in \{a_1, \ldots, a_\alpha\}$ we have*

$$\mathit{Score}_a(T) \geq \log_{(\alpha+1)}(m)$$

Proof. This is a simple extension of the proof of Lemma 4.2 (Minimum Scores). We simply replace the two-child case with a tree with up to $(\alpha + 1)$ children. In this case, we have to use $\log_{(\alpha+1)}$ rather than log to maintain the lemma. ☐

7.3 From Branches to Runs

Lemma 7.3 (Scores to Runs)**.** *Given an accepting output tree ρ of P, if for all $a \in \{a_1, \ldots, a_\alpha\}$ we have $\mathit{Score}_a(\mathit{Tree}(\rho)) \geq m$, then there is some $T \in \mathcal{L}(P_{-1})$ with $|T|_a \geq m$ for all $a \in \{a_1, \ldots, a_\alpha\}$.*

Proof. We will construct a tree ρ_{-1} in $\mathcal{L}(P_{-1})$ top down. At each step we will maintain a "frontier" of ρ_{-1} and extend one leaf of this frontier until the whole tree is constructed. The frontier is of the form

$$(c_1, \eta_1, O_1, B_1, \ldots, c_l, \eta_l, O_l B_l)$$

which means that there are l nodes in the frontier. We have $B_1 \uplus \cdots \uplus B_l = \{a_1, \ldots, a_\alpha\}$ and each B_i indicates that the ith branch, ending in configuration c_i, is responsible for outputting enough of each of the characters in B_i. Each η_i is the corresponding node in Tree(ρ) that is being tracked by the ith branch of the output of P_{-1}.

Let p_f be the final (accepting) control state of P and let $T = \text{Tree}(\rho)$. We begin at the root node of T, which corresponds to the initial configuration of ρ. Let $\langle p, [s]_n \rangle$ be this initial configuration and let $c = \langle (p, p_f, \ldots, p_f, \emptyset, \{a_1, \ldots, a_\alpha\}), s \rangle$ be the configuration of P_{-1} after an application of a rule from $\mathcal{R}_{\text{init}}$. The initial frontier is $(c, \varepsilon, \{a_1, \ldots, a_\alpha\})$.

Thus, assume we have a frontier

$$(c_1, \eta_1, O_1, B_1, \ldots, c_h, \eta_h, O_h, B_h)$$

and for each of the sequences c_{-1}, η, O, B of the frontier we have

1. T' is the subtree of T rooted at η, and
2. $c = \langle p, s \rangle$ labelling η, and
3. $c_{-1} = \langle (p, p_1, \ldots, p_m, O, B), \text{top}_n(s) \rangle$, and
4. the node of ρ corresponding to η has m locations where the top_n stack is first popped via rules reaching $p_1, \ldots, p_m$, moreover, these leaves have corresponding leaves in T', and
5. the branch from the root of the constructed run to the node labelled c_{-1} in the frontier outputs, for each $a \in B$, at least $(m - \text{Score}_a(T'))$ occurrences of a, and
6. $O \cap B = \emptyset$ and for each $a \in O$ there is at least one node labelled by a in T'.

Pick such a sequence c_{-1}, η, O, B. We replace this sequence using a transition of P_{-1} in a way that produces a new frontier with the above properties and moves us a step closer to reaching leaves of T. There are three cases when we are dealing with internal nodes.

- $T' = b[T_1]$.

  In this case there is a transition $c \xrightarrow{b} c'$ via a rule $(p, \gamma) \xrightarrow{b} (p', o)$ where $o \notin \{\text{push}_n, \text{pop}_n\}$. Hence, we have

  $$((p, p_1, \ldots p_m, O, B), \gamma, A_{\mathtt{tt}}) \xrightarrow{\{b\} \cap B} ((p', p_1, \ldots p_m, O \setminus \{b\}, B), o)$$

  in P_{-1} and thus we can extend ρ_{-1} with a transition $c_{-1} \xrightarrow{b} c'_{-1}$ via this rule. The new frontier is obtained by replacing c_{-1}, η, O, B with $c'_{-1}, \eta', O \setminus \{b\}, B$ where η' is the child of η. The properties on the frontier are easily seen to be retained.

- $T' = \varepsilon[T_1, \ldots, T_l]$ from a rule $(p, \gamma) \xrightarrow{\varepsilon} (p'_1, \ldots, p'_l, \text{rew}_\gamma)$ of P.

