Complexity Analysis of the Backward Coverability Algorithm for VASS

Laura Bozzelli[1] and Pierre Ganty[2,⋆]

[1] UPM Facultad de Informática, Madrid, Spain
[2] IMDEA Software Institute, Madrid, Spain

Abstract. By using the known lower and upper complexity bounds of the coverability problem for VASS, we characterize the complexity of the classical backward algorithm for VASS coverability, and provide optimal bounds on the size of the symbolic representation it computes.

1 Introduction

In [3, 4, 15, 10] checking safety properties for concurrent systems like multithreaded programs, communication protocols, or asynchronous programs is reduced to the coverability problem of VASS (Vector Addition System with States), turning it into a central problem in verification of concurrent systems. Given a VASS G and two configurations s_0 and s_f, the *coverability problem* asks whether s_f is coverable from s_0, i.e. there is a computation in G starting at s_0 and leading to a configuration s which *covers* s_f; that is, s and s_f are in the same control state and the counters of s are pointwise greater or equal than those of s_f (this is noted $s_f \trianglelefteq s$). The complexity of the coverability problem, which is complete for EXPSPACE, was settled in the late 70's (Lipton [13] for the lower bound and Rackoff [14] for the matching upper bound). However, rather surprisingly, the complexity analysis of the algorithms that have been implemented to solve the coverability problem have received little or no attention.[1]

In this work, we propose to characterize the complexity of the so-called *backward algorithm* which has been implemented in several tools and whose definition can be attributed to [1, 9] and to some extent [2]. Given a VASS G and a target configuration s_f, the backward algorithm iteratively computes the configurations from which s_f is coverable in 0 steps, 1 step, ... until the set of configurations is saturated. More precisely, the algorithm symbolically computes an increasing (w.r.t set inclusion) sequence of sets of configurations starting from the set of configurations which cover s_f. Let us call each element of the computed sequence an *iterate* which is given by a set of configurations closed by above for $\trianglelefteq$. Since such upward closed sets are infinite, each iterate is finitely represented and manipulated by its *basis*, that is the *finite* set of its *minimal elements* (w.r.t $\trianglelefteq$). First, let us recall that the minimal elements yields a decidable, finite, and

⋆ This author was sponsored by Comunidad de Madrid's Program PROMETIDOS-CM (S2009TIC-1465), PEOPLE-COFUND's program AMAROUT (PCOFUND-2008-229599), and by the Spanish Ministry of Science and Innovation (TIN2010-20639).

[1] As far as we know, no implementation of Rackoff's algorithm exists.

G. Delzanno and I. Potapov (Eds.): RP 2011, LNCS 6945, pp. 96–109, 2011.

canonical representation of each iterate, and second, because $\trianglelefteq$ is a well-quasi ordering on the set of configurations, it follows that the algorithm is guaranteed to reach a fixpoint $B(G,s_f)$ after finitely many steps. Since $B(G,s_f)$ is the basis of the set of configurations from which s_f is coverable in G, we obtain a decision procedure for the coverability problem: (G,s_0,s_f) is a positive instance of the coverability problem iff $s_{min} \trianglelefteq s_0$ for some $s_{min} \in B(G,s_f)$. Note that $B(G,s_f)$ can be used to solve other coverability related problems such as checking whether from each G-configuration, s_f is coverable.

Our contribution. In this paper, we show that the "backward algorithm" is optimal to solve the coverability problem. Using Rackoff's and Lipton's results [14, 13], respectively, we give upper and lower bounds on the number of iterations of the backward algorithm as well as its execution time. Moreover, our complexity analysis allows us to derive upper bounds on the cardinality of $B(G,s_f)$ and the maximal size of the single elements of $B(G,s_f)$, which are doubly exponential in the dimension of G (the number of counters). Furthermore, we provide matching lower bounds by a readaptation of the Lipton's proof [13].

Besides the backward algorithm, VASS analysis tools often implement a *forward algorithm* whose definition is due to Karp and Miller [12]. The forward algorithm returns a finite representation (the *covering set*) of an overapproximation (the *coverability set*) of the set of configurations reachable from the given initial configuration s_0. Such an overapproximation is sound and also complete for certain problems like the coverability problem. By using the covering set, one can solve, for instance, the coverability problem (by asking whether the target configuration s_f belongs to the coverability set) but also the boundedness problem which asks whether the set of reachable configurations from the given initial configuration is finite. From a complexity standpoint, it is mentioned in [7] that the algorithm of Karp and Miller requires non-primitive recursive space. Let us also cite [8] which gives a more refined complexity analysis of the forward algorithm.

Related work. The closest works to our are [17] which provide an upper bound on the size of $B(G,s_f)$. However, the algorithm to compute $B(G,s_f)$ (originally given in [16]) differs from the backward algorithm and does not yield any conclusion about the complexity of the backward algorithm. Moreover, contrary to us the authors do not provide lower bounds on the size of $B(G,s_f)$.[2]

2 Preliminaries

2.1 Notations and Definitions

Let $\mathbb{Z}$ be the set of integers, $\mathbb{N}$ be the set of nonnegative integers, and $\mathbb{N}^+$ be the set of positive integers. For each $k \in \mathbb{N}^+$ and *vector* $v \in \mathbb{Z}^k$, $v[i]$ denotes the ith component of v, for $i \in \{1,\ldots,k\}$. If $v_1, v_2 \in \mathbb{Z}^k$, then $v_1 + v_2$ denotes that vector $v \in \mathbb{Z}^k$ such that $v[i] = v_1[i] + v_2[i]$ for all $i \in \{1,\ldots,k\}$; $v_1 - v_2$ is defined similarly. Let $v \in \mathbb{Z}^k$, define $\|v\| = \max(\{abs(v[i]) \mid i \in \{1,\ldots,k\}\})$, where $abs(v[i])$ is the absolute value of $v[i]$. Finally, for a finite set Q, $|Q|$ denotes the cardinality of Q.

[2] Similarly to Rackoff's algorithm we do not know of any implementation of the algorithm of [16].

2.2 Well-Quasi Orderings

Recall that for a set S, a *partial order* $\preceq$ over S is a reflexive, transitive and antisymmetric binary relation on S. We say that $\preceq$ is a *well-quasi ordering* (wqo, for short) if additionally, for each infinite sequence $s_0, s_1, \ldots$ of elements of S there are indices $i < j$ such that $s_i \preceq s_j$. Given a partial order $\preceq$ over S, a subset U of S is *upward-closed* (w.r.t. $\preceq$) if for all $s, s' \in S$, $s \in U$ and $s \preceq s'$ entail $s' \in U$. A *basis* of U (w.r.t. $\preceq$) is a subset B of U satisfying the following: (1) for each $s \in U$, there is $s' \in B$ such that $s' \preceq s$, and (2) for all $s, s' \in B$, $s \preceq s'$ implies $s = s'$ (i.e., distinct elements of B are incomparable w.r.t. $\preceq$). The following is a well-known result.

