On the Boundedness Problem for Higher-Order Pushdown Vector Addition Systems

Vincent Penelle, Sylvain Salvati, Grégoire Sutre

On the Boundedness Problem for Higher-Order Pushdown Vector Addition Systems

Vincent Penelle
LaBRI, Univ. Bordeaux, CNRS, Bordeaux-INP, Talence, France
vincent.penelle@labri.fr

Sylvain Salvati
CRIStAL, Univ. Lille, INRIA, Lille, France
sylvain.salvati@univ-lille.fr

Grégoire Sutre
LaBRI, Univ. Bordeaux, CNRS, Bordeaux-INP, Talence, France
gregoire.sutre@labri.fr

Abstract

Karp and Miller's algorithm is a well-known decision procedure that solves the termination and boundedness problems for vector addition systems with states (VASS), or equivalently Petri nets. This procedure was later extended to a general class of models, well-structured transition systems, and, more recently, to pushdown VASS. In this paper, we extend pushdown VASS to higher-order pushdown VASS (called HOPVASS), and we investigate whether an approach à la Karp and Miller can still be used to solve termination and boundedness. We provide a decidable characterisation of runs that can be iterated arbitrarily many times, which is the main ingredient of Karp and Miller's approach. However, the resulting Karp and Miller procedure only gives a semi-algorithm for HOPVASS. In fact, we show that coverability, termination and boundedness are all undecidable for HOPVASS, even in the restricted subcase of one counter and an order 2 stack. On the bright side, we prove that this semi-algorithm is in fact an algorithm for higher-order pushdown automata.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory, Theory of computation → Logic and verification

Keywords and phrases Higher-order pushdown automata, Vector addition systems, Boundedness problem, Termination problem, Coverability problem

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.44

Funding This work was supported by the grant ANR-17-CE40-0028 of the French National Research Agency ANR (project BRAVAS).

1 Introduction

Termination of a program is a desirable feature in computer science. As it is undecidable on Turing machines, an important challenge is to find models as expressive as possible while retaining decidability of termination. A prominent model having this property is vector addition systems with states (or VASS for short), introduced by Karp and Miller (without states) to model and analyse concurrent systems [10]. They also provide an algorithm, known as the Karp and Miller tree, which can decide termination as well as boundedness (i.e., finiteness of the set of reachable configurations). This algorithm is not optimal complexity-wise, as it has an Ackermannian worst-case running time [19, 20], whereas termination and boundedness for VASS are ExpSpace-complete [17, 22]. But it is conceptually simple, and

so amenable to other models. Karp and Miller's algorithm has been extended, into a so-called *reduced reachability tree*, to the general class of well-structured transition systems (WSTS), to which VASS belong [6, 7]. It has also recently been applied to VASS equipped with one pushdown stack [14]. These pushdown VASS are not WSTS, thus showing that Karp and Miller's algorithm can apply outside the realm of WSTS.

A well-known extension of pushdown automata is higher-order pushdown automata (or HOPDA for short), introduced in [18, 8, 1], in which stacks are replaced by higher-order stacks – an order n stack being a stack of order $(n-1)$ stacks, with an order 1 stack being a classical stack, and the operations on an order n stack being the copying of the topmost order $(n-1)$ stack on top of it, and its inverse operation. This model is interesting for modelling because of its equivalence to safe higher-order recursion schemes [11]. Furthermore, their transition graphs are exactly the graphs of the so-called prefix-recognisable hierarchy [5, 4], which are known to enjoy decidable MSO model-checking. As all the graphs of the hierarchy enjoy the same decidable properties, a tempting conjecture to make is that what holds for models with an order 1 auxiliary stack also holds for the same models with an order n auxiliary stack, for any order n. The starting point of the present work was thus the conjecture that Karp and Miller's algorithm would be a decision tool for termination and boundedness for higher-order pushdown VASS (or HOPVASS for short).

Contribution. Our contribution in this paper is twofold. We first show that termination, and therefore boundedness, are undecidable for HOPVASS by reducing from termination of Minsky counter machines through a stepwise simulation. We also show that the coverability problem (also known as the control-state reachability problem) is undecidable as well, through the same simulation. Our undecidability results hold even in the restricted subcase of one counter and an order 2 stack. This is in sharp contrast with the same model at order 1, for which boundedness and coverability are decidable [14, 16].

We then give a decidable criterion over sequences of higher-order stack operations which characterises which ones can be applicable arbitrarily many times. The detection of such sequences is crucial for the implementation of Karp and Miller's algorithm. Our criterion, which is decidable in quadratic time, makes Karp and Miller's approach implementable for HOPVASS, but the resulting procedure is only a semi-algorithm. It can find witnesses of non-termination or unboundedness, but it does not terminate in general because, contrary to WSTS and order 1 pushdown VASS, there might be infinite runs that contain no iterable factor. We provide an example that illustrates this fact. More interestingly, we prove, thanks to the same iterability criterion, that our semi-algorithm always terminates on HOPDA. This means that Karp and Miller's algorithm also applies to HOPDA, and thus provides a decision procedure that solves termination and boundedness for HOPDA.

Related work and discussion. The coverability and reachability problems for order 1 pushdown VASS are inter-reducible (in logspace) and Tower-hard [12, 13]. Their decidability status is still open. The boundedness problem for the same model is decidable, and its complexity is between Tower and Hyper-Ackermann [14]. For the subcase of only one counter, coverability is decidable [16] and boundedness is solvable in exponential time [15].

The main framework for our presentation comes from the description of regular sets of higher-order stacks from Carayol presented in [2, 3]. We borrow from it the notion of *reduced* sequence of operations as a short description of the effect of a sequence. Our criterion for iterability is a modification of that reduction notion, in which we aim to keep the domain of definition of the sequence (which is not stable through reduction). To solve this issue,

Carayol introduced *test operations*. Instead, we simply weaken the reduction by forbidding it to reduce *destructive tests* of the highest level, and considering a so-obtained *weak-reduced sequence* for every order. This iterability criterion is similar to the result of Parys in [21], in the sense that the underlying idea is that a sequence of operations is iterable if, and only if, it does not decrease the number of k-stack in the topmost $(k-1)$ stack, for every k. Otherwise, our presentation and our techniques are very different from those of Parys.

To our knowledge, termination and boundedness have never been directly studied on HOPDA. However, there are several existing works from which decidability of termination and boundedness for HOPDA could be easily derived. For example, in [9], Hague, Kochems and Ong study the downward closure of languages of HOPDA, and compute it by deciding the *simultaneous unboundedness problem* of their languages. It follows that finiteness of the language defined by a HOPDA is decidable. Termination and boundedness are easily reducible[1] to the latter problem.

2 Preliminaries

Higher-order pushdown automata. We consider a finite alphabet Σ. The set of *order 1 stacks* (or 1-stacks) over Σ is the set $\mathrm{Stacks}_1(\Sigma) = \Sigma^*$. We denote a 1-stack s as $s = [s_1 \dots s_{|s|}]_1$, where $|s|$ is the length of s, where $s_{|s|}$ is the *topmost letter* of s. The empty stack is denoted $[]_1$. For every $a \in \Sigma$, we define the operations $\mathtt{push}_a$ which adds an a at the top of a stack, and $\mathtt{pop}_a$ which removes the topmost letter of a stack if it is an a and is not applicable otherwise. Formally, $\mathtt{push}_a$ and $\mathtt{pop}_a$ are partial functions from $\mathrm{Stacks}_1(\Sigma)$ to $\mathrm{Stacks}_1(\Sigma)$, defined by $\mathtt{push}_a([s_1 \cdots s_{|s|}]_1) = [s_1 \cdots s_{|s|}a]_1$, and $\mathtt{pop}_a(s) = s'$ if and only if $\mathtt{push}_a(s') = s$. We define the set of order 1 operations $\mathtt{Op}_1(\Sigma) = \{\mathtt{push}_a, \mathtt{pop}_a \mid a \in \Sigma\}$. When Σ is understood, we omit it (we will do it from now on).

For $n > 1$, we define the set of *order n stacks* (or n-stacks) over Σ as $\mathrm{Stacks}_n = (\mathrm{Stacks}_{n-1})^+$. We denote an n-stack s as $s = [s_1 \dots s_{|s|}]_n$, where $s_{|s|}$ is the *topmost* $(n-1)$-*stack* of s. The stack $[[]_{n-1}]_n$ is denoted $[]_n$ for short, and abusively called the *empty stack*. We define the operations $\mathtt{copy}_n$ which copies the topmost $(n-1)$-stack on the top of the stack it is applied to, and $\overline{\mathtt{copy}}_n$ its inverse, i.e., it removes the topmost $(n-1)$-stack of a stack if it is equal to the one right below it, and is not applicable otherwise. Formally, $\mathtt{copy}_n$ and $\overline{\mathtt{copy}}_n$ are partial functions from Stacks_n to Stacks_n, defined by $\mathtt{copy}_n([s_1 \cdots s_{|s|}]_n) = [s_1 \cdots s_{|s|}s_{|s|}]_n$, and $\overline{\mathtt{copy}}_n(s) = s'$ if and only if $\mathtt{copy}_n(s') = s$. We define the set of order n operations $\mathtt{Op}_n = \{\mathtt{copy}_n, \overline{\mathtt{copy}}_n\} \cup \mathtt{Op}_{n-1}$ and we define the application of an operation θ of $\mathtt{Op}_{n-1}$ to an n-stack s as $\theta(s) = [s_1 \cdots s_{|s|-1}\theta(s_{|s|})]_n$. Given $\theta \in \mathtt{Op}_n$, we define $\bar{\theta}$ its inverse, i.e., $\overline{\mathtt{push}_a} = \mathtt{pop}_a$, $\overline{\mathtt{pop}_a} = \mathtt{push}_a$ and $\overline{\overline{\mathtt{copy}}_i} = \mathtt{copy}_i$. Finally, we inductively define the *topmost k-stack* of an n-stack $s = [s_1 \cdots s_{|s|}]_n$ as $\mathrm{top}_n(s) = s$, and $\mathrm{top}_k(s) = \mathrm{top}_k(s_{|s|})$ for $k < n$.

▶ **Example 1.** Assuming that $\Sigma = \{a, b\}$, we have $\mathtt{push}_a([[ab]_1[b]_1]_2) = [[ab]_1[ba]_1]_2$, $\mathtt{copy}_2([[[ab]_1]_2[[a]_1[b]_1]_2]_3) = [[[ab]_1]_2[[a]_1[b]_1[b]_1]_2]_3$, and $\overline{\mathtt{copy}}_2([[b]_1[a]_1]_2)$ is not defined.

An *order n pushdown automaton*, or *n-PDA* for short, or *HOPDA* if the order is left implicit, is a tuple $\mathcal{A} = (Q, q_{\mathrm{init}}, \Sigma, \Delta)$, where Q is a finite set of *states*, q_{init} is an *initial state*, Σ is a *stack alphabet* and $\Delta \subseteq Q \times \mathtt{Op}_n \times Q$ is a finite set of *transitions*. A transition

[1] For termination, simply make all transitions output a letter and make all states accepting, then observe that the resulting HOPDA terminates if, and only if, its language is finite. This observation follows from König's lemma together with the fact that HOPDA are finitely branching. For boundedness, add a new accepting state that the HOPDA may non-deterministically jump to, and from which it "dumps" the contents of the stack on the output tape. All original states are non-accepting and all original transitions are silent. It is readily seen that the resulting HOPDA is bounded if, and only if, its language is finite.

$(p, \theta, q) \in \Delta$ is also written as $p \xrightarrow{\theta} q$. A *configuration* of $\mathcal{A}$ is a pair (q, s), where $q \in Q$ and $s \in \mathrm{Stacks}_n$. The *initial configuration* is $(q_{\mathrm{init}}, []_n)$. A *step* of $\mathcal{A}$ is a triple $((p, s), \theta, (q, t))$, where (p, s) and (q, t) are configurations and θ is an operation, such that $p \xrightarrow{\theta} q$ and $t = \theta(s)$. Such a step is also written as $(p, s) \xrightarrow{\theta} (q, t)$. A *run* of $\mathcal{A}$ is an alternating sequence $(q_0, s_0), \theta_1, (q_1, s_1), \ldots, \theta_k, (q_k, s_k)$ of configurations (q_i, s_i) and operations θ_i, such that $(q_{i-1}, s_{i-1}) \xrightarrow{\theta_i} (q_i, s_i)$ for every $0 < i \leq k$. Such a run is also written as $(q_0, s_0) \xrightarrow{\theta_1} (q_1, s_1) \cdots \xrightarrow{\theta_k} (q_k, s_k)$, and it is called *initialised* when (q_0, s_0) is the initial configuration. The *reachability set* of $\mathcal{A}$ is the set of configurations (q, s) such that there is an initialised run in $\mathcal{A}$ that ends with (q, s).

▶ Remark. Instead of the $\overline{\mathtt{copy}}_n$ operation, the literature usually considers a $\mathtt{pop}_n$ operation that destroys the topmost $(n-1)$-stack (provided that there is one below it). Formally, $\mathtt{pop}_n([s_1 \cdots s_{|s|-1} s_{|s|}]_n) = [s_1 \cdots s_{|s|-1}]_n$ if $|s| > 1$ and is undefined otherwise. Following Carayol [2], we prefer the more symmetric operation $\overline{\mathtt{copy}}_n$ that destroys the topmost $(n-1)$-stack only if it is equal to the previous one.

Higher-order pushdown vector addition systems with states. We let $\mathbb{N}$ denote the set of natural numbers $\mathbb{N} = \{0, 1, \ldots\}$ and we let $\mathbb{Z}$ denote the set of integers $\mathbb{Z} = \{\ldots, -1, 0, 1, \ldots\}$. Consider a *dimension* $d \in N$ with $d > 0$. Given a set S and a vector $\mathbf{x}$ in S^d, we let $\mathbf{x}(c)$ denote the cth component of $\mathbf{x}$, i.e., $\mathbf{x} = (\mathbf{x}(1), \ldots, \mathbf{x}(d))$.