  We separate $B = B'_1 \uplus \cdots \uplus B'_i$ such that B'_j is the set of characters a that have their score derived from T_j (i.e. the subtree with the higher score for a characters). Let O' be the set of all a who had a $+1$ in their score derived from another subtree. Let $\langle x_1, s \rangle, \ldots \langle x_i, s \rangle$ be the configurations labelling the root nodes $\eta_0, \eta_1, \ldots, \eta_i$ of these subtrees. Let $\langle y_1, s \rangle, \ldots, \langle y_j, s \rangle$ be the configurations labelling the root nodes of the remaining subtrees. Since T' includes m leaves that are followed in ρ by pops to $p_1, \ldots, p_m$ we can distribute these control states amongst the branches, obtaining

  $$x^1_1, \ldots, x^1_{j_1}, \ldots x^i_1, \ldots, x^i_{j_i} y^1_1, \ldots, y^1_{i_1}, \ldots y^j_1, \ldots, y^j_{i_j} \ .$$

Finally, we can distribute

$$O \cup O' = X_1 \cup \cdots \cup X_i \cup Y_1 \cup \cdots \cup Y_j$$

amongst the subtrees $T_1, \ldots, T_l$ since O can be distributed by assumption and we chose O' such that this can be done.

From the runs corresponding to $T_1, \ldots, T_l$ and our choices above we know that the tests will pass. That is, $c_{-1} \in \mathcal{L}\left(A^{Y_1}_{p'_1, y^1_1, \ldots y^1_{i_1}}\right), \ldots, c_{-1} \in \mathcal{L}\left(A^{Y_j}_{p'_j, y^j_1, \ldots y^j_{i_j}}\right)$.

Hence, we apply to c_{-1} the rule

$$\left((p, p_1, \ldots, p_m, O, B), \gamma, \begin{array}{c} A^{Y_1}_{y_1, y^1_1, \ldots, y^1_{i_1}} \\ \cap \cdots \cap \\ A^{Y_j}_{y_j, y^j_1, \ldots, y^j_{i_j}} \end{array} \right) \xrightarrow{O' \cap B} \left(\begin{array}{c} \left(x_1, x^1_1, \ldots, x^1_{j_1}, X_1, B_1\right), \\ \ldots, \\ \left(x_i, x^i_1, \ldots, x^i_{j_i}, X_i, B_i\right) \end{array}, \mathrm{rew}_\gamma \right)$$

and obtain configurations $c^1_{-1}, \ldots, c^i_{-1}$ and a new frontier satisfying the required properties by replacing c_{-1}, η, O, B with the sequence

$$c^1_{-1}, \eta_1, X_1, B'_1, \ldots c^i_{-1}, \eta_i, X_i, B'_i \; .$$

- $T' = \varepsilon[T_1, \ldots, T_l]$ not from a rule $(p, \gamma) \xrightarrow{\varepsilon} (p'_1, \ldots, p'_l, \mathrm{rew}_\gamma)$ of P.

  In this case we have that T' (subtree of the decomposition T) corresponds to a run tree $\rho_{T'}$ that can be decomposed into

  - $c[\rho']$ with $c' = \langle p', \mathrm{push}_n(s)\rangle$ at the root of ρ' via a rule $(p, \gamma) \xrightarrow{\varepsilon} (p', \mathrm{push}_n)$ and l leaf nodes labelled $c_1, \ldots, c_l$ respectively, and
  - runs $\rho_1, \ldots, \rho_l$ with the roots labelled $c'_1 = \langle p'_1, s\rangle, \ldots, c'_l = \langle p'_l, s\rangle$ where, for each i, we have $c_i \xrightarrow{\varepsilon} c'_i$ via a pop_n rule, and these are the first points s is seen along each branch, and
  - the leaves of $\rho_1, \ldots, \rho_l$ are the leaves of $\rho_{T'}$.

  There are two cases depending on whether we send the HOPDA down the branch corresponding to the push.