Lemma 1. *[11] Let S be a set and $\preceq$ be a partial order over S which is wqo. Then, each upward-closed subset U of S (w.r.t. $\preceq$) admits a unique basis, which is finite and consists of the minimal elements of U (w.r.t. $\preceq$). Moreover, for each monotone infinite sequence of upward-closed sets $U_0 \subseteq U_1 \subseteq \ldots$, there is $i \geq 0$ such that $U_{i+1} = U_i$.*

Let $k \in \mathbb{N}^+$. We consider the partial order over $\mathbb{N}^k$, written $\trianglelefteq$, which is the componentwise extension of $\leq$ over $\mathbb{N}$: let $v, v' \in \mathbb{N}^k$, $v \trianglelefteq v'$ iff $v[i] \leq v'[i]$ for each $1 \leq i \leq k$. Moreover, for a *finite* set Q, we consider the partial order over $Q \times \mathbb{N}^k$, which (with a little abuse of notation) is again denoted by $\trianglelefteq$, defined as: $\langle q, v\rangle \trianglelefteq \langle q', v'\rangle$ iff $q = q'$ and $v \trianglelefteq v'$. It is well-known that $\trianglelefteq$ is a wqo over $\mathbb{N}^k$ (this result is known as Dickson's Lemma [5]). Hence, it easily follows that $\trianglelefteq$ is a wqo over $Q \times \mathbb{N}^k$, for each *finite* set Q. For $s \in Q \times \mathbb{N}^k$, we denote by $s\!\uparrow$ the upward-closed set given by $\left\{s' \in Q \times \mathbb{N}^k \mid s \trianglelefteq s'\right\}$. In the rest of this paper, if we say that some set $U \subseteq Q \times \mathbb{N}^k$ is upward-closed, we mean that U is upward-closed set w.r.t. $\trianglelefteq$. For $X \subseteq Q \times \mathbb{N}^k$, $min(X)$ denotes the set of minimal elements in X (w.r.t. $\trianglelefteq$). Note that according to Lemma 1, $min(X)$ is the unique (finite) basis of X if X is upward-closed.

2.3 Vector Addition Systems with States (VASS)

Let $d \in \mathbb{N}^+$. A d-VASS G is a pair $\langle Q, \Delta\rangle$, where Q is a non-empty *finite* set of *control points* and $\Delta \subseteq Q \times \mathbb{Z}^d \times Q$ is a *finite* set of transitions in $Q \times \mathbb{Z}^d \times Q$. The d-VASS G induces an infinite directed graph $[\![G]\!] = \left\langle Q \times \mathbb{N}^d, \rightarrow\right\rangle$ whose set of vertices is given by $Q \times \mathbb{N}^d$ and the set of edges is defined as: $\langle q, v\rangle \rightarrow \langle q', v'\rangle$ iff there is $\langle q, u, q'\rangle \in \Delta$ such that $v' = v + u$. Vertices of $[\![G]\!]$ are called *G-states* or simply *states* when G is clear from the context. A *run* $\pi = s_1, \ldots, s_n$ of G is a finite path in the graph $[\![G]\!]$. The length $|\pi|$ of π is n. We define $\|\Delta\| = \max(\{\|v\| \mid \langle q, v, q'\rangle \in \Delta\})$. Moreover, for a state $s = \langle q, v\rangle$ and a *finite* set S of states, define $\|s\| = \|v\|$ and $\|S\| = \max(\{\|s\| \mid s \in S\})$.

For each set S of G-states, $Pre^*(G, S)$ denotes the set of G-states s such that there is a run of G from s to some state in S. Moreover, $Pre(G, S)$ denotes the set of G-states s such that $s \rightarrow s'$ is an edge of $[\![G]\!]$ for some $s' \in S$. It is well-known (see e.g. [1, 9]) that if S is upward-closed, then $Pre^*(G, S)$ and $Pre(G, S)$ are upward-closed as well (this can be easily checked).

2.4 Coverability Problem and Rackoff's Upper Bound

Given a d-VASS $G = \langle Q, \Delta\rangle$ and two G-states s_0 and s_f, a *covering in G of s_f w.r.t. s_0* is a run of G from s_0 which leads to a state s satisfying $s_f \trianglelefteq s$. If such a covering exists, i.e.,

$s_0 \in Pre^*(G, s_f\uparrow)$, we say that s_f *is coverable from* s_0 *in* G. The *coverability problem* asks whether s_f is coverable from s_0 in G for a given d-VASS G and G-states s_0 and s_f. By a straightforward adaptation of the Rackoff's algorithm for the coverability problem [14], we obtain the following result.

Theorem 1. *Let* $G = \langle Q, \Delta \rangle$ *be a* d*-VASS and* s_f *be a state. For each state* s*, if* s_f *is coverable from* s *in* G*, then there is a covering in* G *of* s_f *w.r.t.* s *whose length is independent on* $\|s\|$ *and is at most* $[|Q| \cdot (\|\Delta\| + \|s_f\| + 2)]^{(3d)!+1}$.

Proof of Theorem 1. We need additional definitions. Let $d \in \mathbb{N}^+$ and $I \subseteq \{1, \ldots, d\}$. For $u \in \mathbb{Z}^d$, u^I denotes the vector in $\mathbb{Z}^d$ defined as $u^I[i] = u[i]$ if $i \in I$, and $u^I[i] = 0$ otherwise. For a d-VASS $G = \langle Q, \Delta \rangle$, G^I denotes the d-VASS $G^I = \left\langle Q, \{\langle q, u^I, q' \rangle \mid \langle q, u, q' \rangle \in \Delta\} \right\rangle$. Note that $G^{\{1,\ldots,d\}} = G$. Let $s = \langle q, v \rangle$ be a G-state, we denote by s^I the G-state given by $\langle q, v^I \rangle$, and for a run π, we denote by π^I the sequence of G-states obtained from π by replacing each state s along π with s^I. Note that π^I is a run in G^I. For $B \in \mathbb{N}$, a vector $v \in \mathbb{N}^d$ is B-bounded if $v[i] \leq B$ for each $i \in \{1, \ldots, d\}$. A run π of G is B-bounded if for each state $\langle q, v \rangle$ occurring along π, v is B-bounded.

Fix a d-VASS $G = \langle Q, \Delta \rangle$ and a state $s_f = \langle q_f, v_f \rangle$. For each $I \subseteq \{1, \ldots, d\}$ and G-state s, define $dist(I, s)$ to be the length of the shortest covering in G^I of $(s_f)^I$ w.r.t. s^I, if $(s_f)^I$ is coverable from s^I in G^I (note that $dist(I, s) \geq 1$), and $dist(I, s) = 0$ otherwise. Moreover, for each $k \in \{0, 1, \ldots, d\}$, define $f(k) = sup\{dist(I, s) \mid |I| = k$ and s is a G-state$\}$ (note that $f(k) \geq 1$ since s_f is coverable from itself in G). Then:

Lemma 2. *For all* $k \in \{0, 1, \ldots, d\}$*, the following inequalities hold:*

$$f(k) \leq \begin{cases} |Q| & \text{if } k = 0 \\ |Q| \cdot ((\|\Delta\| + \|s_f\|) \cdot f(k-1))^k + f(k-1) & \text{if } k > 0 \end{cases}$$

Proof. The case $k = 0$ is trivial. Now, assume that $k > 0$. By ind. hyp., $f(k-1)$ is finite. Let s be a G-state and $I \subseteq \{1, \ldots, d\}$ s.t. $|I| = k$ and there is a covering π in G^I of $(s_f)^I$ w.r.t. s^I. We need to show that there is a covering in G^I of $(s_f)^I$ w.r.t. s^I of length bounded by $|Q| \cdot ((\|\Delta\| + \|s_f\|) \cdot f(k-1))^k + f(k-1)$. Let $B = \|\Delta\| \cdot f(k-1) + \|s_f\|$. We distinguish two cases:

Case 1: π is B-bounded. Let s' be the last state of π. Then, there is a B-bounded run π' in G^I from s^I to s' such that the states visited by π' are mutually distinct. It is routine to check that the length of π' is at most $|Q| \cdot B^k$. By hypothesis $(s_f)^I \trianglelefteq s'$, hence π' is also a covering in G^I of $(s_f)^I$ w.r.t. s^I. Thus, since $|Q| \cdot B^k \leq |Q| \cdot ((\|\Delta\| + \|s_f\|) \cdot f(k-1))^k$, the result holds in this case.