An *order n pushdown vector addition system with states of dimension d*, or *d-dim n-PVASS* for short, or *HOPVASS* if the order and the dimension are left implicit, is a tuple $\mathcal{S} = (Q, q_{\mathrm{init}}, \Sigma, \Delta)$, where Q is a finite set of *states*, q_{init} is an *initial state*, Σ is a *stack alphabet* and $\Delta \subseteq Q \times \mathbb{Z}^d \times \mathtt{Op}_n \times Q$ is a finite set of *transitions*. Vectors $\mathbf{a} \in \mathbb{Z}^d$ are called *actions*. A *configuration* of $\mathcal{S}$ is a triple $(q, \mathbf{x}, s)$, where $q \in Q$, $\mathbf{x} \in \mathbb{N}^d$ and $s \in \mathrm{Stacks}_n$. Intuitively, the dimension d is the number of counters, and $\mathbf{x}(1), \ldots, \mathbf{x}(d)$ are the values of these counters. The *initial configuration* is $(q_{\mathrm{init}}, \mathbf{0}, []_n)$. A *step* of $\mathcal{S}$ is a triple $(p, \mathbf{x}, s) \xrightarrow{\mathbf{a}, \theta} (q, \mathbf{y}, t)$, where $(p, \mathbf{x}, s)$ and $(q, \mathbf{y}, t)$ are configurations, $\mathbf{a}$ is an action and θ is an operation, such that $p \xrightarrow{\mathbf{a}, \theta} q$, $\mathbf{y} = \mathbf{x} + \mathbf{a}$ and[2] $t = \theta(s)$. The notions of *run*, *initialised run* and *reachability set* are defined in the same way as for n-PDA.

Coverability, termination and boundedness. We investigate in this paper three basic verification problems on HOPVASS, namely coverability, termination and boundedness. The *coverability problem* asks, given a HOPVASS $\mathcal{S}$ and a state q of $\mathcal{S}$, whether the reachability set of $\mathcal{S}$ contains a configuration whose state is q. The *termination problem* asks, given a HOPVASS $\mathcal{S}$, whether all initialised runs of a $\mathcal{S}$ are finite. The *boundedness problem* asks, given a HOPVASS $\mathcal{S}$, whether the reachability set of $\mathcal{S}$ is finite. Observe that HOPVASS are *finitely branching*, i.e., each configuration is the source of only finitely many steps. This entails that termination is Turing-reducible to boundedness for HOPVASS. Indeed, if the reachability set of a HOPVASS is infinite, then it necessarily has an infinite initialised run, by König's lemma (applied to its reachability tree). Otherwise, we may decide whether it has an infinite initialised run by exploring its reachability graph, which is finite and computable.

[2] The definition of configurations requires counters to be nonnegative. So the equality $\mathbf{y} = \mathbf{x} + \mathbf{a}$ carries the implicit condition that $\mathbf{x} + \mathbf{a} \geq \mathbf{0}$.

3 Undecidability of Coverability and Termination for 1-dim 2-PVASS

It is known that coverability is decidable for 1-dim 1-PVASS [16] and that termination and boundedness are decidable for 1-PVASS of arbitrary dimension [14]. We show in this section that all three problems are undecidable for HOPVASS, even in the restricted subcase of 1-dim 2-PVASS. Our proof proceeds by reduction from coverability and termination in Minsky counter machines, through a stepwise simulation of these machines by 1-dim 2-PVASS.

We use a non-standard presentation of Minsky counter machines (simply called counter machines in the sequel) that is close to VASS and more convenient for our purpose than the standard one. A *d-counter machine* is a triple $\mathcal{M} = (Q, q_{\text{init}}, \Delta)$, where Q is a finite set of *states*, q_{init} is an *initial state* and $\Delta \subseteq Q \times (\mathbb{Z} \cup \{\mathtt{T}\})^d \times Q$ is a finite set of *transitions*. Vectors $\mathbf{a} \in (\mathbb{Z} \cup \{\mathtt{T}\})^d$ are called *actions*. A *configuration* of $\mathcal{M}$ is a pair $(q, \mathbf{x})$, where $q \in Q$ and $\mathbf{x} \in \mathbb{N}^d$. The *initial configuration* is $(q_{\text{init}}, \mathbf{0})$. A *step* of $\mathcal{M}$ is a triple $(p, \mathbf{x}) \xrightarrow{\mathbf{a}} (q, \mathbf{y})$, where $(p, \mathbf{x})$ and $(q, \mathbf{y})$ are configurations and $\mathbf{a}$ is an action, such that $p \xrightarrow{\mathbf{a}} q$ and

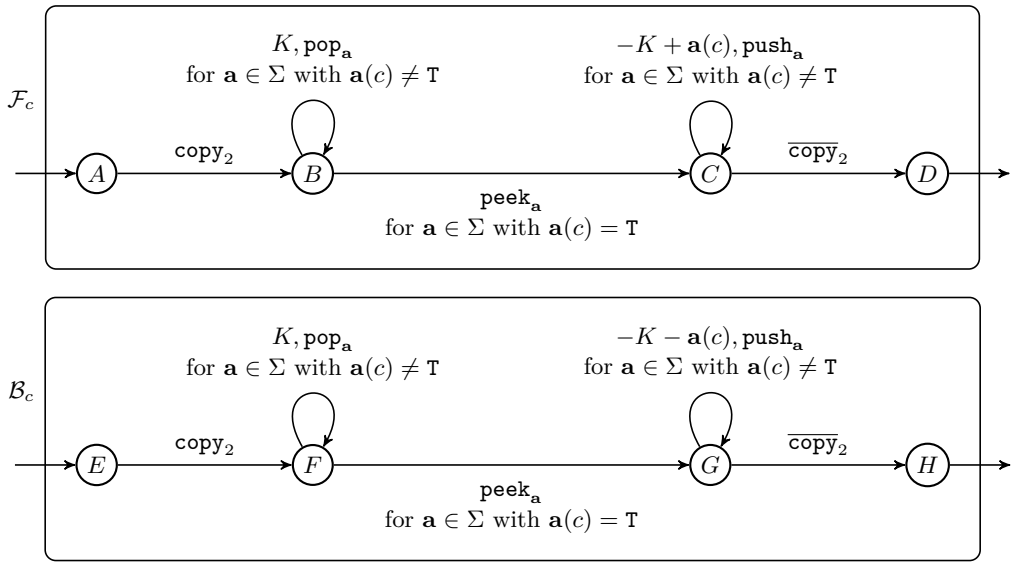
$$\begin{cases} \mathbf{y}(c) = \mathbf{x}(c) + \mathbf{a}(c) & \text{if } \mathbf{a}(c) \in \mathbb{Z} \\ \mathbf{y}(c) = \mathbf{x}(c) = 0 & \text{if } \mathbf{a}(c) = \mathtt{T} \end{cases} \tag{1}$$

for every counter $1 \leq c \leq d$. The notions of *run*, *initialised run* and *reachability set* are defined in the same way as for n-PDA. It is well-known that, for every $d \geq 2$, coverability, termination and boundedness are undecidable for d-counter machines.

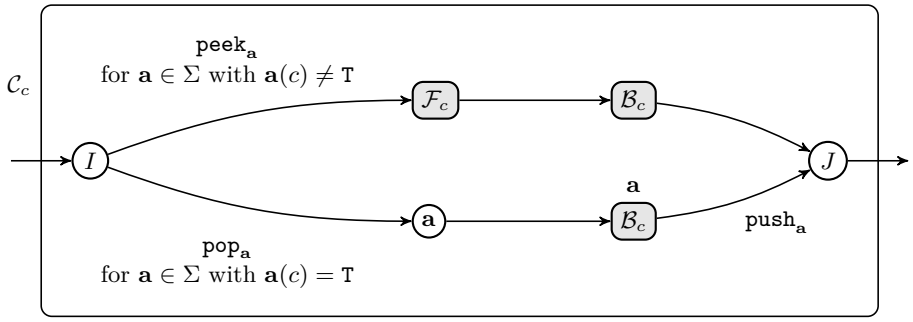
Our simulation of a d-counter machine $\mathcal{M}$ by a 1-dim 2-PVASS $\mathcal{S}$ roughly proceeds as follows. To prevent confusion between the counters of $\mathcal{M}$ and the counter of $\mathcal{S}$, we will denote the latter by κ. When $\mathcal{S}$ is idle, meaning that it is not simulating a step of $\mathcal{M}$, its counter κ is zero and its stack is of the form $[[(\mathtt{T}, \ldots, \mathtt{T})\mathbf{a}_1 \cdots \mathbf{a}_k]_1]_2$, where each $\mathbf{a}_i \in (\mathbb{Z} \cup \{\mathtt{T}\})^d$ is an action of $\mathcal{M}$. Intuitively, the word $w = \mathbf{a}_1 \cdots \mathbf{a}_k$, which we call the *history*, is the sequence of actions that $\mathcal{M}$ has taken to reach its current configuration. The vector $(\mathtt{T}, \ldots, \mathtt{T})$ acts as a bottom symbol. To simulate a step $(p, \mathbf{x}) \xrightarrow{\mathbf{a}} (q, \mathbf{y})$ of $\mathcal{M}$, $\mathcal{S}$ pushes $\mathbf{a}$ onto its stack, which becomes $[[(\mathtt{T}, \ldots, \mathtt{T})w\mathbf{a}]_1]_2$, and then it checks that its new history $w\mathbf{a}$ corresponds to a legal sequence of actions (starting from $\mathbf{0}$) with respect to Equation 1. To perform this check, $\mathcal{S}$ uses the history w and its counter κ to verify, for each counter $1 \leq c \leq d$, that $w\mathbf{a}$ is legal with respect to c. It can do so without destroying the history thanks to $\mathtt{copy}_2$ and $\overline{\mathtt{copy}}_2$ operations. When all checks are complete, $\mathcal{S}$ is again idle, its counter κ is zero and its stack is $[[(\mathtt{T}, \ldots, \mathtt{T})w\mathbf{a}]_1]_2$.

We now present our simulation of d-counter machines by 1-dim 2-PVASS in detail. We start with some additional notations. Given an action $\mathbf{a} \in (\mathbb{Z} \cup \{\mathtt{T}\})^d$, we let $\xrightarrow{\mathbf{a}}$ denote the binary relation on $\mathbb{N}^d$ defined by $\mathbf{x} \xrightarrow{\mathbf{a}} \mathbf{y}$ if Equation 1 holds. Given a word $w = \mathbf{a}_1 \cdots \mathbf{a}_k$ of actions $\mathbf{a}_i \in (\mathbb{Z} \cup \{\mathtt{T}\})^d$, we let $\xrightarrow{w}$ denote the binary relation on $\mathbb{N}^d$ defined by $\mathbf{x} \xrightarrow{w} \mathbf{y}$ if there exists $\mathbf{x}_0, \ldots, \mathbf{x}_k$ such that $\mathbf{x} = \mathbf{x}_0 \xrightarrow{\mathbf{a}_1} \mathbf{x}_1 \cdots \xrightarrow{\mathbf{a}_k} \mathbf{x}_k = \mathbf{y}$, with the convention that $\xrightarrow{\varepsilon}$ is the identity relation on $\mathbb{N}^d$. The notation $\mathbf{x} \xrightarrow{w}$ means that $\mathbf{x} \xrightarrow{w} \mathbf{y}$ for some $\mathbf{y}$. We define the *displacement* $\delta(w)$ of a word $w = \mathbf{a}_1 \cdots \mathbf{a}_k$ in $(\mathbb{Z}^d)^*$ by $\delta(w) = \mathbf{a}_1 + \cdots + \mathbf{a}_k$. Observe that, for such a word $w \in (\mathbb{Z}^d)^*$, it holds that $\mathbf{x} \xrightarrow{w} \mathbf{y}$ if, and only if, $\mathbf{x} + \delta(w) = \mathbf{y}$ and $\mathbf{x} + \delta(v) \geq \mathbf{0}$ for every prefix v of w.

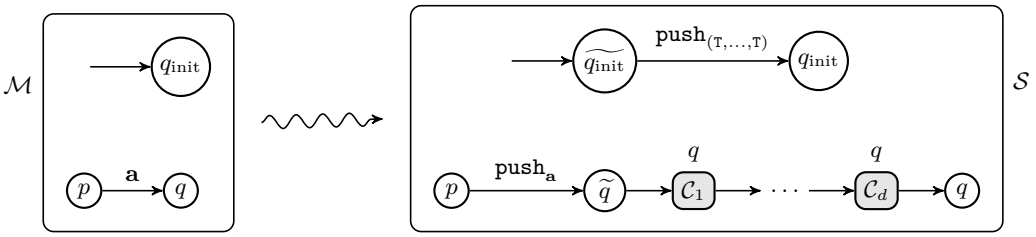
We extend the vector notation $\mathbf{a}(c)$ to sequences of actions $\mathbf{a}_1 \cdots \mathbf{a}_k \in ((\mathbb{Z} \cup \{\mathtt{T}\})^d)^*$ by letting $(\mathbf{a}_1 \cdots \mathbf{a}_k)(c)$ denote the word in $\mathbb{Z}^*$ defined by $(\mathbf{a}_1 \cdots \mathbf{a}_k)(c) = \mathbf{a}_1(c) \cdots \mathbf{a}_k(c)$. Note that for every $\mathbf{x}, \mathbf{y} \in \mathbb{N}^d$ and $w \in ((\mathbb{Z} \cup \{\mathtt{T}\})^d)^*$, it holds that $\mathbf{x} \xrightarrow{w} \mathbf{y}$ if, and only if, $\mathbf{x}(c) \xrightarrow{w(c)} \mathbf{y}(c)$ for every $1 \leq c \leq d$. Observe that the relation $\xrightarrow{w}$ is *forward-deterministic*,



(a) The gadgets $\mathcal{F}_c$ and $\mathcal{B}_c$ that apply, forward for $\mathcal{F}_c$ and backward for $\mathcal{B}_c$, the current history of the cth counter. The constant $K \in \mathbb{N}$ satisfies $|\mathbf{a}(c)| < K$ for every $\mathbf{a} \in \Sigma$ with $\mathbf{a}(c) \neq \mathtt{T}$.



(b) The gadget $\mathcal{C}_c$ that checks that the most recent action of the history is applicable for the cth counter.



(c) Translation of a d-counter machine $\mathcal{M}$ into a 1-dim 2-PVASS $\mathcal{S}$.