  - We separate $B = B'_0 \uplus B'_1 \uplus \cdots \uplus B'_i$ such that B'_j is the set of characters a that have their score derived from T_j (i.e. the subtree with the higher score for a characters). Assume T_1 is amongst these subtrees (and will get B'_0). Let O' be the set of all a who had a $+1$ in their score derived from another subtree. Let $\langle p', \mathrm{push}_n(s)\rangle, \langle x_1, s\rangle, \ldots \langle x_i, s\rangle$ be the configurations labelling the root nodes $\eta_1, \ldots, \eta_i$ of these subtrees, with the first belonging to T_1. Let $\langle y_1, s\rangle, \ldots, \langle y_j, s\rangle$ be the configurations labelling the root nodes of the remaining subtrees. Since T' has m leaves that are followed in ρ by pops to $p_1, \ldots, p_m$ we can distribute these control states amongst the branches, obtaining

    $$x^1_1, \ldots, x^1_{j_1}, \ldots x^i_1, \ldots, x^i_{j_i} y^1_1, \ldots, y^1_{i_1}, \ldots y^j_1, \ldots, y^j_{i_j} \; .$$

    We can also distribute

    $$O \cup O' = X \cup X_1 \cup \cdots \cup X_i \cup Y_1 \cup \cdots \cup Y_j$$

amongst the subtrees $T_1, \ldots, T_l$ with X belonging to T_1 since O can be distributed by assumption and we chose O' such that this can be done.

From the existence of the runs $\rho_1, \ldots, \rho_l$ we know $c_{-1} \in \mathcal{L}\left(A^{Y_1}_{p'_1, y^1_1, \ldots y^1_{i_1}}\right), \ldots, c_{-1} \in \mathcal{L}\left(A^{Y_j}_{p'_j, y^j_1, \ldots y^j_{i_j}}\right)$.

Hence, we apply to c_{-1} the rule

$$\left((p, p_1, \ldots, p_m, O, B), \gamma, \begin{array}{c} A^{Y_1}_{y_1, y^1_1, \ldots, y^1_{i_1}} \\ \cap \cdots \cap \\ A^{Y_j}_{y_j, y^j_1, \ldots, y^j_{i_j}} \end{array}\right) \xrightarrow{O' \cap B} \left(\begin{array}{c} (p', p'_1, \ldots, p'_l, X, B_0)\,, \\ \left(x_1, x^1_1, \ldots, x^1_{j_1}, X_1, B_1\right), \\ \ldots, \\ \left(x_i, x^i_1, \ldots, x^i_{j_i}, X_i, B_i\right) \end{array}, \mathrm{rew}_\gamma\right)$$

and obtain configurations $c^0_{-1}, c^1_{-1}, \ldots, c^i_{-1}$ and a new frontier satisfying the required properties by replacing c_{-1}, η, O, B with the sequence

$$c^0_{-1}, \eta_0, X, B'_0, c^1_{-1}, \eta_1, X_1, B'_1, \ldots c^i_{-1}, \eta_i, X_i, B'_i \ .$$

– We separate $B = B'_1 \uplus \cdots \uplus B'_i$ such that B'_j is the set of characters a that have their score derived from T_j (i.e. the subtree with the higher score for a characters). Assume T_1 is not amongst these subtrees. Let O' be the set of all a who had a $+1$ in their score derived from another subtree. Let $\langle x_1, s\rangle, \ldots \langle x_i, s\rangle$ be the configurations labelling the root nodes $\eta_1, \ldots, \eta_i$ of these subtrees. Let $\langle p', \mathrm{push}_n(s)\rangle, \langle y_1, s\rangle, \ldots, \langle y_j, s\rangle$ be the configurations labelling the root nodes of the remaining subtrees, with the first belonging to T_1. Since T' has m leaves that are followed in ρ by pops to $p_1, \ldots, p_m$ we can distribute these control states amongst the branches, obtaining

$$x^1_1, \ldots, x^1_{j_1}, \ldots x^i_1, \ldots, x^i_{j_i} y^1_1, \ldots, y^1_{i_1}, \ldots y^j_1, \ldots, y^j_{i_j} \ .$$

We can also distribute

$$O \cup O' = X_1 \cup \cdots \cup X_i \cup Y \cup Y_1 \cup \cdots \cup Y_j$$

amongst the subtrees $T_1, \ldots, T_l$ with Y belonging to T_1 since O can be distributed by assumption and we chose O' such that this can be done.

From the existence of ρ' we know that $c_{-1} \in \mathcal{L}\left(A^{Y}_{p', p'_1, \ldots, p'_l}\right)$ and from the existence of $\rho_1, \ldots, \rho_l$ we also know $c_{-1} \in \mathcal{L}\left(A^{Y_1}_{p'_1, y^1_1, \ldots y^1_{i_1}}\right), \ldots, c_{-1} \in \mathcal{L}\left(A^{Y_j}_{p'_j, y^j_1, \ldots y^j_{i_j}}\right)$.