Case 2: π is *not* B-bounded. Then, there is a G-state s_2 s.t. π can be written in the form $\pi = \pi_1 \cdot \pi_2$ so that π_1 is either empty or B-bounded, π_2 starts at state $(s_2)^I = \langle q_2, v_2 \rangle$, and v_2 is not B-bounded. Hence, there is $i \in I$ such that $v_2[i] > B$. Assume that π_1 is not empty and B-bounded (the other case being simpler). Let s_1 be the last state of π_1. As in case 1, we can replace π_1 with a run π'_1 in G^I from s^I to s_1 of length at most $|Q| \cdot B^k$. Let $J = I \setminus \{i\}$ (hence, $|J| = k-1$). Since $(\pi_2)^J$ is a covering in G^J of $(s_f)^J$ w.r.t. $(s_2)^J$, by the ind. hyp., there is a covering π'_2 in G^J of $(s_f)^J$ w.r.t. $(s_2)^J$ of length at most $f(k-1)$. Note that at each step of a run of G, any component of a G-state can decrease at most by $\|\Delta\|$. Thus, since π'_2 has length at most $f(k-1)$, $(s_2)^I = \langle q_2, v_2 \rangle$, and

$v_2[i] > B = \|\Delta\| \cdot f(k-1) + \|s_f\|$, it follows that there exists a covering π_2'' in G^I of $(s_f)^I$ w.r.t. $(s_2)^I$ of length at most $f(k-1)$. Hence, $\pi_1' \cdot \pi_2''$ is a covering in G^I of $(s_f)^I$ w.r.t. s^I of length at most $|Q| \cdot B^k + f(k-1)$. Since $|Q| \cdot B^k \leq |Q| \cdot ((\|\Delta\| + \|s_f\|) \cdot f(k-1))^k$, we are done. □

By solving the recurrence in Lemma 2, we obtain the following result. Hence, Theorem 1 directly follows.

Lemma 3. *For all* $k \in \{0, 1, \ldots, d\}$, $f(k) \leq \big(|Q| \cdot (\|\Delta\| + \|s_f\| + 2)\big)^{(3k)!+1}$.

Proof. By induction on k. The base case $k = 0$ directly follows from Lemma 2. Now, assume that $k > 0$. Let $C = \|\Delta\| + \|s_f\| + 2$. Then,

$$\begin{aligned}
f(k) &\leq |Q| \cdot (C \cdot f(k-1))^k + f(k-1) && \text{by Lemma 2} \\
&\leq |Q| \cdot [(C \cdot f(k-1))^k + f(k-1)] \\
&\leq |Q| \cdot (C \cdot f(k-1))^{k+1} && \text{since } C \cdot f(k-1) \geq 2 \\
&\leq (|Q| \cdot C \cdot f(k-1))^{k+1} \\
&\leq ((|Q| \cdot C)^{(3(k-1))!+2})^{k+1} && \text{by induction hypothesis} \\
&\leq (|Q| \cdot C)^{(3k)!+1}
\end{aligned}$$

□

Note that $min(Pre^*(G, s_f\uparrow))$ constitutes a finite canonical representation of the possibly infinite set $Pre^*(G, s_f\uparrow)$, for which the membership problem (and other basic questions) are decidable.[3] It is well-known that $min(Pre^*(G, s_f\uparrow))$ can be computed by a least fixpoint algorithm [1, 9] refered to as the backward algorithm. However, no elementary upper bound is known on the execution time of this algorithm. By using Theorem 1, we provide in the next section such an upper bound. As a consequence, we derive an upper bound on the cardinality of $min(Pre^*(G, s_f\uparrow))$, which is doubly exponential in the dimension d of G. In Section 4, we show that this double exponential blow-up cannot be avoided.

3 Complexity of the Backward Algorithm for Coverability

First, we recall the standard backward algorithm for coverability [1, 9]. Fix a d-VASS $G = \langle Q, \Delta \rangle$ and a state s_f. We define a monotone infinite sequence $U_0 \subseteq U_1 \subseteq \ldots$ of upward-closed sets of states as: $U_0 = s_f\uparrow$, and $U_{i+1} = U_i \cup Pre(G, U_i)$ for each $i \geq 0$. Since $\trianglelefteq$ (over $Q \times \mathbb{N}^d$) is a wqo, $U_i \subseteq U_{i+1}$ for each $i \geq 0$, and $U_i = U_{i+1}$ iff $min(U_i) = min(U_{i+1})$, by Lemma 1 and definition of the sets U_i, we obtain the following.[4]

Remark 1. For each $i \geq 0$, U_i is the set of states s such that there is a covering of s_f w.r.t. s of length less or equal to i. Moreover, there is $i \geq 0$ such that $min(U_{i+1}) = min(U_i)$. Also, whenever $min(U_{i+1}) = min(U_i)$ for some $i \geq 0$, then $Pre^*(G, s_f\uparrow) = U_i$.

[3] Given $min(U)$ for an upward-closed set U of G-states, one can decide if a given state is in U (membership problem).

[4] Note that $Pre^*(G, s_f\uparrow)$ is the least fixpoint of $\mu X.(s_f\uparrow) \cup Pre(G, X)$.

Remark 2. [1, 9] Given a G-state s, one can compute $min(Pre(G,s\uparrow))$. Hence, for each $i \geq 0$, given $min(U_i)$, one can compute $min(U_{i+1})$ as follows:

$$min(U_{i+1}) = min(min(U_i) \cup \bigcup_{s \in min(U_i)} min(Pre(G,s\uparrow))) \ .$$

Then, the backward algorithm at ith step computes $min(U_i)$. If $min(U_i) = min(U_{i+1})$, then the algorithm terminates and outputs $min(U_i)$. By Remark 1, the algorithm terminates and outputs the basis of $Pre(G,s_f\uparrow)$. Now, we analyze its complexity. Let H be the upper bound in Theorem 1 for G and s_f, i.e., $H = [|Q| \cdot (\|\Delta\| + \|s_f\| + 2)]^{(3d)!+1}$.