Figure 1 Simulation of a d-counter machine $\mathcal{M}$ by a 1-dim 2-PVASS $\mathcal{S}$. The stack alphabet $\Sigma \subseteq (\mathbb{Z} \cup \{\mathtt{T}\})^d$ is the finite set of actions of $\mathcal{M}$. Edges containing "for $\mathbf{a} \in \Sigma$ with φ" denote multiple transitions, one for each $\mathbf{a} \in \Sigma$ satisfying the condition φ.

i.e., $\mathbf{x} \xrightarrow{w} \mathbf{y} \wedge \mathbf{x} \xrightarrow{w} \mathbf{y}' \implies \mathbf{y} = \mathbf{y}'$. In a d-counter machine or 1-dim 2-PVASS, given two configurations α and β, we let $\alpha \xrightarrow{*} \beta$ denote the existence of a run from α to β.

We fix, for the remainder of this section, a d-counter machine $\mathcal{M} = (Q, q_{\text{init}}, \Delta)$. Let $\Sigma \subseteq (\mathbb{Z} \cup \{\mathtt{T}\})^d$ denote the set of actions of $\mathcal{M}$, formally, $\Sigma = \{\mathbf{a} \mid \exists p, q : p \xrightarrow{\mathbf{a}} q\}$. We build from $\mathcal{M}$ a 1-dim 2-PVASS $\mathcal{S}$ with stack alphabet Σ. To simplify the presentation,

we introduce, for every $\mathbf{a} \in \Sigma$, a new order 1 operation $\mathtt{peek}_{\mathbf{a}} = \mathtt{push}_{\mathbf{a}} \circ \mathtt{pop}_{\mathbf{a}}$ that tests, without changing the stack, that the topmost letter of the stack is an $\mathbf{a}$, and is not applicable otherwise. The addition of these $\mathtt{peek}_{\mathbf{a}}$ operations has no impact on the decidability status of coverability, termination and boundedness for 1-dim 2-PVASS.

We present the 1-dim 2-PVASS $\mathcal{S}$ that simulates the d-counter machine $\mathcal{M}$ in a "bottom up" fashion. Recall that κ denotes the counter of $\mathcal{S}$. We start with two gadgets $\mathcal{F}_c$ and $\mathcal{B}_c$, where $1 \leq c \leq d$, that apply on κ, forward for $\mathcal{F}_c$ and backward for $\mathcal{B}_c$, the current history of the cth counter of $\mathcal{M}$. More precisely, they apply the displacement of the suffix v of the history that starts after the most recent zero-test on c. These gadgets are depicted in Figure 1a. Let us explain the behaviour of $\mathcal{F}_c$. We ignore K for the moment. Firstly, the $\mathtt{copy}_2$ from A to B copies the history so that it can be restored before leaving the gadget. The loop on B together with the transition from B to C locates the most recent zero-test on c in the history. The loop on C guesses the suffix v of the history and replays $v(c) \in \mathbb{Z}^*$ on the counter κ. Lastly, the $\overline{\mathtt{copy}}_2$ from C to D ensures that the guesses made in state C are correct and restores the stack to its original contents before entering the gadget. The increments by K in the loop on B are matched by decrements by K in the loop on C. So they do not change the global displacement realised by $\mathcal{F}_c$, which is $\delta(v(c))$. Their purpose is to ensure that the loop on C can be taken only finitely many times. This is crucial for termination. The behaviour of $\mathcal{B}_c$ is identical to that of $\mathcal{F}_c$ except that when $v(c) \in \mathbb{Z}^*$ is replayed on the counter κ, the opposite of each action is applied instead of the action itself. So the global displacement realised by $\mathcal{B}_c$ is $-\delta(v(c))$. The following lemma shows that $\mathcal{F}_c$ and $\mathcal{B}_c$ behave as expected. All proofs of this section can be found in Appendix A.

▶ **Lemma 2.** *Let $x, y \in \mathbb{N}$ and $s, t \in \mathrm{Stacks}_2$. Assume that $s = [[u\mathbf{b}v]_1]_2$ where $\mathbf{b} \in \Sigma$ and $u, v \in \Sigma^*$ are such that $\mathbf{b}(c) = \mathtt{T}$ and $v(c) \in \mathbb{Z}^*$. Then the following assertions hold:*

- *$(A, x, s) \xrightarrow{*} (D, y, t)$ in $\mathcal{F}_c$ if, and only if, $s = t$ and $x + \delta(v(c)) = y$,*
- *$(E, x, s) \xrightarrow{*} (H, y, t)$ in $\mathcal{B}_c$ if, and only if, $s = t$ and $x - \delta(v(c)) = y$.*

Our next gadget, the 1-dim 2-PVASS $\mathcal{C}_c$, where $1 \leq c \leq d$, is depicted in Figure 1b. It uses the gadgets $\mathcal{F}_c$ and $\mathcal{B}_c$ as subsystems. It is understood that each action $\mathbf{a} \in \Sigma$ with $\mathbf{a}(c) = \mathtt{T}$ induces a distinct copy of $\mathcal{B}_c$ in $\mathcal{C}_c$. Provided that $\kappa = 0$ and that w is a legal sequence of actions $\mathbf{0} \xrightarrow{w} \mathbf{x}$, the gadget $\mathcal{C}_c$ checks that the most recent action $\mathbf{a}$ of the history $w\mathbf{a}$ is applicable for the cth counter of $\mathcal{M}$, i.e., that $\mathbf{x}(c) \xrightarrow{\mathbf{a}(c)}$. If $\mathbf{a}(c) \in \mathbb{Z}$ then $\mathcal{C}_c$ goes through $\mathcal{F}_c$, which checks that $\mathbf{x}(c) + \mathbf{a}(c) \geq 0$ and exits with $\kappa = \mathbf{x}(c) + \mathbf{a}(c)$, and then it goes through $\mathcal{B}_c$, which reverts the changes that $\mathcal{F}_c$ did on κ. If $\mathbf{a}(c) = \mathtt{T}$ then $\mathcal{C}_c$ pops $\mathbf{a}$ and then goes through $\mathcal{B}_c$, which checks that $\mathbf{x}(c) \leq 0$ (hence, $\mathbf{x}(c) = 0$) and exits with $\kappa = 0$, and then pushes $\mathbf{a}$ back. In both cases, κ and the stack are restored to their original contents before entering the gadget. The following lemma shows that $\mathcal{C}_c$ behaves as expected.

▶ **Lemma 3.** *Let $y \in \mathbb{N}$ and $s, t \in \mathrm{Stacks}_2$. Assume that $s = [[(\mathtt{T}, \ldots, \mathtt{T})w\mathbf{a}]_1]_2$ where $\mathbf{a} \in \Sigma$ and $w \in \Sigma^*$ are such that $\mathbf{0} \xrightarrow{w}$. Then $(I, 0, s) \xrightarrow{*} (J, y, t)$ in $\mathcal{C}_c$ if, and only if, $y = 0$, $s = t$ and $0 \xrightarrow{w(c)\mathbf{a}(c)}$.*

We are now ready to present our translation of the d-counter machine $\mathcal{M}$ into an "equivalent" 1-dim 2-PVASS $\mathcal{S}$. The translation is depicted in Figure 1c, and corresponds to the informal description of $\mathcal{S}$ that followed the definition of d-counter machines. It is understood that each state q of $\mathcal{M}$ induces distinct copies of $\mathcal{C}_1, \ldots, \mathcal{C}_d$ in $\mathcal{S}$. The initial state of $\mathcal{S}$ is $\widetilde{q_{\mathrm{init}}}$. We need a few additional notations to prove that this translation is correct in the sense that it preserves coverability and termination. Given $\mathbf{x} \in \mathbb{N}^d$ and $s \in \mathrm{Stacks}_2$,

we let $\mathbf{x} \bowtie s$ denote the existence of $w \in \Sigma^*$ such that $\mathbf{0} \xrightarrow{w} \mathbf{x}$ and $s = [[(\mathtt{T}, \dots, \mathtt{T})w]_1]_2$. Given two configurations $(\widetilde{q}, x, s)$ and (q, y, t) of $\mathcal{S}$, where $q \in Q$ is a state of $\mathcal{M}$, we let $(\widetilde{q}, x, s) \rightsquigarrow (q, y, t)$ denote a run in $\mathcal{S}$ from $(\widetilde{q}, x, s)$ to (q, y, t) with no intermediate state in Q. Put differently, such a run is the concatenation of runs of $\mathcal{C}_1, \dots, \mathcal{C}_d$ except for its first and last steps (see Figure 1c). The correctness of the translation of $\mathcal{M}$ into $\mathcal{S}$ is shown by the following two lemmas. Lemma 4 shows that $\bowtie$ induces a "weak simulation" relation from $\mathcal{M}$ to $\mathcal{S}$. This lemma is used to show that every (possibly infinite) initialised run of $\mathcal{M}$ can be translated into an initialised run of $\mathcal{S}$. Lemma 6 shows that the subsystem $\widetilde{q} \to \mathcal{C}_1 \to \cdots \to \mathcal{C}_d \to q$ of $\mathcal{S}$ works as expected, in that it correctly checks that the most recent action of the history is applicable provided that the previous actions of the history are applicable. This lemma is used to show that every (possibly infinite) initialised run of $\mathcal{S}$ can be translated back into an initialised run of $\mathcal{M}$.

▶ **Lemma 4.** *Let $\mathbf{x} \in \mathbb{N}^d$ and $s \in \mathrm{Stacks}_2$ such that $\mathbf{x} \bowtie s$. For every step $(p, \mathbf{x}) \xrightarrow{\mathbf{a}} (q, \mathbf{y})$ in $\mathcal{M}$, there exists a run $(p, 0, s) \xrightarrow{\mathtt{push}_{\mathbf{a}}} (\widetilde{q}, 0, t) \rightsquigarrow (q, 0, t)$ in $\mathcal{S}$ with $\mathbf{y} \bowtie t$.*

▶ **Corollary 5.** *For every initialised run $(q_0, \mathbf{x}_0) \xrightarrow{\mathbf{a}_1} (q_1, \mathbf{x}_1) \cdots \xrightarrow{\mathbf{a}_k} (q_k, \mathbf{x}_k) \cdots$ in $\mathcal{M}$, there is an initialised run $(\widetilde{q_{\mathrm{init}}}, 0, []_2) \xrightarrow{\mathtt{push}_{(\mathtt{T},\dots,\mathtt{T})}} (q_0, 0, s_0) \xrightarrow{\mathtt{push}_{\mathbf{a}_1}} (\widetilde{q_1}, 0, s_1) \rightsquigarrow (q_1, 0, s_1) \cdots \xrightarrow{\mathtt{push}_{\mathbf{a}_k}} (\widetilde{q_k}, 0, s_k) \rightsquigarrow (q_k, 0, s_k) \cdots$ in $\mathcal{S}$.*

▶ **Lemma 6.** *Assume that $t = [[(\mathtt{T}, \dots, \mathtt{T})w\mathbf{a}]_1]_2$ where $\mathbf{a} \in \Sigma$ and $w \in \Sigma^*$ are such that $\mathbf{0} \xrightarrow{w}$. For every run $(\widetilde{q}, 0, t) \rightsquigarrow (q, x, s)$, it holds that $x = 0$, $s = t$ and $\mathbf{0} \xrightarrow{w\mathbf{a}}$.*

▶ **Corollary 7.** *Every initialised run of $\mathcal{S}$ that is infinite or ends with a configuration whose state is in Q, is of the form $(\widetilde{q_{\mathrm{init}}}, 0, []_2) \xrightarrow{\mathtt{push}_{(\mathtt{T},\dots,\mathtt{T})}} (q_0, 0, s_0) \xrightarrow{\mathtt{push}_{\mathbf{a}_1}} (\widetilde{q_1}, 0, s_1) \rightsquigarrow (q_1, 0, s_1) \cdots \xrightarrow{\mathtt{push}_{\mathbf{a}_k}} (\widetilde{q_k}, 0, s_k) \rightsquigarrow (q_k, 0, s_k) \cdots$ with $q_i \in Q$. Moreover, for every such run in $\mathcal{S}$, there is an initialised run $(q_0, \mathbf{x}_0) \xrightarrow{\mathbf{a}_1} (q_1, \mathbf{x}_1) \cdots \xrightarrow{\mathbf{a}_k} (q_k, \mathbf{x}_k) \cdots$ in $\mathcal{M}$.*

An immediate consequence of Corollaries 5 and 7 is that the coverability and termination problems for d-counter machines are many-one reducible to the coverability and termination problems for 1-dim 2-PVASS, respectively. Since coverability and termination are undecidable for 2-counter machines, they are also undecidable for 1-dim 2-PVASS. Moreover, as mentioned in Section 2, termination is Turing-reducible to boundedness for 1-dim 2-PVASS, since they are finitely branching. We have shown the following theorem.

▶ **Theorem 8.** *The coverability problem, the termination problem and the boundedness problem are undecidable for 1-dim 2-PVASS.*

▶ Remark. Theorem 8 also holds for 1-dim 2-PVASS defined with $\mathtt{pop}_2$ operations instead of $\overline{\mathtt{copy}}_2$ operations. Indeed, we may replace the gadgets $\mathcal{F}_c$ and $\mathcal{B}_c$ by "equivalent" ones using $\mathtt{pop}_2$ and no $\overline{\mathtt{copy}}_2$. Intuitively, instead of guessing and replaying in state C the suffix of the history, $\mathcal{F}'_c$ copies the history twice. Each loop uses a fresh copy of the history and then destroys this copy with a $\mathtt{pop}_2$. Both loops use $\mathtt{pop}_{\mathbf{a}}$ operations to browse through the history (backwards). The construction of $\mathcal{B}'_c$ is similar. The new gadgets $\mathcal{F}'_c$ and $\mathcal{B}'_c$ also satisfy Lemma 2, and it follows that the resulting 1-dim 2-PVASS $\mathcal{S}'$ also simulates the d-counter machine $\mathcal{M}$.

4 Iterability of Operation Sequences

In this section, we show that we can characterise exactly the sequences of operations which can be applied arbitrarily many times to a given stack. This result is used in the next section to provide a semi-decision procedure for the non-boundedness and non-termination problems for HOPVASS, using the Karp and Miller reduced tree (as used in [14]).