Hence, we apply to c_{-1} the rule

$$\left((p, p_1, \ldots, p_m, O, B), \gamma, \begin{array}{c} A^{Y}_{p', p'_1, \ldots, p'_l} \cap \\ A^{Y_1}_{y_1, y^1_1, \ldots, y^1_{i_1}} \\ \cap \cdots \cap \\ A^{Y_j}_{y_j, y^j_1, \ldots, y^j_{i_j}} \end{array}\right) \xrightarrow{O' \cap B} \left(\begin{array}{c} \left(x_1, x^1_1, \ldots, x^1_{j_1}, X_1, B_1\right), \\ \ldots, \\ \left(x_i, x^i_1, \ldots, x^i_{j_i}, X_i, B_i\right) \end{array}, \mathrm{rew}_\gamma\right)$$

and obtain configurations $c^1_{-1}, \ldots, c^i_{-1}$ and a new frontier satisfying the required properties by replacing c_{-1}, η, O, B with the sequence

$$c^1_{-1}, \eta_1, X_1, B'_1, \ldots c^i_{-1}, \eta_i, X_i, B'_i \ .$$

Finally, we reach a leaf node η with a run outputting the required number of as. We need to show that the run constructed is accepting. From the tree decomposition, we know that the corresponding node of ρ is immediately followed by a pop_n. Thus, from our conditions on the frontier, we must have $m = 1$ and $O = \emptyset$. We also have a rule $(p, \gamma) \xrightarrow{\varepsilon} (p_1, \text{pop}_n)$ and therefore $((p, p_1, \emptyset, B), \gamma, A_{\mathtt{tt}}) \xrightarrow{\varepsilon} (f, \text{rew}_\gamma)$ with which we can complete the run of P_{-1} as required. ☐

7.4 The Other Direction

Finally, we need to show that each accepting run tree of P_{-1} gives rise to an accepting run tree of P containing at least as many of each output character a.

Lemma 7.4 (P_{-1} to P)**.** *We have* $\mathit{Diagonal}_{a_1,\ldots,a_\alpha}(P_{-1})$ *implies* $\mathit{Diagonal}_{a_1,\ldots,a_\alpha}(P)$*.*

Proof. Take an accepting run tree ρ_{-1} of P_{-1}. We show that there exists a corresponding run tree ρ of P outputting at least as many as.

We maintain a frontier

$$c_1, \ldots, c_h$$

of ρ_{-1} and a run ρ of P "with holes" such that

- there are h nodes of ρ labelled by $c_1, \ldots, c_h$ respectively (these are the holes), and
- each of these holes labelled c is the only child of a parent node labelled c' of P, and
- for each corresponding pair c and c' we have
  - $c' = \langle p, s\rangle$, and
  - $c = \langle (p, p_1, \ldots, p_m, O, B), \text{top}_n(s)\rangle$, and
  - the node labelled by c has m children with the ith child being labelled $\langle p_i, \text{pop}_n(s)\rangle$, and
  - all leaf nodes of ρ are accepting, and
  - for each $a \in \{a_1, \ldots, a_\alpha\}$ the number of a output by run tree of P is at least as many as on the branch of P_{-1} to the configuration with $a \in B$ less 1 if $a \in O$.

Initially after a rule from $\mathcal{R}_{\text{init}}$ we have the frontier $c = \langle (p, p_f, \ldots, p_f, \emptyset, \{a_1, \ldots, a_\alpha\}), s\rangle$ with corresponding run ρ of P being

$$\langle p, [s]_n\rangle [c[\langle p_f, []_n\rangle, \ldots, \langle p_f, []_n\rangle]] \ .$$

Pick a configuration $c_{-1} = \langle (p, p_1, \ldots, p_m, O, B), \text{top}_n(s)\rangle$ of the frontier that is not a leaf of ρ_{-1} and its corresponding node in ρ with parent labelled $c = \langle p, s\rangle$. Let ρ'_{-1} be the subtree of P_{-1} rooted at this configuration.

We show how to extend the frontier closer to the leaves of ρ_{-1}. There are several cases depending on the transition of P_{-1} used to exit our chosen node.