Lemma 4. *The sequence $min(U_0), min(U_1), \ldots$ is stable at H, i.e. $min(U_H) = min(U_{H+1})$.*

Proof. By contradiction. Assume that $min(U_H) \neq min(U_{H+1})$. Then, $U_H \neq U_{H+1}$ and since $U_H \subseteq U_{H+1}$, there must be $s \in U_{H+1} \setminus U_H$. By Remark 1, it follows that each covering in G of s_f w.r.t. s has length at least $H+1$. Since $s \in U_{H+1} \subseteq Pre^*(G,s_f\uparrow)$, s_f is coverable from s. Thus, by definition of H and Theorem 1, there must be a covering of s_f w.r.t. s of length at most H, which is a contradiction. □

Lemma 5. *Let S be a* finite *set of states. Then, one can compute a* finite *set B_S of states such that $min(S\uparrow \cup Pre(G,S\uparrow)) \subseteq B_S \subseteq S\uparrow \cup Pre(G,S\uparrow)$, $|B_S|$ is at most $O(|\Delta| \cdot |S|)$, and $\|B_S\|$ is at most $O(\|\Delta\| + \|S\|)$. Moreover, B_S can be computed in time $O(d \cdot |\Delta| \cdot |S| \cdot \log(\|\Delta\| + \|S\| + 2))$.*

Proof. For $v \in \mathbb{Z}^d$, $pos(v)$ denotes the vector in $\mathbb{N}^d$ defined as: $pos(v)[i] = v[i]$ if $v[i] \in \mathbb{N}$, and $pos(v)[i] = 0$ otherwise. Then, $B_S = S \cup A_S$, where A_S is given by

$$A_S = \{\langle q, pos(v' - v)\rangle \mid \langle q, v, q'\rangle \in \Delta \text{ and } \langle q', v'\rangle \in S \text{ for some } q' \in Q\}$$

We show the following, hence, the result easily follows:

1. $A_S \subseteq Pre(G,S\uparrow)$
2. For each $s \in Pre(G,S\uparrow)$, there is $s' \in A_S$ such that $s \trianglerighteq s'$.

Proof of Property 1: let $s \in A_S$. By construction there are $\langle q,v,q'\rangle \in \Delta$ and $\langle q',v'\rangle \in S$ such that $s = \langle q, pos(v'-v)\rangle$. Evidently, it suffices to show that $pos(v'-v) + v \trianglerighteq v'$. Since $pos(v'-v) \trianglerighteq v'-v$, the result follows.

Proof of Property 2: let $s \in Pre(G,S\uparrow)$, where $s = \langle q,v\rangle$ for some $q \in Q$ and $v \in \mathbb{N}^d$. Then, there is $\langle q,v',q'\rangle \in \Delta$ such that $\langle q', v+v'\rangle \in S\uparrow$. Hence, there is $\langle q', v_{min}\rangle \in S$ such that $v + v' \trianglerighteq v_{min}$. Hence, $v \trianglerighteq v_{min} - v'$. Since $v \in \mathbb{N}^d$, we obtain that $v \trianglerighteq pos(v_{min} - v')$. Let $s' = \langle q, pos(v_{min} - v')\rangle$. Note that $s' \in A_S$. Thus, since $s \trianglerighteq s'$, we are done. □

Note that given a *finite* set S of G-states, $min(S)$ can be easily computed in time $O(d \cdot |S|^2 \cdot \log(\|S\| + 2))$. Hence, by Lemma 5, we obtain the following.

Corollary 1. *Let S be a* finite *set of G-states and $S_{min} = min(S\uparrow \cup\ Pre(G,S\uparrow))$. Then, $\|S_{min}\|$ is at most $O(\|\Delta\| + \|S\|)$. Moreover, S_{min} can be computed in time $O(d \cdot |\Delta|^2 \cdot |S|^2 \cdot \log(\|\Delta\| + \|S\| + 2))$.*

By Lemma 4 and Remark 1, $min(Pre^*(G, s_f\uparrow)) = min(U_H)$. Then, Corollary 1 shows that the backward algorithm terminates in time

$$O(H \cdot d \cdot |\Delta|^2 \cdot max_{0 \leq i \leq H} |min(U_i)|^2 \cdot \log(\|\Delta\| + max_{0 \leq i \leq H} \|min(U_i)\| + 2))$$

Note that, by Corollary 1, for each $i \geq 0$, $\|min(U_i)\| = O(i \cdot \|\Delta\| + \|s_f\|)$. Hence, $|min(U_i)|$ is at most $O(|Q| \cdot (i \cdot \|\Delta\| + \|s_f\|)^d)$. Also, $max_{0 \leq i \leq H} \|min(U_i)\| = O(H \cdot \|\Delta\| + \|s_f\|)$ and $max_{0 \leq i \leq H} |min(U_i)|^2 = O(|Q|^2 \cdot (H \cdot \|\Delta\| + \|s_f\|)^{2d})$. Therefore, since $H = \big(|Q| \cdot (\|\Delta\| + \|s_f\| + 2)\big)^{2^{O(d \cdot \log d)}}$, we obtain the following.

Theorem 2. *The backward algorithm terminates in time* $\big(|Q| \cdot (\|\Delta\| + \|s_f\| + 2)\big)^{2^{O(d \cdot \log d)}}$, $\|min(Pre^*(G, s_f\uparrow))\|$ *and* $|min(Pre^*(G, s_f\uparrow))|$ *are at most* $\big(|Q| \cdot (\|\Delta\| + \|s_f\| + 2)\big)^{2^{O(d \cdot \log d)}}$.

4 Lower Bound

In this section, we prove the following result by an adaptation of Lipton's proof of EXPSPACE-hardness for reachability in VASS [13].

First, we need the following notation. Let $G = \langle Q, \Delta \rangle$ be a d-VASS and $q \in Q$. We denote by $q\uparrow$ the upward-closed set $\{q\} \times \mathbb{N}^d$ of G-states. Also, for a set S of G-states, we denote by $[Pre^*(G, S)]_q$ the subset of $\mathbb{N}^d$ given by $\{v \in \mathbb{N}^d \mid \langle q, v \rangle \in Pre^*(G, S)\}$.

Theorem 3. *For each* $n \in \mathbb{N}$*, one can build a* $O(n)$*-VASS* $G_n = \langle Q_n, \Delta_n \rangle$ *and* $q_n \in Q_n$ *s.t.* $|Q_n| = O(n)$*,* $|\Delta_n| = O(n)$*,* $\|\Delta_n\| = 1$*, and the following holds:* (1) $|min(Pre^*(G_n, q_n\uparrow))|$ *is at least* 2^{2^n} *(hence,* $\|min(Pre^*(G_n, q_n\uparrow)))\|$ *is at least* $2^{2^{\Omega(n)}}$*), and* (2) *there are states* $s \in min(Pre^*(G_n, q_n\uparrow))$ *s.t. each run from* s *to a state in* $q_n\uparrow$ *has length at least* 2^{2^n}.

By Property 2 in Theorem 3 and the results in the previous section, we easily deduce the following.

Corollary 2. *Let* $n \in \mathbb{N}$*,* $G_n = \langle Q_n, \Delta_n \rangle$ *and* $q_n \in Q_n$ *as in Theorem 3. Then, the number of iterations of the backward algorithm with input* G_n *and* $\langle q_n, \langle 0, \ldots, 0 \rangle \rangle$ *is at least* 2^{2^n}.

To make clear the proof of Theorem 3, we consider an high-level variant of VASS, called *net Programs* [6], corresponding to a subclass of nondeterministic counter machines with nonrecursive subroutines. Then, we show that in order to prove Theorem 3, it is sufficient to prove a similar result for net programs. Finally, in Section 4.2, we prove the variant of Theorem 3 for net programs.

For $m, k \in \mathbb{N}^+$ s.t. $k \leq m$ and $U \subseteq \mathbb{N}^m$, $\Pi_k(U)$ denotes the subset of $\mathbb{N}^k$ given by $\{v \in \mathbb{N}^k \mid \langle v[1], \ldots, v[k], 0, \ldots, 0 \rangle \in U\}$. Note that $\Pi_k(U)$ is upward-closed if U is upward-closed. Moreover, the following holds.