We consider sequences of operations in $\mathtt{Op}_n$, called n-blocks for short. Given $\rho = \theta_1 \cdots \theta_m \in \mathtt{Op}_n^*$, we identify it with the partial function $\rho = \theta_m \circ \theta_{m-1} \circ \cdots \circ \theta_1$. We denote by $\mathrm{dom}(\rho)$ the set of n-stacks s such that $\rho(s)$ is defined. We define $\bar{\rho} = \overline{\theta_m} \cdots \overline{\theta_1}$, and observe that $\bar{\rho}$ is the partial inverse of ρ. We want to characterise n-blocks which are *iterable*, i.e., which can be applied arbitrarily many times to a given stack.

▶ **Definition 9.** An n-block ρ is *iterable* on a stack s if for all i, $s \in \mathrm{dom}(\rho^i)$.

To investigate iterability, we are interested in the global effect of an n-block while keeping track of its condition of application. We thus need a normal form of sequences which keeps track of these two things, and a criterion on this normal form to determine if it is iterable or not. Following Carayol [2, 3], we say that an n-block is *reduced* if it does not contain any factor of the form $\theta\bar{\theta}$ with $\theta \in \mathtt{Op}_n$ (in [2], they are called minimal, most details come from [3]). Given an n-block ρ, we let $\mathrm{red}(\rho)$ denote the unique reduced operation sequence obtained from ρ by recursively removing $\theta\bar{\theta}$ factors. It is immediate to observe that $\mathrm{red}(\rho)(s) = \rho(s)$ for every stack $s \in \mathrm{dom}(\rho)$. In particular, $\mathrm{dom}(\rho) \subseteq \mathrm{dom}(\mathrm{red}(\rho))$. Intuitively, $\mathrm{red}(\rho)$ is the minimal n-block performing the transformation performed by ρ, in the sense that it does not contain a factor which does not modify the stack. The reduced n-block is thus a good normal form for determining the global effect of an n-block. However, reducing an n-block may yield an n-block with a wider domain, e.g., $\mathtt{pop}_a\mathtt{push}_a$ is only applicable to stacks whose topmost symbol is an a, while its reduced n-block is ε which is applicable to all stacks.

Thus, red is not a good tool to investigate iterability. We need to preserve the domain of applicability of an n-block, while getting rid of factors that are always applicable and do not modify the stack. To do this, Carayol adds test operations in the normal form of regular sets of n-blocks, at the expense of augmenting the number of operations and having no real normal form for n-blocks themselves as some tests may be redundant and not easy to eliminate. We propose here a different approach which consists in associating to each n-block, n normal blocks, one for every order. Each keeps track of the destructive operations of its order the original n-block has to perform to be applicable, while getting rid of factors which do not modify the stack and do not restrict the domain of application, and reducing the block in the classical sense for lower orders. To this end, we define a weaker variant of reduction, $\overline{\mathrm{red}}_n$, which does not remove factors of the form $\overline{\mathtt{copy}}_n\mathtt{copy}_n$ but is otherwise identical to red. The idea will thus be to consider for an n-block ρ, all its weak reduced blocks at every order: ρ will be iterable if, and only if, for every k, $\overline{\mathrm{red}}_k(\rho)$ is iterable. Furthermore, it will be possible to check syntactically whether $\overline{\mathrm{red}}_k(\rho)$ is iterable or not.

The *restriction* of an n-block ρ to an order k, written $\rho|_k$, is the k-block obtained by removing every operation of order strictly higher than k in ρ, e.g., $(\mathtt{push}_a\mathtt{copy}_2\mathtt{push}_b)|_1 = \mathtt{push}_a\mathtt{push}_b$.

▶ **Definition 10.** Given an n-block ρ and an order $k \leq n$, we call $\overline{\mathrm{red}}_k(\rho)$ the only k-block obtained from $\rho|_k$ by applying the following rewriting system:

$$\theta\bar{\theta} \to \varepsilon, \text{ for } \theta \in \mathtt{Op}_k \backslash \{\overline{\mathtt{copy}}_k\} \text{ if } k > 1, \text{ and } \theta \in \{\mathtt{push}_a \mid a \in \Sigma\} \text{ if } k = 1.$$

Observe that this rewriting system is confluent, as is the classical reduction rewriting system. Therefore, $\overline{\mathrm{red}}_k(\rho)$ can be computed in linear time, by always reducing the leftmost factor first.

The following theorem shows that the domain of an n-block ρ is equal to the intersection of the domains of the weak reductions of ρ of every order. This implies that $\overline{\mathrm{red}}_1(\rho), \cdots, \overline{\mathrm{red}}_n(\rho)$ is indeed a good normal form of ρ in the sense it describes entirely the effect of ρ in a canonical way, and it preserves its domain of definition (contrary to $\mathrm{red}(\rho)$).

Intuitively, this result comes from the fact than whenever a reduction step (in red) enlarges the domain of applicability of ρ, it is due to the removal of a factor of the form $\overline{\mathtt{copy}}_k\mathtt{copy}_k$ (or $\mathtt{pop}_a\mathtt{push}_a$). As such factors are left intact in $\overline{\mathrm{red}}_k(\rho)$, a stack added to the domain of $\mathrm{red}(\rho)$ in this way is not added to the domain of $\overline{\mathrm{red}}_k(\rho)$.

▶ **Theorem 11.** *For every n-stack s and n-block ρ, $s \in \mathrm{dom}(\rho)$ if, and only if, for every $k \leq n$, $s \in \mathrm{dom}(\overline{\mathrm{red}}_k(\rho))$.*

Proof. ($\Rightarrow$) Suppose $s \in \mathrm{dom}(\rho)$. By definition of application, $s \in \mathrm{dom}(\rho|_k)$ for every k. We observe that $\mathrm{dom}(\rho|_k) \subseteq \mathrm{dom}(\overline{\mathrm{red}}_k(\rho))$ as every reduction step cannot restrict the domain of application. It follows that $s \in \mathrm{dom}(\overline{\mathrm{red}}_k(\rho))$.

($\Leftarrow$) For the sake of simplicity, in the following we suppose that when we reduce a $\overline{\mathtt{copy}}_k\mathtt{copy}_k$, $\overline{\mathtt{copy}}_k$ could not be matched with some $\mathtt{copy}_k$ at its left and similarly for $\mathtt{copy}_k$ at its right (w.l.o.g., as the system is confluent).

We show that for every weak reduction step, either $\mathrm{dom}(\rho)$ is not modified, either it is increased, but every stack added to it is not in one of the $\mathrm{dom}(\overline{\mathrm{red}}_k(\rho))$:

- If $\rho = \rho_1\mathtt{copy}_k\overline{\mathtt{copy}}_k\rho_2$ for $k \leq n$ (resp. $\rho_1\mathtt{push}_a\mathtt{pop}_a\rho_2$), then $\mathrm{dom}(\rho) = \mathrm{dom}(\rho_1\rho_2)$.
- If $\rho = \rho_1\overline{\mathtt{copy}}_k\mathtt{copy}_k\rho_2$ for $k < n$ (resp. $\rho_1\mathtt{pop}_a\mathtt{push}_a\rho_2$), then for every stack s in $\mathrm{dom}(\rho_1\rho_2)\backslash\mathrm{dom}(\rho)$, we get that $\rho_1(s) \notin \mathrm{dom}(\overline{\mathtt{copy}}_k)$ (resp. $\rho_1(s) \notin \mathrm{dom}(\mathtt{pop}_a)$). As $\overline{\mathrm{red}}_k(\rho) = \overline{\mathrm{red}}_k(\rho_1)\overline{\mathtt{copy}}_k\mathtt{copy}_k\overline{\mathrm{red}}_k(\rho_2)$ (resp. $\overline{\mathrm{red}}_1(\rho_1)\mathtt{pop}_a\mathtt{push}_a\overline{\mathrm{red}}_1(\rho_2)$) and $\overline{\mathrm{red}}_k(\rho_1)(s) = \rho_1|_k(s)$, we get that $s \notin \mathrm{dom}(\overline{\mathrm{red}}_k(\rho))$.

Therefore, by induction on the weak reduction steps of ρ, if $s \in \mathrm{dom}(\overline{\mathrm{red}}_n(\rho))\backslash\mathrm{dom}(\rho)$ then $s \notin \mathrm{dom}(\overline{\mathrm{red}}_k(\rho))$ for some $k < n$. As furthermore for every $k \leq n$, $s \in \mathrm{dom}(\rho)$ implies that $s \in \mathrm{dom}(\rho|_k)$, and therefore that $s \in \mathrm{dom}(\overline{\mathrm{red}}_k(\rho))$, we get the result. ◀

The rest of this subsection is devoted to proving that it is decidable whether an n-block is iterable on some stack or not. The decision algorithm is based on the observation that when ρ is iterable then for every k, $\overline{\mathrm{red}}_k(\rho)$ can be written as $\overline{\rho_{E_k}}\rho_{I_k}\rho_{E_k}$ and ρ_{I_k} does not contain $\overline{\mathtt{copy}}_k$ (see Theorem 15). Thus, if we iterate ρ, at each level, the accumulated effect will only be the accumulated effect of ρ_{I_k}, as ρ_{E_k} and $\overline{\rho_{E_k}}$ cancel each other. We show that ρ is iterable if, and only if, this accumulated effect does not decrease the "size of the stack" at any level, i.e., ρ_{I_k} does not contain $\overline{\mathtt{copy}}_k$ for any k. The proof, if rather technical in its formulation, only relies on the definition of the weak reduction and the two following auxiliary lemmas, the second being proven in Appendix B.

▶ **Lemma 12** ([3], Lemme 4.1.7)**.** *For every n-block ρ, $\mathrm{red}(\rho) = \varepsilon$ if, and only if, there is a stack s such that $s = \rho(s)$.*

▶ **Lemma 13.** *For every n-block ρ and order k, if $\overline{\mathrm{red}}_k(\rho)$ contains a factor of the form $\mathtt{copy}_k\rho'\overline{\mathtt{copy}}_k$ (or $\mathtt{push}_a\rho'\mathtt{pop}_b$ if $k = 1$), then $\mathrm{dom}(\rho) = \emptyset$.*

▶ **Corollary 14.** *For every n-block ρ and order k, if $\overline{\mathrm{red}}_k(\rho)$ is of the form $\overline{\rho_1}\rho_2\rho_1$ with ρ_1 containing a $\overline{\mathtt{copy}}_k$ (or a $\mathtt{pop}_a$ if $k = 1$), then $\mathrm{dom}(\rho) = \emptyset$.*

Proof. If ρ_1 contains a $\overline{\texttt{copy}}_k$ (resp. $\texttt{pop}_a$), then $\overline{\rho_1}$ contains a $\texttt{copy}_k$ (resp. $\texttt{push}_a$), therefore, $\overline{\text{red}}_k(\rho)$ contains a factor of the form $\texttt{copy}_k\rho'\overline{\texttt{copy}}_k$ (resp. $\texttt{push}_a\rho'\texttt{pop}_a$). ◀

▶ **Theorem 15.** *Given a stack s and an n-block ρ, ρ is iterable on s if, and only if, $s \in \text{dom}(\rho)$ and for every $k \leq n$, $\overline{\text{red}}_k(\rho)$ is of the form $\overline{\rho_{E_k}}\rho_{I_k}\rho_{E_k}$ with ρ_{I_k} containing no $\overline{\texttt{copy}}_k$ (or no $\texttt{pop}_a$ if $k = 1$).*

Proof. ($\Rightarrow$) We only do the proof for $k \geq 2$. The case for $k = 1$ is similar. It suffices to replace $\texttt{copy}_k$ and $\overline{\texttt{copy}}_k$ with $\texttt{push}_a$ and $\texttt{pop}_a$. Suppose that ρ is iterable on a stack s, i.e., for every i, $s \in \text{dom}(\rho^i)$. Fix k and let ρ_{E_k} be the largest suffix of $\overline{\text{red}}_k(\rho)$ such that $\overline{\text{red}}_k(\rho) = \overline{\rho_{E_k}}\rho_{I_k}\rho_{E_k}$. Notice that this choice of ρ_{E_k} implies that $\overline{\text{red}}_k(\rho^2_{I_k}) = \rho^2_{I_k}$. We prove by way of contradiction that so does ρ_{I_k}. Suppose now that ρ_{I_k} contains an occurrence of $\overline{\texttt{copy}}_k$. From Lemma 13, we get that $\rho_{I_k} = \rho_1\overline{\texttt{copy}}_k\rho_2$ with ρ_1 containing no order k operation. Suppose that ρ_2 contains a $\texttt{copy}_k$. Lemma 13 entails that $\rho_2 = \rho_3\texttt{copy}_k\rho_4$ and ρ_4 contains no order k operation. By maximality of ρ_{E_k}, we get $\overline{\text{red}}_k(\rho^2) = \overline{\rho_{E_k}}\rho_1\overline{\texttt{copy}}_k\rho_3\texttt{copy}_k\rho_4\rho_1\overline{\texttt{copy}}_k\rho_3\texttt{copy}_k\rho_4\rho_{E_k}$. As $s \in \text{dom}(\rho^2)$, by the definition of application, we get that $(\rho_4\rho_1)(s') = s'$, for $s' = (\overline{\rho_{E_k}}\rho_1\overline{\texttt{copy}}_k\rho_3\texttt{copy}_k)(s)$. From Lemma 12, we get $\text{red}(\rho_4\rho_1) = \varepsilon$. As it contains no order k operations, we also have that $\overline{\text{red}}_k(\rho_4\rho_1) = \varepsilon$. But then, we obtain that $\overline{\text{red}}_k(\rho^2_{I_k}) = \overline{\text{red}}_k(\rho_1\overline{\texttt{copy}}_k\rho_3\texttt{copy}_k\rho_4\rho_1\overline{\texttt{copy}}_k\rho_3\texttt{copy}_k\rho_4) = \overline{\text{red}}_k(\rho_1\overline{\texttt{copy}}_k\rho_3\texttt{copy}_k\rho_4\rho_1\overline{\texttt{copy}}_k\rho_3\texttt{copy}_k\rho_4) = \overline{\text{red}}_k(\rho_1\overline{\texttt{copy}}_k\rho_3\texttt{copy}_k\overline{\texttt{copy}}_k\rho_3\texttt{copy}_k\rho_4)$ and $\overline{\text{red}}_k(\rho^2_{I_k}) = \overline{\text{red}}_k(\rho_1\overline{\texttt{copy}}_k\rho_3\rho_3\texttt{copy}_k\rho_4)$ so that $\overline{\text{red}}_k(\rho^2_{I_k}) \neq \rho^2_{I_k}$. This is in contradiction with the remark we made above. Therefore, ρ_2 does not contain any $\texttt{copy}_k$.