- $\rho'_{-1} = c_{-1}\left[\rho^1_{-1}\right]$ and the rule applied is of the form

  $$((p, p_1, \ldots, p_m, O, B), \gamma, A_{\mathtt{tt}}) \xrightarrow{\{b\}\cap B} ((p', p_1, \ldots, p_m, O \setminus \{b\}, B), o) \ .$$

  Let c'_{-1} be the configuration labelling the root of ρ^1_{-1}. We have $(p, \gamma) \xrightarrow{b} (p', o) \in \mathcal{R}$ and $o \notin \{\text{push}_n, \text{pop}_n\}$. We can apply $c \xrightarrow{b} c'$. Let η be the node labelled c_{-1}. We insert above η a node labelled c'. Then we change the label of η to c'_{-1}. We keep the same children of η. This extended run maintains all properties as required.

- $\rho'_{-1} = c_{-1}\big[\rho^1_{-1}, \ldots, \rho^i_{-1}\big]$ via a rule

$$\left((p, p_1, \ldots, p_m, O, B), \gamma, \begin{array}{c} A^{Y_1}_{y_1, y^1_1, \ldots, y^1_{i_1}} \\ \cap \cdots \cap \\ A^{Y_j}_{y_j, y^j_1, \ldots, y^j_{i_j}} \end{array} \right) \xrightarrow{O' \cap B} \left(\begin{array}{c} \big(x_1, x^1_1, \ldots, x^1_{j_1}, X_1, B_1\big) , \\ \ldots, \\ \big(x_i, x^i_1, \ldots, x^i_{j_i}, X_i, B_i\big) \end{array} , \mathrm{rew}_\gamma \right)$$

derived from some rule

$$(p, \gamma) \xrightarrow{\varepsilon} (p'_1, \ldots, p'_l, \mathrm{rew}_\gamma) \in \mathcal{R} \ .$$

In this case, we apply the above rule to ρ which means taking the node η labelled c and replacing its "hole" child with l new children. We need to rebuild the rest of the tree the from these nodes. These nodes have configurations $\langle p'_1, s\rangle, \ldots, \langle p'_l, s\rangle$. These control states are distributed between $x_1, \ldots, x_i$ and $y_1, \ldots, y_j$. Consider y_1 (the other $y_2, \ldots, y_j$ are identical). We have from the respective passed test that $\langle y_1, s\rangle$ has a run where the first popping of the top_n stack leads to configurations $\langle y^1_1, s\rangle, \ldots, \langle y^1_{i_1}, s\rangle$. We insert this run underneath the node corresponding to the y_1. Since $y^1_1, \ldots, y^1_{i_1}$ appear amongst $p_1, \ldots, p_m$ we append the subtrees that appeared as the relevant children of the node labelled c_{-1} to complete these branches. The remaining subtrees corresponding to $p_1, \ldots, p_m$ are distributed amongst $x^1_1, \ldots, x^1_{j_1}, \ldots, x^i_1, \ldots, x^i_{j_i}$. Consider x_1 (the others are identical arguments), we have a new child labelled by $\langle \big(x_1, x^1_1, \ldots, x^1_{j_1}, X_1, B_1\big), \mathrm{top}_n(s)\rangle$. We take the subrees distributed to $x^1_1, \ldots, x^1_{j_1}$ as children of this new child to satisfy the requirements.

The new frontier replaces c_{-1} with

$$\begin{array}{c} \langle \big(x_1, x^1_1, \ldots, x^1_{j_1}, X_1, B_1\big), \mathrm{top}_n(s)\rangle, \\ \ldots, \\ \langle \big(x_j, x^i_1, \ldots, x^i_{j_i}, X_j, B_j\big), \mathrm{top}_n(s)\rangle \end{array}$$

which satisfies all properties as needed.

- $\rho'_{-1} = c_{-1}\big[\rho^1_{-1}, \ldots, \rho^i_{-1}\big]$ via a rule

$$\left((p, p_1, \ldots, p_m, O, B), \gamma, \begin{array}{c} A^{Y_1}_{y_1, y^1_1, \ldots, y^1_{i_1}} \\ \cap \cdots \cap \\ A^{Y_j}_{y_j, y^j_1, \ldots, y^j_{i_j}} \end{array} \right) \xrightarrow{O' \cap B} \left(\begin{array}{c} (p', p'_1, \ldots, p'_l, X, B_0) , \\ \big(x_1, x^1_1, \ldots, x^1_{j_1}, X_1, B_1\big) , \\ \ldots, \\ \big(x_i, x^i_1, \ldots, x^i_{j_i}, X_i, B_i\big) \end{array} , \mathrm{rew}_\gamma \right)$$

derived from some rule

$$(p, \gamma) \xrightarrow{\varepsilon} (p', \mathrm{push}_n)$$

In this case, we apply the above rule to ρ. This means replacing the node labelled c_{-1} with one labelled $\langle p', \mathrm{push}_n(s)\rangle$. This new node has a new child node with the label

$$\langle (p', p'_1, \ldots, p'_l, X, B_0), \mathrm{top}_n(s)\rangle \ .$$

We need to add l children to this new "hole" node.