Lemma 6. *Let* $m, k \in \mathbb{N}^+$ *such that* $k \leq m$ *and* U *be an upward-closed subset of* $\mathbb{N}^m$*. Then,* $|min(U)| \geq |min(\Pi_k(U))|$.

Proof. For $v \in \mathbb{N}^k$, we denote by $v \cdot \underline{0}$ the vector in $\mathbb{N}^m$ given by $\langle v[1], \ldots, v[k], 0, \ldots, 0 \rangle$. Let $v \in min(\Pi_k(U))$. We show that $v \cdot \underline{0} \in min(U)$, hence, the result follows. Since $v \cdot \underline{0} \in U$, there is $v' \in min(U)$ such that $v' \trianglelefteq v \cdot \underline{0} \in U$. Hence, $v' = v'' \cdot \underline{0}$, $v'' \in \Pi_k(U)$, and $v'' \trianglelefteq v$. Since $v \in min(\Pi_k(U))$, it follows that $v'' = v$, hence $v \cdot \underline{0} = v' \in min(U)$, and we are done. □

4.1 Net Programs

A net program is similar to a nondeterministic Minsky counter machine, but does not have the ability to test a (counter) variable for zero. However, it has the possibility of transferring control to a subroutine (or subprogram). Formally, a net program P on a finite set $\{x_1,\ldots,x_d\}$ of (counter) variables is a tuple $P = \langle ID_1,\ldots,ID_n,\texttt{Code}\rangle$, where $ID_1,\ldots,ID_n$ are pairwise distinct *subprogram identifiers*, and Code assigns to each $1 \leq p \leq n$, the *code* $\texttt{Code}(ID_p)$ of subprogram ID_p, which is a sequence of the form

$$\texttt{Code}(ID_p) = \texttt{l}_1 : I_1;\ldots \texttt{l}_{k-1} : I_{k-1};\texttt{l}_k : \textbf{return};$$

where $k \geq 1$, $\texttt{l}_1,\ldots,\texttt{l}_k$ are pairwise distinct (instruction) *labels*, $\texttt{l}_1$ (resp., $\texttt{l}_k$) is the initial (resp., final) label of subprogram ID_p, and each I_j is an instruction of the form:

- *increment*: $x_i := x_i + 1$ (where $1 \leq i \leq d$),
- *decrement*: $x_i := x_i - 1$ (where $1 \leq i \leq d$),
- *unconditional jump*: **goto** $\texttt{l}$ (where $\texttt{l} \in \{\texttt{l}_1,\ldots,\texttt{l}_k\}$),
- *nondeterministic jump*: **goto** $\texttt{l}$ **or goto** $\texttt{l}'$ (where $\texttt{l},\texttt{l}' \in \{\texttt{l}_1,\ldots,\texttt{l}_k\}$),
- *subprogram call*: **call** $\texttt{ID}_i$ (where $i > p$).[5]

Additionally, we require that labels of distinct subprograms are distinct as well. The subprogram ID_1 is called the *main subprogram* of P, and the initial (resp., final) label of P is the initial (resp., final) label of the main subprogram. For each (instruction) label $\texttt{l}$ of P, we denote by $ID(\texttt{l})$ the identifier of the unique subprogram having $\texttt{l}$ as label. Moreover, if $\texttt{l}$ is the label of a call instruction, we denote by $called(\texttt{l})$ the identifier of the called subprogram. Now, we formally define the semantics of net programs. An *extended label* of the net program P above is a pair of the form $\langle C,\texttt{l}\rangle$, where $\texttt{l}$ is a label of P and C is a *caller context*, i.e., a (possibly empty) sequence of P-labels $C = \texttt{l}_1\ldots\texttt{l}_k$ such that the following holds: (i) each $\texttt{l}_i$ is the label of a call instruction, and (ii) if C is nonempty, then $ID(\texttt{l}_{i+1}) = called(\texttt{l}_i)$ for each $1 \leq i \leq k$, where $\texttt{l}_{k+1} = \texttt{l}$. Note that the set of extended labels of P, written $EL(P)$, is finite. A *P-state* is a pair $\langle\langle C,\texttt{l}\rangle,v\rangle$, where $\langle C,\texttt{l}\rangle \in EL(P)$ and $v \in \mathbb{N}^d$ is a valuation of variables $\{x_1,\ldots,x_d\}$ assigning to each variable x_i, the value $v[i]$. The net program P induces a transition relation $\to$ over P-states, as follows $\langle\langle C,\texttt{l}\rangle,v\rangle \to \langle\langle C',\texttt{l}'\rangle,v'\rangle$ iff:

- if $\texttt{l}$ is the label of an increment (resp., decrement, resp., jump) instruction, then $\langle\langle C',\texttt{l}'\rangle,v'\rangle$ is as expected (note that $C' = C$ and if $\texttt{l}$ is the label of a decrement, then $\langle\langle C,\texttt{l}\rangle,v\rangle$ has a successor iff the value in v of the decremented variable is greater than 0);
- if $\texttt{l}$ is the label of a call instruction "**call** ID_j", then $v' = v$, $C' = C\cdot\texttt{l}$, and $\texttt{l}'$ is the initial label of subprogram ID_j;
- if $\texttt{l}$ is the label of a return instruction, then $v' = v$, $C = C'\cdot\texttt{l}''$ for some $\texttt{l}''$, and $\texttt{l}'$ is the label which follows the call instruction label $\texttt{l}''$ in $\texttt{Code}(ID(\texttt{l}''))$.

A run or execution of P is a finite sequence $s_1,\ldots,s_h$ of P-states such that $s_i \to s_{i+1}$ for each $1 \leq i < h$. For a set S of P-states, let $Pre^*(P,S)$ be the set of P-states s such that there is a run of P from s leading to some P-state in S. For each label $\texttt{l}$, we denote by $[Pre^*(P,S)]_{\texttt{l}}$ the set $\{v \in \mathbb{N}^d \mid \langle\langle\varepsilon,\texttt{l}\rangle,v\rangle \in Pre^*(P,S)\}$, and by $\texttt{l}\!\uparrow$ the set of P-states

[5] The requirement $i > p$ ensures that there are no recursive calls.

$\{\langle \varepsilon, \mathtt{l} \rangle\} \times \mathbb{N}^d$. It is easy to show that if S is an upward-closed set of P-states, then $Pre^*(P,S)$ is upward-closed as well. The following result allows us to reduce the proof of Theorem 3 to its variant for the class of net programs.