Thus, ρ_{I_k} contains some $\overline{\texttt{copy}}_k$ and does not contain any $\texttt{copy}_k$. Therefore for every s', $\text{top}_k(\rho_{I_k}(s'))$ has strictly less $(k-1)$-stacks than s' and ρ_{I_k} is only applicable a finite number of times to all stacks. From Theorem 11, as $s \in \text{dom}(\rho^i)$ for all i, we get that $s \in \text{dom}(\overline{\text{red}}_k(\rho)^i)$ for all i, and therefore $\overline{\rho_{E_k}}(s) \in \text{dom}(\rho^i_{I_k})$, for all i. We thus get a contradiction. Therefore, for every k, ρ_{I_k} contains no $\overline{\texttt{copy}}_k$ operation.

($\Leftarrow$) We proceed by induction on the order n. Let us consider a 1-block ρ and a stack $s \in \text{dom}(\rho)$ such that $\overline{\text{red}}_1(\rho) = \overline{\rho_{E_1}}\rho_{I_1}\rho_{E_1}$, with ρ_{I_1} containing no $\texttt{pop}_a$ with $a \in \Sigma$. As $s \in \text{dom}(\rho)$, Corollary 14 shows that ρ_{E_1} contains no $\texttt{pop}_a$ with $a \in \Sigma$. As a consequence, we have that $\text{dom}(\rho_{I_1}) = \text{dom}(\rho_{E_1}) = \text{Stacks}_1$. Therefore $\overline{\rho_{E_1}}(s) \in \text{dom}(\rho^i_{I_1}\rho_{E_1})$ for all i, and $\overline{\rho_{E_1}}(s)$ is defined as $s \in \text{dom}(\rho)$. As $\overline{\text{red}}_1(\rho^i) = \overline{\rho_{E_1}}\rho^i_{I_1}\rho_{E_1}$, we get $s \in \text{dom}(\overline{\text{red}}_1(\rho^i))$ for all i. Using Theorem 11, we get that ρ is iterable on s.

Suppose now that the property holds for $(n-1)$-blocks, and consider an n-block ρ and a stack $s \in \text{dom}(\rho)$ such that for all $k \leq n$, $\overline{\text{red}}_k(\rho) = \overline{\rho_{E_k}}\rho_{I_k}\rho_{E_k}$ with ρ_{I_k} containing no $\overline{\texttt{copy}}_k$. From Corollary 14, we get that ρ_{E_k} contains no $\overline{\texttt{copy}}_k$ as well. Let us show that $s \in \text{dom}(\overline{\text{red}}_n(\rho^i))$ for all i. By hypothesis of induction, $\rho|_{n-1}$ is iterable on s, thus $s \in \text{dom}((\rho|_{n-1})^i)$ for all i. Observe $(\rho|_{n-1})^i = \rho^i|_{n-1}$. Therefore, $s \in \text{dom}((\overline{\text{red}}_n(\rho^i)|_{n-1}))$ for all i, as the latter can be obtained from $\rho^i|_{n-1}$ by applying reduction steps. As $s \in \text{dom}(\rho)$, $\rho_{E_n}(s)$ is defined, we deduce that $\overline{\rho_{E_n}}(s) \in \text{dom}((\rho^i_{I_n}\rho_{E_n})|_{n-1})$ for all i. Given an n-block ρ' containing no $\overline{\texttt{copy}}_n$, as $\texttt{copy}_n$ is applicable to all stacks, we get that for all s', if $s' \in \text{dom}(\rho'|_{n-1})$, then $s' \in \text{dom}(\rho')$. Thus $\overline{\rho_{E_n}}(s) \in \text{dom}(\rho^i_{I_n}\rho_{E_n})$ for all i, which entails $s \in \text{dom}(\overline{\text{red}}_n(\rho^i))$ for all i. As by hypothesis of induction, $s \in \text{dom}(\overline{\text{red}}_k(\rho^i))$ for all i and $k < n$, by Theorem 11, we deduce that ρ is iterable on s. ◀

An important remark which stems from the characterisation of Theorem 15 is that a block is either applicable only finitely many times to every stack, or it is applicable arbitrarily many times to every stack on which it can be applied once.

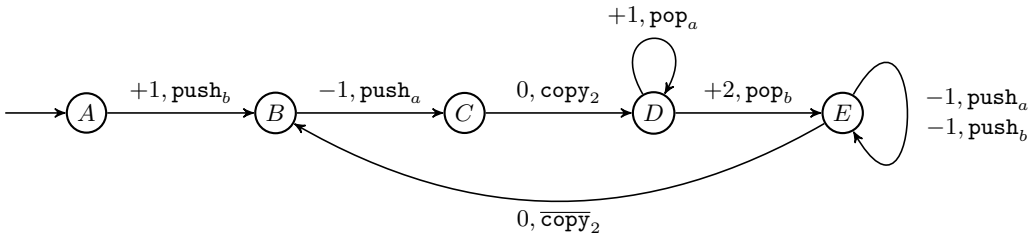


Figure 2 A 1-dim 2-PVASS whose reduced reachability tree is infinite.

5 Testing Non-Termination and Unboundedness

In this section, we use the characterisation of iterable blocks of Theorem 15 to obtain a semi-algorithm à la Karp and Miller for the termination and boundedness problems for HOPVASS. As seen in Section 3, these problems are undecidable. It is however possible to search for witnesses of non-termination, and, with a slight modification, unboundedness. We first present the semi-algorithm, called reduced reachability tree, and prove its correctness. Then, we present an example of HOPVASS on which it does not terminate. Finally, we prove that the semi-algorithm always terminates on HOPDA, and so is a decision procedure for the termination and boundedness problems for HOPDA. We also recall that it is a decision procedure for 1-PVASS as well [14]. We borrow its presentation from that paper.

We define the *reachability tree* of a d-dim n-PVASS $\mathcal{S}$ as follows. Nodes of the tree are labelled by configuration of $\mathcal{S}$. The root r is labelled by the initial configuration $(q_{\text{init}}, \mathbf{0}, []_n)$, written $r : (q_{\text{init}}, \mathbf{0}, []_n)$. Each node $u : (p, \mathbf{x}, s)$ has one child $v : (q, \mathbf{y}, t)$ for each step $(p, \mathbf{x}, s) \xrightarrow{\mathbf{a},\theta} (q, \mathbf{y}, t)$ in $\mathcal{S}$, and the edge from u to v is labelled by the pair $(\mathbf{a}, \theta)$. Notice that the reachability tree of $\mathcal{S}$ is finitely branching.

We say that a node $u : (p, \mathbf{x}, s)$ *subsumes* a node $v : (q, \mathbf{y}, t)$ if u is a proper ancestor of v, $p = q$, $\mathbf{x} \leq \mathbf{y}$, and the block ρ from u to v is iterable on s. Furthermore, we say that u *strictly subsumes* v if $\mathbf{x} < \mathbf{y}$ or $\text{red}(\rho)$ is not ε.

▶ **Theorem 16.** *If the reachability tree of a d-dim n-PVASS $\mathcal{S}$ contains two nodes u and v such that u* subsumes *v (resp., u* strictly subsumes *v), then $\mathcal{S}$ has an infinite initialised run (resp., an infinite reachability set).*

For VASS [10], WSTS [6, 7] and 1-PVASS [14], it can be shown that every infinite branch of the reachability tree contains two nodes such that one subsumes the other. Therefore, termination and boundedness can be solved by constructing the so-called *reduced reachability tree*, or *RRT* for short, which is constructed like the reachability tree, but on which every branch is stopped at the first node subsumed by one of its ancestors. When the RRT of an HOPVASS is finite, it can be computed and it contains enough information to decide termination and boundedness.

As seen in Section 3, boundedness is undecidable for HOPVASS. Figure 2 depicts an example of a 1-dim 2-PVASS whose reduced reachability tree is infinite. There is only one infinite run in this HOPVASS, and for any two configurations with the same state in this run, either the latter one has a smaller counter value, or the sequence of operations between them is not an iterable n-block. That can be proven by an easy case study on the configurations (see Appendix C).

We now turn to show that the RRT is finite in natural subcases of HOPVASS. In [14], it is shown that the RRT is finite for 1-PVASS, by replacing, in the definition of subsumption, our iterability condition with the condition that s is a prefix of every stack appearing on the path from u to v. Actually, the technique presented here yields, at order 1, a (slightly) smaller RRT than in [14], as contrary to it, we can detect that a block in the tree is iterable even if it destructs the stack and then reconstructs it. The rest of the section is devoted to the proof that the RRT is also finite in the case of HOPDA. We first have to introduce some notations and recall some facts.

▶ **Lemma 17.** *If ρ is a block applicable to $[]_n$, then for every order k, $\overline{\mathrm{red}}_k(\rho) = \mathrm{red}(\rho|_k)$ and $\overline{\mathrm{red}}_k(\rho)$ contains no $\overline{\mathtt{copy}}_k$ (no $\mathtt{pop}_a$ if $k = 1$).*

From [2, 3], for every n-stack s, there exists a *unique reduced block* ρ_s such that $\rho_s([]_n) = s$. We define a norm on n-stacks, such that $||s||$ is the length $|\rho_s|$ of the reduced block ρ_s. For every $k \leq n$, we define $||s||_k = ||\mathrm{top}_k(s)||$. We make the following observations.

▶ **Lemma 18.** *For every n-stack s and orders $k < k' \leq n$, it holds that $||s||_k \leq ||s||_{k'}$.*

▶ **Lemma 19.** *Given m, there are at most $(2(|\Gamma|+n-1)-1)^m$ n-stacks s such that $||s||_n = m$.*

We are now ready to prove the main result of this section, namely that the RRT of an HOPDA is finite. To do so, we investigate all possible forms of infinite branch that can appear in the reachability tree of an HOPDA and show that, in all cases, it is possible to extract an iterable block between two nodes with the same state. The easy case is when the branch visits only finitely many stacks, hence, finitely many configurations. In that case, there are two identical configurations on the branch, and the block between them is obviously iterable.

The other case is more involved. When the RRT has an infinite branch, this branch represents an infinite run $(q_0, s_0) \xrightarrow{\theta_1} (q_1, s_1) \cdots \xrightarrow{\theta_k} (q_k, s_k) \cdots$, with $q_0 = q_{\mathrm{init}}$ and $s_0 = []_n$. We then consider the smallest order k for which the sequence $(||s_i||_k)_{i\in\mathbb{N}}$, is unbounded. For every m, we show that we can extract a particular subsequence of positions $j_1, \cdots, j_m$ such that $||s_{j_i}||_k = i$, and that for all stacks s between s_{j_i+1} and s_{j_m}, $||s||_k > i$. We then show that the reduced sequence $\overline{\mathrm{red}}(\theta_{j_i+1} \cdots \theta_{j_{i'}})$ does not contain any $\overline{\mathtt{copy}}_{k'}$ with $k' \geq k$. As k is the smallest order such that $(||s_i||_k)_{i\in\mathbb{N}}$ is unbounded, there are finitely many $(k-1)$-stacks that can appear at the top of the stacks s_i. Consequently we can find a subsequence of $j_1, \cdots, j_m$ such that the topmost $k-1$ stack is the same for all the stacks s_{j_i} with $1 \leq i \leq m$. When m is chosen to be large enough, there must be i and i' so that $q_{j_i} = q_{j'_i}$. Then the conditions are met for us to use Theorem 15. This gives us an iterable block between two nodes with the same state on the infinite branch we considered.

▶ **Theorem 20.** *The reduced reachability tree of an HOPDA is finite.*

Proof. We consider an n-PDA and suppose its RRT is infinite. By Koenig's Lemma, it contains an infinite branch $(q_0, s_0 = []_n), (q_1, s_1), (q_2, s_2), \cdots$, and for every $i \geq 1$, we call θ_i the operation such that $s_i = \theta_i(s_{i-1})$. We thus get an infinite n-block $\theta_1\theta_2 \cdots$. Observe that for every i, we get $\rho_{s_i} = \mathrm{red}(\theta_1 \cdots \theta_i)$.

Suppose that the sequence of $||s_i||_n$ is bounded, i.e., there exists $m \in \mathbb{N}$ such that for every i, $||s_i||_n \leq m$. From Lemma 19 there are finitely many n-stacks of norm at most m. Therefore, there is a stack s such that there are infinitely many i such that $s = s_i$. As Q is finite, there are two positions $i < j$ such that $(q_i, s_i) = (q_j, s_j)$. Thus, $s_i = (\theta_{i+1} \cdots \theta_j)(s_i)$, and therefore $\theta_{i+1} \cdots \theta_j$ is iterable on s_i. Therefore i subsumes j, which contradicts the fact that the branch considered is infinite in the RRT.

Suppose now that the sequence of $||s_i||_n$ is unbounded, i.e., for every $m \in \mathbb{N}$, there exists i such that $||s_i||_n > m$. As for every $k < k' \leq n$ and $s \in \text{Stacks}_n$, $||s||_k \leq ||s||_{k'}$ (Lemma 18), we can fix k such that for every $k' \geq k$, the sequence of $||s_i||_{k'}$ is unbounded, but for every $k' < k$ the sequence of $||s_i||_{k'}$ is bounded. For every $m \in \mathbb{N}$, we define j_m the *first position* at which $||s_i||_k = m$, i.e., $j_m = \min(i \mid ||s_i||_k = m)$. Given $p < m \in \mathbb{N}$, we define $i(p,m)$ the *last position before* j_m at which $||s_i||_k = p$, i.e., $i(p,m) = \max(i < j_m \mid ||s_i||_k = p)$. As for every stack s, operation θ and order k, $||s||_k - 1 \leq ||\theta(s)||_k \leq ||s||_k + 1$, the j_m and $i(p,m)$ are defined for every $p \leq m$. Furthermore, observe that $i(p,m)$ is strictly increasing with respect to p.