These nodes have configurations $\langle p'_1, s\rangle, \ldots, \langle p'_l, s\rangle$ (since $s = \mathrm{pop}_n(\mathrm{push}_n(s))$). These control states are distributed between $x_1, \ldots, x_i$ and $y_1, \ldots, y_j$. Consider y_1 (the other $y_2, \ldots, y_j$ are identical). We have from the passed test that $\langle y_1, s\rangle$ has a run where the first popping of the top_n stack leads to configurations $\langle y^1_1, \mathrm{pop}_n(s)\rangle, \ldots, \langle y^1_{i_1}, \mathrm{pop}_n(s)\rangle$.

We append this run tree as a child of the node corresponding to y_1. Since $y_1^1, \ldots, y_{i_1}^1$ appear amongst $p_1, \ldots, p_m$ we append the relevant subtrees we had already constructed for these nodes to complete these branches with the required properties.

Now consider x_1 (the other cases are symmetric). In this case we append a node labelled $\langle (x_1, x_1^1, \ldots, x_{j_1}^1, X_1, B_1), \mathrm{top}_n(s) \rangle$ as a child of the node corresponding to x_1. Since $x_1^1, \ldots, x_{j_1}^1$ appear amongst $p_1, \ldots, p_m$ we append the relevant subtrees we had already constructed for these nodes to complete these branches with the required properties.

The new frontier replaces c_{-1} with

$$\langle (p', p_1', \ldots, p_l', X, B_0), \mathrm{top}_n(s) \rangle$$

and

$$\begin{gathered} \langle (x_1, x_1^1, \ldots, x_{j_1}^1, X_1, B_1), \mathrm{top}_n(s) \rangle, \\ \ldots, \\ \langle (x_j, x_1^i, \ldots, x_{j_i}^i, X_i, B_i), \mathrm{top}_n(s) \rangle \end{gathered}$$

which satisfies all the required properties.

- $\rho'_{-1} = c_{-1}\left[\rho_{-1}^1, \ldots, \rho_{-1}^i\right]$ via a rule

$$\left((p, p_1, \ldots, p_m, O, B), \gamma, \begin{matrix} A^Y_{p', p_1', \ldots, p_l'} \cap \\ A^{Y_1}_{y_1, y_1^1, \ldots, y_{i_1}^1} \\ \cap \cdots \cap \\ A^{Y_j}_{y_j, y_1^j, \ldots, y_{i_j}^j} \end{matrix} \right) \xrightarrow{O' \cap B} \left(\begin{matrix} (x_1, x_1^1, \ldots, x_{j_1}^1, X_1, B_1), \\ \ldots, \\ (x_i, x_1^i, \ldots, x_{j_i}^i, X_i, B_i) \end{matrix}, \mathrm{rew}_\gamma \right)$$

derived from some rule

$$(p, \gamma) \xrightarrow{\varepsilon} (p', \mathrm{push}_n)$$

In this case, we again apply the above rule to ρ. This means replacing the node labelled c_{-1} with one labelled $\langle p', \mathrm{push}_n(s) \rangle$. Since we know the test $A^y_{p', p_1', \ldots, p_l'}$ passed we have a run popping the newly pushed stack to controls $p_1', \ldots, p_l'$. We set this run tree as the only child of the node whose label we replaced. This new tree has l leaves which we need to complete.

These leaf nodes are completed using the same argument as the previous case. That is, they are labelled with configurations $\langle p_1', s \rangle, \ldots, \langle p_l', s \rangle$. These control states are distributed between $x_1, \ldots, x_i$ and $y_1, \ldots, y_j$. Consider y_1 (the other $y_2, \ldots, y_j$ are identical). We have from the passed test that $\langle y_1, s \rangle$ has a run where the first popping of the top_n stack leads to configurations $\langle y_1^1, \mathrm{pop}_n(s) \rangle, \ldots, \langle y_{i_1}^1, \mathrm{pop}_n(s) \rangle$. We append this run tree as a child of the node corresponding to y_1. Since $y_1^1, \ldots, y_{i_1}^1$ appear amongst $p_1, \ldots, p_m$ we append the relevant subtrees we had already constructed for these nodes to complete these branches with the required properties.