Theorem 4. *Let P be a net program on $\{x_1,\ldots,x_d\}$, k be the number of call instructions of P, and* `start` *and* `end` *be the initial and final labels of P. Then, one can build in* linear-time *a* $(d+k)$-VASS $G = \langle Q,V \rangle$ *such that Q is the set of P-labels, $\|\Delta\| = 1$, $|\Delta| \leq 2 \cdot N$, where N is the number of P-instructions, and $|min([Pre^*(G,\mathtt{end}\uparrow)]_{\mathtt{start}})| \geq |min([Pre^*(P,\mathtt{end}\uparrow)]_{\mathtt{start}})|$. Moreover, for each $s \in min(Pre^*(P,\mathtt{end}\uparrow))$, there is a G-state $s' \in min(Pre^*(G,\mathtt{end}\uparrow))$ such that for each run π in G from s' to a G-state in $\mathtt{end}\uparrow$, there is a run of P from s to a P-state in $\mathtt{end}\uparrow$ of length $|\pi|$.*

Proof. Let $L = \{\mathtt{l}_1,\ldots,\mathtt{l}_k\}$ be the set of call instruction labels of P. The $(d+k)$-VASS $G = \langle Q,\Delta \rangle$ is defined as follows (intuitively, we use an additional dimension for each call instruction label of P): Q is the set of P-labels and the set of transitions Δ is obtained in the following way:

- for each increment "$\mathtt{l} : x_i := x_i + 1; \mathtt{l}' : I; \ldots$", we add the transition $\langle \mathtt{l}, v, \mathtt{l}' \rangle$, where $v[i] = 1$ and $v[j] = 0$ for $j \neq i$;
- for each decrement "$\mathtt{l} : x_i := x_i - 1; \mathtt{l}' : I; \ldots$", we add the transition $\langle \mathtt{l}, v, \mathtt{l}' \rangle$, where $v[i] = -1$ and $v[j] = 0$ for $j \neq i$;
- for each unconditional jump "$\mathtt{l}$: **goto** $\mathtt{l}'; \ldots$", we add transition $\langle \mathtt{l}, \underline{0}^{d+k}, \mathtt{l}' \rangle$;
- for each nondeterministic jump "$\mathtt{l}$: **goto** $\mathtt{l}'$ **or goto** $\mathtt{l}''; \ldots$", we add two transitions given by $\langle \mathtt{l}, \underline{0}^{d+k}, \mathtt{l}' \rangle$ and $\langle \mathtt{l}, \underline{0}^{d+k}, \mathtt{l}'' \rangle$;
- for each call instruction "$\mathtt{l}_i$: **call** $ID_p; \mathtt{l} : I; \ldots$" (where $1 \leq i \leq k$), we add two transitions $\langle \mathtt{l}_i, v_+, \mathtt{l}_0 \rangle$ and $\langle \mathtt{l}_f, v_-, \mathtt{l} \rangle$, where: (i) $\mathtt{l}_0$ (resp., $\mathtt{l}_f$) is the initial (resp., final) label of subprogram ID_p, (ii) $v_+[d+i] = 1$ and $v_+[j] = 0$ for $j \neq d+i$, and (iii) $v_-[d+i] = -1$ and $v_-[j] = 0$ for $j \neq d+i$.

Note that $\|\Delta\| = 1$ and $|\Delta| \leq 2 \cdot N$, where N is the number of P-instructions. Now, we establish the correspondence between the runs of P and the runs of G. Let H be the mapping assigning to each state s of P of the form $\langle \langle \mathtt{l}_{i_1} \ldots \mathtt{l}_{i_p}, \mathtt{l} \rangle, v \rangle$, the G-state $H(s)$ defined as follows (note that $i_1,\ldots,i_p \in \{1,\ldots,k\}$ and are pairwise distinct)[6]: $H(s) = \langle \mathtt{l}, v_{ext} \rangle$, where for each $1 \leq j \leq d+k$, $v_{ext}[j] = v[j]$ if $j \leq d$, $v_{ext}[j] = 1$ if $j = d + i_h$ for some $1 \leq h \leq p$, and $v_{ext}[j] = 0$ otherwise. By construction, we obtain the following:

Claim: let $s_0,s_1,\ldots,s_n$ be a sequence of states of P. Then, $s_0,s_1,\ldots,s_n$ is a run of P if and only if $H(s_0),H(s_1),\ldots,H(s_n)$ is a run of G. Moreover, for each state s'_0 of P, each run of G from $H(s'_0)$ has the form $H(s'_0),H(s'_1),\ldots,H(s'_m)$ for some sequence $s'_1,\ldots,s'_m$ of P-states.

By the claim above, it follows that $\Pi_d([Pre^*(G,\mathtt{end}\uparrow)]_{\mathtt{start}}) = [Pre^*(P,\mathtt{end}\uparrow)]_{\mathtt{start}}$. Thus, by Lemma 6 and the claim above, Theorem 4 easily follows. □

4.2 Proof of Theorem 3

Theorem 3 directly follows from Theorem 4 and the following result.

[6] Moreover, note that $called(\mathtt{l}_{i_1}),\ldots,called(\mathtt{l}_{i_k})$ are pairwise distinct and $called(\mathtt{l}_{i_k}) = ID(\mathtt{l})$.

Theorem 5. *For each $n \in \mathbb{N}$, one can build a net program P_n with initial (resp., final) label* `start` *(resp.,* `end`*), $O(n)$ instructions, and $O(n)$ variables such that $|min([Pre^*(P_n, \texttt{end}\uparrow)]_{start})| \geq 2^{2^n}$. Also, there exists $v \in min([Pre^*(P_n, \texttt{end}\uparrow)]_{start})$ such that each run from $\langle\langle \varepsilon, \texttt{start}\rangle, v\rangle$ to a state in $\texttt{end}\uparrow$ has length at least 2^{2^n}.*

In the rest of this section, we prove Theorem 5.

Construction of P_n. Let $n \in \mathbb{N}$, define $Var_n = \{w_1, w_2, y_n, \overline{y}_n\} \cup \bigcup_{i=0}^{n-1} \{y_i, \overline{y}_i, z_i, \overline{z}_i\}$. The net program P_n has set of variables Var_n and is given by

$$\langle Main_n, Lipton_n, Init_0, \dots, Init_{n-1}, Dec_n(\overline{y}_n), Dec_{n-1}(\overline{y}_{n-1}), Dec_{n-1}(\overline{z}_{n-1}), \dots \\ Dec_0(\overline{y}_0), Dec_0(\overline{z}_0), Set_n, \texttt{Code}\rangle$$

where `Code` is given in Figures 1–3.[7]

The construction of P_n ensures the following: if initially (i.e., at call time of the main subprogram $Main_n$) each variable in $Var_n \setminus \{w_1, w_2\}$ has value 0, then the main subprogram $Main_n$ *can return*[8] if and only if the sum of the initial values of w_1 and w_2 is greater or equal to 2^{2^n}. Now, we proceed with the description of the various subprograms of P_n. The main subprogram $Main_n$ simply calls the subprograms Set_n and $Lipton_n$ (in the given order) and returns. It is easy to check (see Figure 1) that the subprogram Set_n ensures the following.

<u>$Main_n$</u> :

`start` : **call** Set_n;
  call $Lipton_n$;
`end` : **return**.

<u>$Lipton_n$</u> :

`start` : **call** $Init_0$;
  ......
  call $Init_{n-1}$;
  call $Dec_n(\overline{y}_n)$;
`end` : **return**.

<u>Set_n</u> :

`start` : **goto** `0` **or goto** `end`;
`0` : **goto** `1` **or goto** `2`;
`1` : $w_1 := w_1 - 1; \overline{y}_n = \overline{y}_n + 1$;
  goto `start`;
`2` : $w_2 := w_2 - 1; \overline{y}_n = \overline{y}_n + 1$;
  goto `start`;
`end` : **return**.

Fig. 1. The subprograms $Main_n$, $Lipton_n$, and Set_n of P_n

Lemma 7. *Assume that Set_n is called with the value of $\overline{y}_n$ being 0. Then: (1) whenever Set_n returns, the value of $\overline{y}_n$ is less or equal to the sum of the initial values (at call time of Set_n) of variables w_1 and w_2, and (2) there is an execution such that Set_n returns with the value of $\overline{y}_n$ being exactly the sum of the initial values of w_1 and w_2.*

[7] In Figures 1–3, for clarity, some instruction labels are omitted, and some labels of distinct subprograms are equal (we tacitely assume that they are prefixed by the ID of the associated subprogram).