Let us show that for every $k' \geq k$, for every $p < p' < m$, there is no $\overline{\texttt{copy}}_{k'}$ in $\overline{\text{red}}_{k'}(\theta_{i(p,m)+1} \cdots \theta_{i(p',m)})$. Observe first that, as by definition $||s_{i(p,m)}||_k < ||s_{i(p,m)+1}||_k$, $\theta_{i(p,m)} \in \texttt{Op}_k$. Suppose there is a position i with $i(p,m) < i < i(p',m)$ such that $\theta_i = \overline{\texttt{copy}}_{k'}$. As $\theta_1 \cdots \theta_i$ is applicable to $[]_n$, $\overline{\text{red}}_{k'}(\theta_0 \cdots \theta_i)$ does not contain any $\overline{\texttt{copy}}_{k'}$ (Lemma 17), and therefore there is a position $i' < i$ such that $\theta_{i'} = \texttt{copy}_{k'}$, $\theta_{i'+1} \cdots \theta_{i-1}$ does not contain any $\texttt{copy}_k$ nor $\overline{\texttt{copy}}_k$ and $\overline{\text{red}}_{k'}(\theta_{i'+1} \cdots \theta_{i-1}) = \varepsilon$. As $\theta_{i'+1} \cdots \theta_{i-1}$ contains no $\texttt{copy}_k$ nor $\overline{\texttt{copy}}_k$, by definition of reduction, for every $i' < \ell < i$, $||s_\ell||_k \geq ||s_{i'}||_k = ||s_i||_k$. Suppose $i' < i(p,m) + 1$, we thus have $||s_{i(p,m)}|| \geq ||s_i||$, which contradicts the definition of $i(p,m)$. Therefore $i' \geq i(p,m) + 1$, and $\overline{\text{red}}_{k'}(\theta_{i(p,m)+1} \cdots \theta_i)$ does not contain any $\overline{\texttt{copy}}_{k'}$. Thus, in any case, $\overline{\text{red}}_{k'}(\theta_{i(p,m)+1} \cdots \theta_{i(p',m)})$ does not contain any $\overline{\texttt{copy}}_{k'}$.

From Lemma 19, there are at most $(2(|\Gamma| + n - 1) - 1)^{h+1}$ $(k-1)$-stacks of norm at most h. We take $m > |Q| * (2(|\Gamma| + n - 1) - 1)^{h+1}$, where h is the highest value for $||s_i||_{k-1}$. Therefore, we can find $|Q|+1$ positions $p_1 < p_2 < \cdots < p_{|Q|+1}$ such that $\overline{\text{red}}_{k-1}(\theta_1 \cdots \theta_{i(p_i,m)})$ is the same for every p_i, and therefore, for every $i < j$, $\overline{\text{red}}_{k-1}(\theta_{i(p_i,m)+1} \cdots \theta_{i(p_j,m)}) = \varepsilon$. As from what precedes, for every $k' \geq k$ and $i < j$, $\overline{\text{red}}_{k'}(\theta_{i(p_i,m)+1} \cdots \theta_{i(p_j,m)})$ does not contain any $\overline{\texttt{copy}}_{k'}$, from Theorem 15 we get that $\theta_{i(p_i,m)+1} \cdots \theta_{i(p_j,m)}$ is iterable on $s_{i(p_i,m)}$. We can furthermore find $i < j$ such that $q_{i(p_i,m)} = q_{i(p_j,m)}$, and therefore, we get that $(q_{i(p_i,m)}, s_{i(p_i,m)})$ subsumes $(q_{i(p_j,m)}, s_{i(p_j,m)})$, which contradicts the fact that the RRT is infinite. ◀

We derive from Theorem 20 that we can solve termination and boundedness for HOPDA by computing the RRT and checking whether it contains a (strictly) subsumed node.

6 Conclusion

In this paper, we have investigated whether an approach à la Karp and Miller can be used to solve termination and boundedness for HOPVASS.

On the negative side, we have shown that coverability, termination, and boundedness are all undecidable for HOPVASS, even in the restricted subcase of one counter and an order 2 stack. This is in sharp contrast with the same model at order 1, for which all three problems are decidable [14, 16].

On the positive side, we have identified a simple and decidable criterion characterising which sequences of higher-order stack operations can be iterated. Such a criterion is crucial for the implementation of Karp and Miller's approach. While the resulting Karp and Miller procedure is only a semi-algorithm for HOPVASS, we have shown that it always terminates for HOPDA. Moreover, when dealing with 1-PVASS, this algorithm is a variant of the algorithm proposed in [14].

We have considered symmetric higher-order operations (as in [2]), namely $\texttt{copy}_n$ operations and their inverse $\overline{\texttt{copy}}_n$. Our undecidability results still hold for HOPVASS defined with $\texttt{pop}_n$ operations instead of $\overline{\texttt{copy}}_n$. We conjecture that Karp and Miller's approach can still be applied to HOPVASS with $\texttt{pop}_n$ and yields an algorithm for HOPDA with $\texttt{pop}_n$.

References

1. Alfred V. Aho. Nested Stack Automata. *J. ACM*, 16(3):383–406, 1969. doi:10.1145/321526.321529.
2. Arnaud Carayol. Regular Sets of Higher-Order Pushdown Stacks. In Joanna Jedrzejowicz and Andrzej Szepietowski, editors, *Mathematical Foundations of Computer Science 2005, 30th International Symposium, MFCS 2005, Gdansk, Poland, August 29 - September 2, 2005, Proceedings*, volume 3618 of *Lecture Notes in Computer Science*, pages 168–179. Springer, 2005. doi:10.1007/11549345_16.
3. Arnaud Carayol. *Automates infinis, logiques et langages*. PhD thesis, University of Rennes 1, France, 2006. URL: https://tel.archives-ouvertes.fr/tel-00628513.
4. Arnaud Carayol and Stefan Wöhrle. The Caucal Hierarchy of Infinite Graphs in Terms of Logic and Higher-Order Pushdown Automata. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science, 23rd Conference, Mumbai, India, December 15-17, 2003, Proceedings*, volume 2914 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 2003. doi:10.1007/978-3-540-24597-1_10.
5. Didier Caucal. On Infinite Terms Having a Decidable Monadic Theory. In Krzysztof Diks and Wojciech Rytter, editors, *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*, volume 2420 of *Lecture Notes in Computer Science*, pages 165–176. Springer, 2002. doi:10.1007/3-540-45687-2_13.
6. Alain Finkel. A Generalization of the Procedure of Karp and Miller to Well Structured Transition Systems. In Thomas Ottmann, editor, *Automata, Languages and Programming, 14th International Colloquium, ICALP87, Karlsruhe, Germany, July 13-17, 1987, Proceedings*, volume 267 of *Lecture Notes in Computer Science*, pages 499–508. Springer, 1987. doi:10.1007/3-540-18088-5_43.
7. Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001. doi:10.1016/S0304-3975(00)00102-X.
8. Sheila A. Greibach. Full AFLs and Nested Iterated Substitution. *Information and Control*, 16(1):7–35, 1970. doi:10.1016/S0019-9958(70)80039-0.
9. Matthew Hague, Jonathan Kochems, and C.-H. Luke Ong. Unboundedness and downward closures of higher-order pushdown automata. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 151–163. ACM, 2016. doi:10.1145/2837614.2837627.
10. Richard M. Karp and Raymond E. Miller. Parallel Program Schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969. doi:10.1016/S0022-0000(69)80011-5.
11. Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-Order Pushdown Trees Are Easy. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002. doi:10.1007/3-540-45931-6_15.
12. Ranko Lazic. The reachability problem for vector addition systems with a stack is not elementary. *CoRR*, abs/1310.1767, 2013. Presented at RP'12. arXiv:1310.1767.
13. Ranko Lazic and Patrick Totzke. What Makes Petri Nets Harder to Verify: Stack or Data? In Thomas Gibson-Robinson, Philippa J. Hopcroft, and Ranko Lazic, editors, *Concurrency, Security, and Puzzles - Essays Dedicated to Andrew William Roscoe on the Occasion of His 60th Birthday*, volume 10160 of *Lecture Notes in Computer Science*, pages 144–161. Springer, 2017. doi:10.1007/978-3-319-51046-0_8.

14 Jérôme Leroux, M. Praveen, and Grégoire Sutre. Hyper-Ackermannian bounds for pushdown vector addition systems. In Thomas A. Henzinger and Dale Miller, editors, *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 63:1–63:10. ACM, 2014. doi:10.1145/2603088.2603146.

15 Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. On Boundedness Problems for Pushdown Vector Addition Systems. In Mikolaj Bojanczyk, Slawomir Lasota, and Igor Potapov, editors, *Reachability Problems - 9th International Workshop, RP 2015, Warsaw, Poland, September 21-23, 2015, Proceedings*, volume 9328 of *Lecture Notes in Computer Science*, pages 101–113. Springer, 2015. doi:10.1007/978-3-319-24537-9_10.

16 Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. On the Coverability Problem for Pushdown Vector Addition Systems in One Dimension. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 324–336. Springer, 2015. doi:10.1007/978-3-662-47666-6_26.

17 Richard J. Lipton. The reachability problem requires exponential space. Technical Report 63, Yale University, January 1976.

18 A.N. Maslov. Multilevel Stack Automata. *Probl. Inf. Transm.*, 12(1):38–43, 1976.

19 Ernst W. Mayr and Albert R. Meyer. The Complexity of the Finite Containment Problem for Petri Nets. *J. ACM*, 28(3):561–576, 1981. doi:10.1145/322261.322271.

20 Ken McAloon. Petri nets and large finite sets. *Theor. Comput. Sci.*, 32:173–183, 1984. doi:10.1016/0304-3975(84)90029-X.

21 Pawel Parys. A Pumping Lemma for Pushdown Graphs of Any Level. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 54–65. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.54.

22 Charles Rackoff. The Covering and Boundedness Problems for Vector Addition Systems. *Theor. Comput. Sci.*, 6:223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

A Proofs of Section 3

▶ **Lemma 2.** *Let $x, y \in \mathbb{N}$ and $s, t \in \mathrm{Stacks}_2$. Assume that $s = [[u\mathbf{b}v]_1]_2$ where $\mathbf{b} \in \Sigma$ and $u, v \in \Sigma^*$ are such that $\mathbf{b}(c) = \mathtt{T}$ and $v(c) \in \mathbb{Z}^*$. Then the following assertions hold:*

- *$(A, x, s) \xrightarrow{*}_{\mathcal{F}_c} (D, y, t)$ in $\mathcal{F}_c$ if, and only if, $s = t$ and $x + \delta(v(c)) = y$,*
- *$(E, x, s) \xrightarrow{*}_{\mathcal{B}_c} (H, y, t)$ in $\mathcal{B}_c$ if, and only if, $s = t$ and $x - \delta(v(c)) = y$.*

Proof. We start with the proof of the first assertion. Define $w = u\mathbf{b}v$. Suppose that there are runs from (A, x, s) to (D, y, t) in $\mathcal{F}_c$, and pick one of them. Since $\mathbf{b}(c) = \mathtt{T}$ and $\mathbf{a}(c) \neq \mathtt{T}$ for every action $\mathbf{a}$ occurring in v, the run necessarily begins with the following steps:

$$(A, x, s) \xrightarrow{\mathtt{copy}_2} (B, x, [[w]_1[u\mathbf{b}v]_1]_2) \xrightarrow{*} (B, x', [[w]_1[u\mathbf{b}]_1]_2) \xrightarrow{\mathtt{peek}_{\mathbf{b}}} (C, x', [[w]_1[u\mathbf{b}]_1]_2) \quad (2)$$

where $x' = x + |v|K$. Then, the run necessarily continues with the following steps:

$$(C, x', [[w]_1[u\mathbf{b}]_1]_2) \xrightarrow{-K+\mathbf{a}_1(c),\mathtt{push}_{\mathbf{a}_1}} \cdots \xrightarrow{-K+\mathbf{a}_k(c),\mathtt{push}_{\mathbf{a}_k}} (C, z, [[w]_1[u\mathbf{b}\mathbf{a}_1 \cdots \mathbf{a}_k]_1]_2) \quad (3)$$

for some $z \in \mathbb{N}$ and some actions $\mathbf{a}_1, \ldots, \mathbf{a}_k$ in Σ such that $\mathbf{a}_i(c) \neq \mathtt{T}$ for every $1 \leq i \leq k$. It follows from the definition of steps in 1-dim 2-PVASS that $x' \xrightarrow{(-K+\mathbf{a}_1(c))\cdots(-K+\mathbf{a}_k(c))} z$.

This entails that $z = x' - kK + \delta(\mathbf{a}_1(c)\cdots\mathbf{a}_k(c))$. Finally, the run necessarily ends with the following step:

$$(C, z, [[w]_1[u\mathbf{b}\mathbf{a}_1\cdots\mathbf{a}_k]_1]_2) \xrightarrow{\overline{\mathtt{copy}}_2} (D, y, t) \tag{4}$$

It follows that $y = z$ and that $t = \overline{\mathtt{copy}}_2([[w]_1[u\mathbf{b}\mathbf{a}_1\cdots\mathbf{a}_k]_1]_2)$. The last equality entails that $t = [[w]_1]_2 = s$ and $w = u\mathbf{b}\mathbf{a}_1\cdots\mathbf{a}_k$. Since $w = u\mathbf{b}v$, we get that $v = \mathbf{a}_1\cdots\mathbf{a}_k$, hence, $v(c) = \mathbf{a}_1(c)\cdots\mathbf{a}_k(c)$. We conclude that $y = z = x' - |v|K + \delta(v(c)) = x + \delta(v(c))$.