Now consider x_1 (the other cases are symmetric). In this case we append a node labelled $\langle (x_1, x_1^1, \ldots, x_{j_1}^1, X_1, B_1), \mathrm{top}_n(s) \rangle$ as a child of the node corresponding to x_1. Since $x_1^1, \ldots, x_{j_1}^1$ appear amongst $p_1, \ldots, p_m$ we append the relevant subtrees we had already constructed for these nodes to complete these branches with the required properties.

The new frontier replaces c_{-1} with

$$\langle (p', p_1', \ldots, p_l', X, B_0), \mathrm{top}_n(s) \rangle$$

and

$$\langle (x_1, x^1_1, \ldots, x^1_{j_1}, X_1, B_1), \mathrm{top}_n(s)\rangle,$$
$$\ldots,$$
$$\langle (x_j, x^i_1, \ldots, x^i_{j_i}, X_i, B_i), \mathrm{top}_n(s)\rangle$$

which satisfies all the required properties.

- $\rho'_{-1} = c_{-1}[\langle f, s\rangle]$.

  In this case c has the form

  $$\langle (p, p', \emptyset, B), \mathrm{top}_n(s)\rangle$$

  and there is a rule

  $$(p, \gamma) \xrightarrow{\varepsilon} (p', \mathrm{pop}_n) \ .$$

  We can remove the hole from ρ by applying this rule. That is, we remove the hole node, setting its parent to have its (only) child as its child. This is possible since by our conditions the child has the label $\langle p', \mathrm{pop}_n(s)\rangle$. We remove c_{-1} from the frontier.

Thus, the frontier moves towards the leaves of the tree and finally is empty. At this point we have an accepting run of P as required. To see that the run outputs enough of each character, one needs to observe that at each stage the tests and O component of the control state ensured at least one character output for each that appeared in some O' labelling a transition. Then, for characters output along branches followed were reproduced faithfully. □

8 Conclusions

We have shown, using a recent result by Zetzsche, that the downwards closures of languages defined by HOPDA are computable. We believe this to be a useful foundational result upon which new analyses may be based. Our result already has several immediate consequences, including piecewise testability and asynchronous parameterised systems.

A natural next step is to consider collapsible pushdown systems, which are equivalent to recursion schemes (without the safety constraint). However, it is not currently clear how to generalise our techniques due to the non-local behaviour introduced by collapse. We may also try to adapt our techniques to a higher-order version of BS-automata [3], which may be used, e.g., to check boundedness of resource usage for higher-order programs.

Acknowledgements We thank Georg Zetzsche for keeping us up to date with his work, Jason Crampton for knowing about logarithms when they were most required, and Chris Broadbent for discussions. This work was supported by the Engineering and Physical Sciences Research Council [EP/K009907/1].

References

[1] K. Aehlig, J. G. de Miranda, and C.-H. L. Ong. Safety is not a restriction at level 2 for string languages. In *Foundations of Software Science and Computational Structures, 8th International Conference, FOSSACS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, pages 490–504, 2005.

[2] A. V. Aho. Indexed grammars - an extension of context-free grammars. *J. ACM*, 15(4):647–671, 1968.

[3] Mikolaj Bojanczyk. Beyond omega-regular languages. In *27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, March 4-6, 2010, Nancy, France*, pages 11–16, 2010.

[4] Ahmed Bouajjani, Markus Müller-Olm, and Tayssir Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In *CONCUR*, pages 473–487, 2005.

[5] C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. C-shore: a collapsible approach to higher-order verification. In *ICFP*, pages 13–24, 2013.

[6] C. H. Broadbent and N. Kobayashi. Saturation-based model checking of higher-order recursion schemes. In *CSL*, pages 129–148, 2013.

[7] Christopher H. Broadbent, Arnaud Carayol, Matthew Hague, and Olivier Serre. A saturation method for collapsible pushdown systems. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, pages 165–176, 2012.

[8] Christopher H. Broadbent, Arnaud Carayol, C.-H. Luke Ong, and Olivier Serre. Recursion schemes and logical reflection. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 120–129, 2010.

[9] B. Courcelle. On constructing obstruction sets of words. *Bulletin of the EATCS*, 44:178–186, 1991.