[8] i.e., there is a run leading to a state whose label is the final label of subprogram $Main_n$.

The subprogram $Lipton_n$ (see Figure 1), whose construction corresponds to a variant of that given by Lipton in [13] (see also [6]), ensures the following: if initially (i.e., at call time of $Lipton_n$) all the variables in $Var_n \setminus \{w_1, w_2, \overline{y}_n\}$ have value 0, then $Lipton_n$ can return if and only if the initial value of $\overline{y}_n$ is greater or equal to 2^{2^n}. The implementation of $Lipton_n$ is based on subprograms $Init_i$, $Dec_i(\overline{z}_i)$, and $Dec_j(\overline{y}_j)$ (where $0 \leq i \leq n-1$ and $0 \leq j \leq n$). The subprograms $Dec_i(\overline{z}_i)$ and $Dec_j(\overline{y}_j)$ (see Figure 2) ensure the following.

<u>$Dec_0(\overline{x}_0)$</u> :

* x_0 is either y_0 or z_0 *
`start` : $\overline{x}_0 := \overline{x}_0 - 1;$
$\overline{x}_0 := \overline{x}_0 - 1;$
$x_0 := x_0 + 1;$
$x_0 := x_0 + 1;$
`end` : **return**.

<u>$Dec_{i+1}(\overline{x}_{i+1})$</u> :

* x_{i+1} is either y_{i+1} or z_{i+1} *
* Initially, $y_i = z_i = 2^{2^i}$ and $\overline{y}_i = \overline{z}_i = 0$ *
`out-loop` : $y_i := y_i - 1; \overline{y}_i := \overline{y}_i + 1;$
`in-loop` : $z_i := z_i - 1; \overline{z}_i := \overline{z}_i + 1;$
$\overline{x}_{i+1} := \overline{x}_{i+1} - 1; x_{i+1} := x_{i+1} + 1;$
goto `in-continue` **or goto** `in-exit`;
`in-continue` : $z_i := z_i - 1; z_i := z_i + 1;$ **goto** `in-loop`;
`in-exit` : **call** $Dec_i(\overline{z}_i)$;
goto `out-continue` **or goto** `out-exit`;
`out-continue` : $y_i := y_i - 1; y_i := y_i + 1;$ **goto** `out-loop`;
`out-exit` : **call** $Dec_i(\overline{y}_i)$;
`end` : **return**.

Fig. 2. The subprograms $Dec_0(\overline{x}_0)$ and $Dec_{i+1}(\overline{x}_{i+1})$ of P_n

Lemma 8. *Let $0 \leq j \leq n$ and $x_j \in \{y_j, z_j\}$ such that $x_j = y_j$ if $j = n$. Assume that $Dec_j(\overline{x}_j)$ is called with the values of $\overline{y}_h$ and $\overline{z}_h$ being 0 and the values of y_h and z_h being 2^{2^h} for each $0 \leq h < j$. Then, the following holds:*

- *$Dec_j(\overline{x}_j)$ can return* iff *the initial value of $\overline{x}_j$ (at call time of $Dec_j(\overline{x}_j)$) is at least 2^{2^j}. Moreover, if the initial value of $\overline{x}_j$ is exactly 2^{2^j} and the initial value of x_j is 0, then whenever $Dec_j(\overline{x}_j)$ returns, the values of $\overline{x}_j$ and x_j (at return time) are swapped (i.e., x_j has value 2^{2^j} and $\overline{x}_j$ has value 0).*
- *Whenever $Dec_j(\overline{x}_j)$ returns, there are no side-effects on the variables $x \in Var_n \setminus \{\overline{x}_j, x_j\}$ (the values of x at call and return times are the same).*
- *Whenever $Dec_j(\overline{x}_j)$ returns, the number of computational steps from the call time to the return time is at least 2^{2^j}.*

Proof. The proof is by induction on j. The base case ($j = 0$) is trivial (see Figure 2). Now, assume that $j = i+1$ for some $0 \leq i < n$. Let us consider the code of $Dec_{i+1}(\overline{x}_{i+1})$ in Figure 2, which consists of two nested loops: the inner loop is associated with the counter variable z_i, while the outer loop is associated with the counter variable y_i. Note that the body of the inner loop decrements $\overline{x}_{i+1}$. Essentially, since the initial values of y_i and z_i are 2^{2^i}, each of two nested loops can be executed 2^{2^i}-times. Since $2^{2^i} \cdot 2^{2^i} = 2^{2^{i+1}}$, it follows that $\overline{x}_{i+1}$ can be decreased by $2^{2^{i+1}}$. Fix $x_i \in \{y_i, z_i\}$. First, note that at each step the invariant $x_i + \overline{x}_i = 2^{2^i}$ is preserved. Moreover, for the loop associated with counter variable x_i, $Dec_{i+1}(\overline{x}_{i+1})$ can *guess* that the continuation (resp., exit) condition is satisfied, i.e., $x_i > 0$ (resp., $x_i = 0$), by a nondeterministic jump instruction. The continuation condition is implemented by decrementing and then incrementing x_i, while the exit condition is implemented by a call to $Dec_i(\overline{x}_i)$. By the induction hypothesis, $Dec_i(\overline{x}_i)$ can return if and only if $\overline{x}_i$ has value 2^{2^i}, i.e., x_i has value 0. Thus, if the *guess* is not correct, the subprogram $Dec_{i+1}(\overline{x}_{i+1})$ stops without returning. Moreover, by the induction hypothesis, whenever $Dec_i(\overline{x}_i)$ returns, the values of x_i and $\overline{x}_i$ are swapped. This ensures that the inner loop can be re-initialized correctly, and whenever $Dec_{i+1}(\overline{x}_{i+1})$ returns, the values of x_i and $\overline{x}_i$ correspond to the initial ones. Thus, it follows that $Dec_{i+1}(\overline{x}_{i+1})$ can return if and only if $\overline{x}_{i+1}$ can be decreased by $2^{2^{i+1}}$ (i.e., the initial value of $\overline{x}_{i+1}$ is at least $2^{2^{i+1}}$). Finally, if the initial value of $\overline{x}_{i+1}$ is $2^{2^{i+1}}$ and the initial value of x_{i+1} is 0, then the body of the inner loop of $Dec_{i+1}(\overline{x}_{i+1})$ ensure that at return time, the values of x_{i+1} and $\overline{x}_{i+1}$ are swapped. □

Finally, for each $0 \leq i \leq n-1$, the subprogram $Init_i$ (see Figure 3) is used to set the values of y_i and z_i to 2^{2^i}. More precisely, $Init_i$ ensures the following.