Conversely, suppose that $s = t$ and $x + \delta(v(c)) = y$. Let us write v as $v = \mathbf{a}_1\cdots\mathbf{a}_k$ with $\mathbf{a}_i \in \Sigma$. Note that $\mathbf{a}_i(c) \neq \mathtt{T}$ for every $1 \leq i \leq k$, by assumption. Therefore, Equation 2 is a run in $\mathcal{F}_c$, where $x' = x + kK$. Observe that, for every $1 \leq i \leq k$,

$$\begin{aligned} x' + (-K + \mathbf{a}_1(c)) + \cdots + (-K + \mathbf{a}_i(c)) &= x + (k-i)K + \mathbf{a}_1(c) + \cdots + \mathbf{a}_i(c) \\ &= y + (K - \mathbf{a}_{i+1}(c)) + \cdots + (K - \mathbf{a}_k(c)) \\ &\geq 0 \end{aligned}$$

It follows that $x' \xrightarrow{(-K+\mathbf{a}_1(c))\cdots(-K+\mathbf{a}_k(c))} y$. We deduce that Equation 3 is also a run in $\mathcal{F}_c$, by letting $z = y$. Moreover, Equation 4 is also a run in $\mathcal{F}_c$ since $z = y$ and $w = u\mathbf{b}v = u\mathbf{b}\mathbf{a}_1\cdots\mathbf{a}_k$. By concatenating these three runs, we obtain that $(A, x, s) \xrightarrow{*} (D, y, t)$ in $\mathcal{F}_c$.

The second assertion follows from the first assertion by replacing $v = \mathbf{a}_1\cdots\mathbf{a}_k$ with $v' = \mathbf{a}'_1\cdots\mathbf{a}'_k$, where $\mathbf{a}'_i$ differs from $\mathbf{a}_i$ only in c, with $\mathbf{a}'_i(c) = -\mathbf{a}_i(c)$. ◀

▶ **Lemma 3.** *Let $y \in \mathbb{N}$ and $s, t \in \mathrm{Stacks}_2$. Assume that $s = [[(\mathtt{T}, \ldots, \mathtt{T})w\mathbf{a}]_1]_2$ where $\mathbf{a} \in \Sigma$ and $w \in \Sigma^*$ are such that $\mathbf{0} \xrightarrow{w}$. Then $(I, 0, s) \xrightarrow{*} (J, y, t)$ in $\mathcal{C}_c$ if, and only if, $y = 0$, $s = t$ and $0 \xrightarrow{w(c)\mathbf{a}(c)}$.*

Proof. We may write $(\mathtt{T}, \ldots, \mathtt{T})w = u\mathbf{b}v$ for some $\mathbf{b} \in \Sigma$ and $u, v \in \Sigma^*$ such that $\mathbf{b}(c) = \mathtt{T}$ and $v(c) \in \mathbb{Z}^*$. This entails that $\mathtt{T}w(c) = u(c)\mathtt{T}v(c)$. Since $\mathbf{0} \xrightarrow{w}$, we get that $0 \xrightarrow{u(c)\mathtt{T}v(c)}$, hence, $0 \xrightarrow{u(c)\mathtt{T}} 0 \xrightarrow{v(c)} x$ for $x = \delta(v(c))$. Observe that $x \xrightarrow{\mathbf{a}(c)}$ if, and only if, $0 \xrightarrow{w(c)\mathbf{a}(c)}$. The "only if" direction follows from $0 \xrightarrow{w(c)} x$ and the "if" direction follows from forward determinism of $\xrightarrow{w(c)}$. We now proceed with the proof of the lemma.

Suppose that $(I, 0, s) \xrightarrow{*} (J, y, t)$ in $\mathcal{C}_c$. We consider two cases, depending on $\mathbf{a}(c)$. If $\mathbf{a}(c) \in \mathbb{Z}$ then, by definition of $\mathcal{C}_c$ (see Figure 1b), we have $(A, 0, s) \xrightarrow{*} (D, z, t')$ in $\mathcal{F}_c$ and $(E, z, t') \xrightarrow{*} (H, y, t)$ in $\mathcal{B}_c$, for some $z \in \mathbb{N}$ and $t' \in \mathrm{Stacks}_2$. Recall that $s = [[u\mathbf{b}v\mathbf{a}]_1]_2$. We get from Lemma 2 that $s = t'$ and $\delta(v(c)\mathbf{a}(c)) = z$, and we get from Lemma 2 that $t' = t$ and $z - \delta(v(c)\mathbf{a}(c)) = y$. It follows that $y = 0$ and $x + \mathbf{a}(c)) = \delta(v(c)) + \mathbf{a}(c)) = z \geq 0$. We derive that $x \xrightarrow{\mathbf{a}(c)}$, hence, $0 \xrightarrow{w(c)\mathbf{a}(c)}$.

The other case is when $\mathbf{a}(c) = \mathtt{T}$. In that case, by definition of $\mathcal{C}_c$ (see Figure 1b), we have $(E, 0, s') \xrightarrow{*} (H, y, t')$ in $\mathcal{B}_c$, for some $s', t' \in \mathrm{Stacks}_2$ such that $s' = \mathtt{pop}_{\mathbf{a}}(s)$ and $t = \mathtt{push}_{\mathbf{a}}(t')$. Note that $s' = [[u\mathbf{b}v]_1]_2$. We get from Lemma 2 that $s' = t'$ and $-\delta(v(c)) = y$, hence, $\delta(v(c)) \leq 0$. It follows that $x = \delta(v(c)) = 0$, and therefore $x \xrightarrow{\mathtt{T}}$. Moreover, we deduce from $s' = t'$ that $t = \mathtt{push}_{\mathbf{a}}(\mathtt{pop}_{\mathbf{a}}(s)) = s$. We have shown that $y = 0$, $s = t$ and $x \xrightarrow{\mathbf{a}(c)}$. Hence, $0 \xrightarrow{w(c)\mathbf{a}(c)}$.

Conversely, suppose that $0 \xrightarrow{w(c)\mathbf{a}(c)}$ and let us show that $(I, 0, s) \xrightarrow{*} (J, 0, s)$. Note that $x \xrightarrow{\mathbf{a}(c)}$ and let $z \in \mathbb{N}$ such that $x \xrightarrow{\mathbf{a}(c)} z$. It follows that $z = \delta(v(c)\mathbf{a}(c))$ since $x = \delta(v(c))$.

Recall that $s = [[u\mathbf{b}v\mathbf{a}]_1]_2$. We again consider two cases, depending on $\mathbf{a}(c)$. If $\mathbf{a}(c) \in \mathbb{Z}$ then we get from Lemma 2 that $(A,0,s) \xrightarrow{*} (D,z,s)$ in $\mathcal{F}_c$, and we get from Lemma 2 that $(E,z,s) \xrightarrow{*} (H,0,s)$ in $\mathcal{B}_c$. It follows that $(I,0,s) \xrightarrow{*} (J,0,s)$ in $\mathcal{C}_c$. If $\mathbf{a}(c) = \mathtt{T}$ then $x = 0$ since $x \xrightarrow{\mathbf{a}(c)}$. Hence, $\delta(v(c)) = 0$. Let $s' = [[u\mathbf{b}v]_1]_2$ and note that $s' = \mathtt{pop}_{\mathbf{a}}(s)$. We get that from Lemma 2 that $(E,0,s') \xrightarrow{*} (H,0,s')$ in $\mathcal{B}_c$. It follows that $(I,0,s) \xrightarrow{*} (J,0,s)$ in $\mathcal{C}_c$. ◀

▶ **Lemma 4.** *Let $\mathbf{x} \in \mathbb{N}^d$ and $s \in \mathrm{Stacks}_2$ such that $\mathbf{x} \bowtie s$. For every step $(p,\mathbf{x}) \xrightarrow{\mathbf{a}} (q,\mathbf{y})$ in $\mathcal{M}$, there exists a run $(p,0,s) \xrightarrow{\mathtt{push}_{\mathbf{a}}} (\widetilde{q},0,t) \rightsquigarrow (q,0,t)$ in $\mathcal{S}$ with $\mathbf{y} \bowtie t$.*

Proof. Consider a step $(p,\mathbf{x}) \xrightarrow{\mathbf{a}} (q,\mathbf{y})$ in $\mathcal{M}$. Since $\mathbf{x} \bowtie s$, there exists of $w \in \Sigma^*$ such that $\mathbf{0} \xrightarrow{w} \mathbf{x}$ and $s = [[(\mathtt{T},\dots,\mathtt{T})w]_1]_2$. Let $t = \mathtt{push}_{\mathbf{a}}(s) = [[(\mathtt{T},\dots,\mathtt{T})w\mathbf{a}]_1]_2$. Observe that, by construction of $\mathcal{S}$ from $\mathcal{M}$ (see Figure 1c), $(p,0,s) \xrightarrow{\mathtt{push}_{\mathbf{a}}} (\widetilde{q},0,t)$ is a step in $\mathcal{S}$, since $p \xrightarrow{\mathbf{a}} q$ is a transition in $\mathcal{M}$. Notice that $\mathbf{0} \xrightarrow{w\mathbf{a}} \mathbf{y}$ since $\mathbf{0} \xrightarrow{w} \mathbf{x}$ and $\mathbf{x} \xrightarrow{\mathbf{a}} \mathbf{y}$. This entails that $\mathbf{y} \bowtie t$ and that $0 \xrightarrow{w(c)\mathbf{a}(c)}$ for every $1 \le c \le d$. We derive from Lemma 3 that $(I,0,t) \xrightarrow{*} (J,0,t)$ in $\mathcal{C}_c$ for every $1 \le c \le d$. It follows that $(\widetilde{q},0,t) \rightsquigarrow (q,0,t)$ in $\mathcal{S}$. ◀

▶ **Corollary 5.** *For every initialised run $(q_0,\mathbf{x}_0) \xrightarrow{\mathbf{a}_1} (q_1,\mathbf{x}_1) \cdots \xrightarrow{\mathbf{a}_k} (q_k,\mathbf{x}_k) \cdots$ in $\mathcal{M}$, there is an initialised run $(\widetilde{q_{\mathrm{init}}},0,[]_2) \xrightarrow{\mathtt{push}_{(\mathtt{T},\dots,\mathtt{T})}} (q_0,0,s_0) \xrightarrow{\mathtt{push}_{\mathbf{a}_1}} (\widetilde{q_1},0,s_1) \rightsquigarrow (q_1,0,s_1) \cdots \xrightarrow{\mathtt{push}_{\mathbf{a}_k}} (\widetilde{q_k},0,s_k) \rightsquigarrow (q_k,0,s_k) \cdots$ in $\mathcal{S}$.*

Proof. Recall that the initial configuration of $\mathcal{M}$ is $(q_0,\mathbf{x}_0) = (q_{\mathrm{init}},\mathbf{0})$ and that the initial configuration of $\mathcal{S}$ is $(\widetilde{q_{\mathrm{init}}},0,[]_2)$. Observe that $(\widetilde{q_{\mathrm{init}}},0,[]_2) \xrightarrow{\mathtt{push}_{(\mathtt{T},\dots,\mathtt{T})}} (q_{\mathrm{init}},0,s_0)$ in $\mathcal{S}$ for the stack $s_0 = [[(\mathtt{T},\dots,\mathtt{T})]_1]_2$. Note that $\mathbf{0} \bowtie s_0$. It follows from Lemma 4, by induction on i, that there exists $s_i \in \mathrm{Stacks}_2$ and runs $(q_{i-1},0,s_{i-1}) \xrightarrow{\mathtt{push}_{\mathbf{a}_i}} (\widetilde{q_i},0,s_i) \rightsquigarrow (q_i,0,s_i)$ in $\mathcal{S}$ with $\mathbf{x}_i \bowtie s_i$, for every $i \ge 1$. We obtain the desired initialised run of $\mathcal{S}$ by concatenating these runs. ◀

▶ **Lemma 6.** *Assume that $t = [[(\mathtt{T},\dots,\mathtt{T})w\mathbf{a}]_1]_2$ where $\mathbf{a} \in \Sigma$ and $w \in \Sigma^*$ are such that $\mathbf{0} \xrightarrow{w}$. For every run $(\widetilde{q},0,t) \rightsquigarrow (q,x,s)$, it holds that $x = 0$, $s = t$ and $\mathbf{0} \xrightarrow{w\mathbf{a}}$.*

Proof. Consider a run $(\widetilde{q},0,t) \rightsquigarrow (q,x,s)$. Recall that $\rightsquigarrow$ means that the run can be decomposed into a first step (moving from $\widetilde{q}$ to $\mathcal{C}_1$), a last step (moving from $\mathcal{C}_d$ to q), and runs of $\mathcal{C}_1,\dots,\mathcal{C}_d$ in between. So there exists $x_0,\dots,x_d \in \mathbb{N}$ and $s_0,\dots,s_d \in \mathrm{Stacks}_2$, with $x_0 = 0$, $s_0 = t$, $x_d = x$ and $s_d = s$, such that $(I,x_{c-1},s_{c-1}) \xrightarrow{*} (J,x_c,s_c)$ in $\mathcal{C}_c$, for every $1 \le c \le d$. We derive from Lemma 3, by induction on c, that $x_c = 0$, $s_c = t$ and $0 \xrightarrow{w(c)\mathbf{a}(c)}$, for every $1 \le c \le d$. It follows that $x = x_d = 0$, $t = s_d = s$ and $\mathbf{0} \xrightarrow{w\mathbf{a}}$. ◀

▶ **Corollary 7.** *Every initialised run of $\mathcal{S}$ that is infinite or ends with a configuration whose state is in Q, is of the form $(\widetilde{q_{\mathrm{init}}},0,[]_2) \xrightarrow{\mathtt{push}_{(\mathtt{T},\dots,\mathtt{T})}} (q_0,0,s_0) \xrightarrow{\mathtt{push}_{\mathbf{a}_1}} (\widetilde{q_1},0,s_1) \rightsquigarrow (q_1,0,s_1) \cdots \xrightarrow{\mathtt{push}_{\mathbf{a}_k}} (\widetilde{q_k},0,s_k) \rightsquigarrow (q_k,0,s_k) \cdots$ with $q_i \in Q$. Moreover, for every such run in $\mathcal{S}$, there is an initialised run $(q_0,\mathbf{x}_0) \xrightarrow{\mathbf{a}_1} (q_1,\mathbf{x}_1) \cdots \xrightarrow{\mathbf{a}_k} (q_k,\mathbf{x}_k) \cdots$ in $\mathcal{M}$.*