[10] A. Cyriac, P. Gastin, and K. N. Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR*, pages 547–561, 2012.

[11] W. Czerwiński and W. Martens. A note on decidable separability by piecewise testable languages. *CoRR*, abs/1410.1042, 2014.

[12] J. Esparza, A. Kucera, and S. Schwoon. Model checking LTL with regular valuations for pushdown systems. *Inf. Comput.*, 186(2):355–376, 2003.

[13] Javier Esparza and Pierre Ganty. Complexity of pattern-based verification for multithreaded programs. In *POPL*, pages 499–510, 2011.

[14] Javier Esparza and Andreas Podelski. Efficient algorithms for pre* and post* on interprocedural parallel flow graphs. In *POPL*, pages 1–11, 2000.

[15] M. Hague. Saturation of concurrent collapsible pushdown systems. In *FSTTCS*, pages 313–325, 2013.

[16] M. Hague. Senescent ground tree rewrite systems. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 48:1–48:10, 2014.

[17] M. Hague and A. W. Lin. Synchronisation- and reversal-bounded analysis of multithreaded programs with counters. In *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, pages 260–276, 2012.

[18] M. Hague, A. S. Murawski, C.-H. Luke Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, pages 452–461, 2008.

[19] L.H. Haines. On free monoids partially ordered by embedding. *J. Combinatorial Theory*, 6:9498, 1969.

[20] Vineet Kahlon. Boundedness vs. unboundedness of lock chains: Characterizing decidability of pairwise CFL-reachability for threads communicating via locks. In *LICS*, pages 27–36, 2009.

[21] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS '02: Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures*, pages 205–222, London, UK, 2002. Springer-Verlag.

[22] T. Knapik, D. Niwinski, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and panic automata. In *ICALP*, pages 1450–1461, 2005.

[23] N. Kobayashi. Model-checking higher-order functions. In *PPDP*, pages 25–36, 2009.

[24] N. Kobayashi. GTRecS2: A model checker for recursion schemes based on games and types. A

tool available at `http://www-kb.is.s.u-tokyo.ac.jp/~koba/gtrecs2/`, 2012.

[25] N. Kobayashi and A. Igarashi. Model-checking higher-order programs with recursive types. In *ESOP*, pages 431–450, 2013.

[26] N. Kobayashi, R. Sato, and H. Unno. Predicate abstraction and cegar for higher-order model checking. In *PLDI*, pages 222–233, 2011.

[27] Akash Lal and Thomas W. Reps. Reducing concurrent analysis under a context bound to sequential analysis. *Formal Methods in System Design*, 35(1):73–97, 2009.

[28] P. Madhusudan and G. Parlato. The tree width of auxiliary storage. In *POPL*, pages 283–294, 2011.

[29] A. N. Maslov. Multilevel stack automata. *Problems of Information Transmission*, 15:1170–1174, 1976.

[30] R. P. Neatherway, S. J. Ramsay, and C.-H. L. Ong. A traversal-based algorithm for higher-order model checking. In *ICFP*, pages 353–364, 2012.

[31] P. Parys. On the significance of the collapse operation. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 521–530, 2012.

[32] V. Penelle. Rewriting higher-order stack trees. In *Computer Science - Theory and Applications - 10th International Computer Science Symposium in Russia, CSR 2015, Listvyanka, Russia, July 13-17, 2015, Proceedings*, pages 364–397, 2015.

[33] G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Trans. Program. Lang. Syst.*, 22(2):416–430, 2000.

[34] S. J. Ramsay, R. P. Neatherway, and C.-H. L. Ong. A type-directed abstraction refinement approach to higher-order model checking. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 61–72, 2014.

[35] A. Seth. Games on higher order multi-stack pushdown systems. In *RP*, pages 203–216, 2009.

[36] S. La Torre, A. Muscholl, and I. Walukiewicz. Safety of parametrized asynchronous shared-memory systems is almost always decidable. In *CONCUR*, 2015. To appear.

[37] Salvatore La Torre and Margherita Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In *CONCUR*, pages 203–218, 2011.

[38] H. Unno, N. Tabuchi, and N. Kobayashi. Verification of tree-processing programs via higher-order model checking. In *APLAS*, 2010.

[39] J. van Leeuwen. Effective constructions in well-partially-ordered free monoids. *Discrete Mathematics*, 21(3):237252, 1978.

[40] Georg Zetzsche. An approach to computing downward closures. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, pages 440–451, 2015.