Proof. Consider an initialised run in $\mathcal{S}$ that is infinite or ends with a configuration whose state is in Q. If the run is infinite, then it visits infinitely many configurations whose state are in Q. Because if it were not the case, then an infinite suffix of the run would remain forever in the same $\mathcal{F}_c$ or $\mathcal{B}_c$. This is impossible as each loop in $\mathcal{F}_c$ or $\mathcal{B}_c$ either shrinks the stack or decreases the counter κ, since K satisfies $|\mathbf{a}(c)| < K$ for every $\mathbf{a} \in \Sigma$ with $\mathbf{a}(c) \neq \mathtt{T}$. So the initialised run under consideration starts with the step $(\widetilde{q_{\mathrm{init}}},0,[]_2) \xrightarrow{\mathtt{push}_{(\mathtt{T},\dots,\mathtt{T})}} (q_{\mathrm{init}},0,[[(\mathtt{T},\dots,\mathtt{T})]_1]_2)$ followed by a run of the form:

$$(q_0,x_0,s_0) \xrightarrow{\mathtt{push}_{\mathbf{a}_1}} (\widetilde{q_1},y_1,t_1) \rightsquigarrow (q_1,x_1,s_1) \cdots \xrightarrow{\mathtt{push}_{\mathbf{a}_k}} (\widetilde{q_k},y_k,t_k) \rightsquigarrow (q_k,x_k,s_k) \cdots$$

where $(q_0, x_0, s_0) = (q_{\text{init}}, 0, [[(\mathtt{T}, \ldots, \mathtt{T})]_1]_2)$ and $q_1, \ldots, q_k, \ldots$ are in Q. Observe that $y_i = x_{i-1}$ and $t_i = \mathtt{push}_{\mathbf{a}_i}(s_{i-1})$, for every $i \geq 1$. We derive from Lemma 6, by induction on i, that $x_i = 0$, $s_i = t_i = [(\mathtt{T}, \ldots, \mathtt{T})\mathbf{a}_1 \cdots \mathbf{a}_i]_2$ and $\mathbf{0} \xrightarrow{\mathbf{a}_1 \cdots \mathbf{a}_i}$, for every $i \geq 1$. Let $\mathbf{x}_i \in \mathbb{N}^d$ such that $\mathbf{0} \xrightarrow{\mathbf{a}_1 \cdots \mathbf{a}_i} \mathbf{x}_i$. It is readily seen that $\mathbf{0} \xrightarrow{\mathbf{a}_1} \mathbf{x}_1 \cdots \xrightarrow{\mathbf{a}_k} \mathbf{x}_k \cdots$. This comes from the observation that $\xrightarrow{w}$ is forward deterministic. Moreover, the construction of $\mathcal{S}$ from $\mathcal{M}$ (see Figure 1c) entails that $q_{i-1} \xrightarrow{\mathbf{a}_i} q_i$ is a transition of $\mathcal{M}$, for every $i \geq 1$. It follows that $(q_0, \mathbf{0}) \xrightarrow{\mathbf{a}_1} (q_1, \mathbf{x}_1) \cdots \xrightarrow{\mathbf{a}_k} (q_k, \mathbf{x}_k) \cdots$ is a run in $\mathcal{M}$. ◀

B Proofs of Section 4

▶ **Lemma 13.** *For every n-block ρ and order k, if $\overline{\text{red}}_k(\rho)$ contains a factor of the form $\mathtt{copy}_k\rho'\overline{\mathtt{copy}}_k$ (or $\mathtt{push}_a\rho'\mathtt{pop}_b$ if $k = 1$), then $\text{dom}(\rho) = \emptyset$.*

Proof. By contradiction, suppose $s \in \text{dom}(\rho)$ and $\overline{\text{red}}_k(\rho)$ contains a factor of the form $\mathtt{copy}_k\rho'\overline{\mathtt{copy}}_k$ (or $\mathtt{push}_a\rho'\mathtt{pop}_b$ if $k = 1$). Let ρ_2 be one of the smallest such ρ'.

If $k = 1$ then we get that $\rho_2 = \varepsilon$, hence, $\overline{\text{red}}_k(\rho)$ contains a factor of the form $\mathtt{push}_a\mathtt{pop}_b$. If $a = b$, this contradicts the fact that $\overline{\text{red}}_k(\rho)$ is weakly reduced. If $a \neq b$, this contradicts the assumption that $\text{dom}(\rho) \neq \emptyset$.

If $k > 1$ then we get that $\rho_2 \in \mathtt{Op}^*_{k-1}$. We may write $\overline{\text{red}}_k(\rho) = \rho_1\mathtt{copy}_k\rho_2\overline{\mathtt{copy}}_k\rho_3$ for some ρ_1 and ρ_3. By Theorem 11, $s \in \text{dom}(\overline{\text{red}}_k(\rho))$. Therefore $(\rho_1\mathtt{copy}_k\rho_2)(s) \in \text{dom}(\overline{\mathtt{copy}}_k)$. As $\rho_2 \in \mathtt{Op}^*_{k-1}$, we necessarily have $\rho_2(s) = s$. By Lemma 12, $\text{red}(\rho_2) = \varepsilon$, and as it is in $\mathtt{Op}^*_{k-1}$, we derive that $\overline{\text{red}}_k(\rho_2) = \varepsilon$. Therefore, $\overline{\text{red}}_k(\rho) = \rho_1\mathtt{copy}_k\overline{\mathtt{copy}}_k\rho_3$, which contradicts the fact that $\overline{\text{red}}_k(\rho)$ is weakly reduced. ◀

C Proofs and comments of Section 5

▶ **Lemma 21.** *For every n-block ρ and every $i \geq 1$, $\text{red}(\rho) = \varepsilon$ if, and only if, $\text{red}(\rho^i) = \varepsilon$.*

Proof. We only prove the "if" direction as the "only if" direction is trivial. By contradiction, suppose that $i \geq 2$, $\text{red}(\rho^i) = \varepsilon$ and $\text{red}(\rho) \neq \varepsilon$. Let us decompose $\text{red}(\rho)$ as $\text{red}(\rho) = \overline{\rho_1}\rho_2\rho_1$ where ρ_1 is maximal in length. Note that $\rho_2 \neq \varepsilon$ since we would get $\text{red}(\rho) = \text{red}(\overline{\rho_1}\rho_1) = \varepsilon$ otherwise. By definition of red, it holds that $\text{red}(\rho^i) = \text{red}(\text{red}(\rho)^i) = \text{red}(\overline{\rho_1}\rho_2^i\rho_1) = \varepsilon$. So there exists a $\theta\bar{\theta}$ factor in $\overline{\rho_1}\rho_2^i\rho_1$. Recall that $\overline{\rho_1}\rho_2\rho_1$ is reduced. This means this $\theta\bar{\theta}$ factor is necessarily at the junction between to consecutive ρ_2. Formally, we get that $\rho_2 = \bar{\theta}\rho_3\theta$ for some ρ_3. Hence, $\text{red}(\rho) = \overline{\rho_1}\bar{\theta}\rho_3\theta\rho_1$, which contradicts the maximality of ρ_1. ◀

▶ **Corollary 22.** *For every n-block ρ and stack s such that ρ is iterable on s, if $\rho(s) \neq s$ then the infinite sequence $s, \rho(s), \rho^2(s), \ldots, \rho^i(s), \ldots$ contains no repetition.*

Proof. If the sequence $s, \rho(s), \rho^2(s), \ldots, \rho^i(s), \ldots$ contains a repetition, then there is a stack t satisfying $\rho^j(t) = t$ for some $j \geq 1$. This entails, by Lemmas 12 and 21, that $\text{red}(\rho) = \varepsilon$, hence, $\rho(s) = s$. ◀

▶ **Theorem 16.** *If the reachability tree of a d-dim n-PVASS $\mathcal{S}$ contains two nodes u and v such that u* subsumes *v (resp., u* strictly subsumes *v), then $\mathcal{S}$ has an infinite initialised run (resp., an infinite reachability set).*

Proof. We have $v : (q, \mathbf{x}, s)$ and $v' : (q, \mathbf{x} + \mathbf{y}, \rho(s))$, where $\mathbf{y}$ is a componentwise nonnegative vector, and ρ is the n-block on the run from v to v'. As $\mathbf{x} \leq \mathbf{x} + \mathbf{y}$, we know that vectorwise, the run from v to v' is applicable to v' (by monotony). As ρ is iterable on s, we know that stackwise, the run from v to v' is applicable to v'. Thus we can apply the run from v to v' on v', and obtain a new node $z : (q, \mathbf{x} + 2\mathbf{y}, \rho^2(s))$, and iterate the process to obtain an infinite sequence of nodes $v_i : (q, \mathbf{x} + i\mathbf{y}, \rho^i(s))$. We therefore have an infinite run in $\mathcal{S}$.

If furthermore v strictly subsumes v', then $\mathbf{y} \neq 0$ or $\rho(s) \neq s$, and we get that all these nodes are labelled by distinct configurations (this claim is obvious if $\mathbf{y} \neq 0$ and comes from Corollary 22 if $\rho(s) \neq s$). Thus $\mathcal{S}$ can reach infinitely many configurations and is thus unbounded. ◀

Non-completeness of the test. We detail here why the only run of the HOPVASS of Figure 2 does not contain any iterable subrun.

- One can show that the only possible configurations containing state B are of the form $(B, 1, [[ba^n]_1]_2)$. Moreover, the only run moving from $(B, 1, [[ba^n]_1]_2)$ to $(B, 1, [[ba^{n+1}]_1]_2)$ passes through the states C, D, E and performs the stack operations sequence $\rho = \mathtt{push}_a \mathtt{copy}_2 \mathtt{pop}_a^{n+1} \mathtt{pop}_b \mathtt{push}_b \mathtt{pop}_a^{n+1} \overline{\mathtt{copy}}_2$. $\overline{\mathrm{red}}_1(\rho) = \mathtt{pop}_a^n \mathtt{pop}_b \mathtt{push}_b \mathtt{push}_a^{n+1}$, and one can see it cannot be written as $\overline{\rho_{E_1}} \rho_{I_1} \rho_{E_1}$ with ρ_{I_1} containing no $\mathtt{pop}$ operation. Intuitively, this run adds an a at the top of the stack, copies the stack, and pops it until the b on the bottom before reconstructing it. As it needs to go to the bottommost symbol while adding a new symbol on top, it cannot be applied a second time, as it doesn't go deep enough anymore.
- Similarly, all possible configurations containing state C are of the form $(C, 0, [[ba^n]_1]_2)$, and the same reasoning applies.
- Configurations containing D are of the form $(D, k, [[ba^n]_1[ba^{n-k}]_1]_2)$. The run going from $(D, k, [[ba^n]_1[ba^{n-k}]_1]_2)$ to $(D, k', [[ba^{n+1}]_1[ba^{n+1-k'}]_1]_2)$ passes through E, B, C, and performs the stack operations sequence $\rho = \mathtt{pop}_a^k \mathtt{pop}_b \mathtt{push}_b \mathtt{push}_a^n \overline{\mathtt{copy}}_2 \mathtt{push}_a \mathtt{copy}_2 \mathtt{pop}_a^{k'}$. It is easy to see that either $\overline{\mathrm{red}}_1(\rho)$ is not iterable, or $\overline{\mathrm{red}}_2(\rho)$ is not iterable (depending on k and k'). The run going from $(D, k, [[ba^n]_1[ba^{n-k}]_1]_2)$ to $(D, k', [[ba^n]_1[ba^{n-k'}]_1]_2)$ stays on D and performs the stack operations sequence $\mathtt{pop}_a^{k-k'}$, which is not iterable.
- Configuration containing E are of the form $(E, k, [[ba^n]_1[ba^{n-k}]_1]_2)$. The run going from $(E, k, [[ba^n]_1[ba^{n-k}]_1]_2)$ to $(E, k', [[ba^{n+1}]_1[ba^{n+1-k'}]_1]_2)$ is similar to the previous case. The run going from $(E, k, [[ba^n]_1[ba^{n-k}]_1]_2)$ to $(E, k', [[ba^n]_1[ba^{n-k'}]_1]_2)$ decreases the counter value, and thus cannot be iterated.

▶ **Lemma 17.** *If ρ is a block applicable to $[]_n$, then for every order k, $\overline{\mathrm{red}}_k(\rho) = \mathrm{red}(\rho|_k)$ and $\overline{\mathrm{red}}_k(\rho)$ contains no $\overline{\mathtt{copy}}_k$ (no $\mathtt{pop}_a$ if $k = 1$).*

Proof. Suppose there is a $\overline{\mathtt{copy}}_k$ in $\rho|_k$ at position j in $\rho = \theta_1 \cdots \theta_m$. As ρ is applicable to $[]_n$, there is a $i < j$ such that $\theta_i = \mathtt{copy}_k$, there is no other $\mathtt{copy}_k$ and $\overline{\mathtt{copy}}_k$ between i and j (w.l.o.g) and $\theta_{i+1} \cdots \theta_{j-1}(\theta_1 \cdots \theta_i([]_n)) = \theta_1 \cdots \theta_i([]_n)$. Thus, by Lemma 12, $\mathrm{red}(\theta_{i+1} \cdots \theta_{j-1}|_k) = \varepsilon$. Therefore, $\overline{\mathrm{red}}_k(\rho)$ does not contain any $\overline{\mathtt{copy}}_k$.

Furthermore, for orders lower than k, both red and $\overline{\mathrm{red}}_k$ coincide syntactically, therefore $\overline{\mathrm{red}}_k(\rho)$ is reduced for red, and by unicity of the reduced k-block, we get the result. ◀

▶ **Lemma 18.** *For every n-stack s and orders $k < k' \leq n$, it holds that $||s||_k \leq ||s||_{k'}$.*

Proof. Given a stack s, and two orders $k < k'$, by definition of application, we have $\rho_{\mathrm{top}_{k'}(s)}|_k([]_k) = \mathrm{top}_k(s) = \rho_{\mathrm{top}_k(s)}([]_k)$. By minimality of the reduced k-block and the definition of restriction, we have $||s||_k = |\rho_{\mathrm{top}_k(s)}| \leq |\rho_{\mathrm{top}_{k'}(s)}|_k| \leq |\rho_{\mathrm{top}_{k'}(s)}| = ||s||_{k'}$. ◀

▶ **Lemma 19.** *Given m, there are at most $(2(|\Gamma|+n-1)-1)^m$ n-stacks s such that $||s||_n = m$.*

Proof. An n-stack s has norm m if and only if its reduced n-block has length m. Such an n-block is a word in $\mathtt{Op}_{n-1} \cup \{\mathtt{copy}_n\}^m$, hence there are at most $(2(|\Gamma| + n - 1) - 1)^m$ such words. ◀