$\underline{Init_0}$:

```
start : y0 := y0 + 1;
        y0 := y0 + 1;
        z0 := z0 + 1;
        z0 := z0 + 1;
end   : return.
```

$\underline{Init_{i+1}}$:

```
    out-loop : yi := yi − 1; ȳi := ȳi + 1;
     in-loop : zi := zi − 1; z̄i := z̄i + 1;
               yi+1 := yi+1 + 1; zi+1 := zi+1 + 1;
               goto in-continue or goto in-exit;
 in-continue : zi := zi − 1; zi := zi + 1; goto in-loop;
     in-exit : call Deci(z̄i);
               goto out-continue or goto out-exit;
out-continue : yi := yi − 1; yi := yi + 1; goto out-loop;
    out-exit : call Deci(ȳi);
         end : return.
```

Fig. 3. The subprograms $Init_0$ and $Init_{i+1}$ of P_n

Lemma 9. *Let* $0 \leq j \leq n-1$. *Assume that* $Init_j$ *is called with the following condition being satisfied at call time: (i) the values of* $y_j, z_j, \overline{y}_j, \overline{z}_j$ *are* 0, *and (ii) the values of* $\overline{y}_h, \overline{z}_h$ *are* 0 *and the values of* y_h *and* z_h *are* 2^{2^h} *for each* $0 \leq h < j$. *Then,* $Init_i$ *can return. Moreover, whenever* $Init_i$ *returns,* y_i *and* z_i *have value* 2^{2^i} *and there are no-side effects for the other variables* $x \in Var_n \setminus \{y_i, z_i\}$ (*i.e., the values of* x *at call and return times are the same*).

Proof. The proof is by induction on j. The base case ($j = 0$) is trivial (see Figure 3). Now, assume that $j = i+1$ for some $0 \leq i < n-1$. Let us consider the code of $Init_{i+1}$ in Figure 3, which is the same as $Dec_{i+1}(\overline{x}_{i+1})$, with the unique difference that the body of the inner loop increments the two variables y_{i+1} and z_{i+1}. Hence, reasoning as in the proof of Lemma 8, the result easily follows (in particular, under the considered assumptions, whenever $Init_{i+1}$ returns, the values of y_{i+1} and z_{i+1} are increased exactly by $2^{2^{i+1}}$). □

Assume that at call time of subprogram $Lipton_n$, each variable in $Var_n \setminus \{w_1, w_2, \overline{y}_n\}$ has value 0. Then, By Lemmata 8 and 9, $Lipton_n$ can return iff at call time $\overline{y}_n$ has value at least 2^{2^n}. Moreover, whenever $Lipton_n$ returns, then the number of computational steps from the call time to the return time is at least 2^{2^n}. Thus, by Lemma 7, we obtain the following.

Lemma 10. *Assume that at call time of $Main_n$ each variable in $Var_n \setminus \{w_1, w_2\}$ has value* 0*. Then, $Main_n$ can return* iff *the sum of the values of w_1 and w_2 at call time is at least 2^{2^n}. Moreover, whenever $Main_n$ returns, then the number of computational steps from the call time to the return time is at least 2^{2^n}.*

Proof of Theorem 5. First, we need an additional result. For all $n \in \mathbb{N}$, we denote by Λ_n and $\Upsilon_n \subseteq \Lambda_n$ the subsets of $\mathbb{N}^2$ given by

$$\Lambda_n = \{v \in \mathbb{N}^2 \mid v[1] + v[2] \geq 2^{2^n}\} \text{ and } \Upsilon_n = \{v \in \mathbb{N}^2 \mid v[1] + v[2] = 2^{2^n}\}$$

Lemma 11. *Let $n, m \in \mathbb{N}$ and U be an upward-closed subset of $\mathbb{N}^{m+2}$ such that $\Pi_2(U) = \Lambda_n$. Then, $|min(U)| \geq 2^{2^n}$ and $min(U) \supseteq \{v \cdot \underline{0}^m \mid v \in \Upsilon_n\}$.*

Proof. For $v \in \mathbb{N}^2$, we denote by $v \cdot \underline{0}$ the vector in $\mathbb{N}^{m+2}$ given by $\langle v[1], v[2], 0, \ldots, 0 \rangle$. First,we show the following.

Claim 1: $\Upsilon_n \subseteq min(\Lambda_n)$

Proof of Claim 1: Let $v \in \Upsilon_n$. Since $v \in \Lambda_n$, there must be $v_{min} \in min(\Lambda_n)$ such that $v_{min} \trianglelefteq v$ (note that Λ_n is upward-closed). By definition of Υ_n, it follows that $v_{min} \in \Upsilon_n$. Thus, since all elements in Υ_n are pairwise incomparable, we obtain that $v_{min} = v$. Hence, $v \in min(\Lambda_n)$, and the result follows. □

Moreover, by the proof of Lemma 6, the following holds

Claim 2: $min(U) \supseteq \{v \cdot \underline{0} \mid v \in min(\Pi_2(U))\}$**.**

Evidently, Υ_n has cardinality 2^{2^n}. By hypothesis, $\Pi_2(U) = \Lambda_n$. Thus, by Claims 1 and 2, the result follows. □

Fix $n \in \mathbb{N}$ and an ordering of Var_n such that w_1 and w_2 precede all the other variables. Let `start` (resp., `end`) be the initial (resp., final) label of P_n. By construction, P_n has $O(n)$ instructions and $O(n)$ variables. Thus, by Lemmata 10 and 11, Theorem 5 easily follows.

Acknowledgement. We would like to thank Rupak Majumdar for asking about the complexity of the backward algorithm and Javier Esparza for encouraging us.

References

[1] Abdulla, P., Čerāns, K., Jonsson, B., Tsay, Y.: General Decidability Theorems for Infinite-State Systems. In: LICS 1996, pp. 313–321. IEEE Computer Society Press, Los Alamitos (1996)

[2] Arnold, A., Latteux, M.: Recursivite et cones rationnels fermes par intersection. Calcolo 15, 381–394 (1978)

[3] Delzanno, G., Raskin, J.-F.: Symbolic representation of upward-closed sets. In: Graf, S. (ed.) TACAS 2000. LNCS, vol. 1785, pp. 426–440. Springer, Heidelberg (2000)
[4] Delzanno, G., Raskin, J.-F., Van Begin, L.: Attacking symbolic state explosion. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 298–310. Springer, Heidelberg (2001)
[5] Dickson, L.E.: Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. American Journal of Mathematics 35, 413–422 (1913)
[6] Esparza, J.: Decidability and Complexity of Petri Net Problems - An Introduction. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 374–428. Springer, Heidelberg (1998)
[7] Esparza, J., Nielsen, M.: Decidability issues for Petri nets - a survey. Journal of Informatik Processing and Cybernetics 30(3), 143–160 (1994)
[8] Figueira, D., Figueira, S., Schmitz, S., Schnoebelen, P.: Ackermannian and primitive-recursive bounds with Dickson's lemma. In: LICS 2011: Proc. 26th Annual IEEE Symp. on Logic in Computer Science, pp. 269–278. IEEE Computer Society Press, Los Alamitos (2011)
[9] Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theoretical Computer Science 256(1-2), 63–92 (2001)
[10] Ganty, P., Majumdar, R.: Algorithmic verification of asynchronous programs. CoRR, abs/1011.0551 (2010)
[11] Higman, G.: Ordering by divisibility in abstract algebras. Proceedings of the London Mathematical Society (3) 2(7), 326–336 (1952)
[12] Karp, R.M., Miller, R.E.: Parallel program schemata. Journal of Comput. Syst. Sci. 3(2), 147–195 (1969)
[13] Lipton, R.: The Reachability Problem Requires Exponential Space. Technical Report 62, Yale University (1976)
[14] Rackoff, C.: The Covering and Boundedness Problems for Vector Addition Systems. Theoretical Computer Science 6, 223–231 (1978)
[15] Sen, K., Viswanathan, M.: Model checking multithreaded programs with asynchronous atomic methods. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 300–314. Springer, Heidelberg (2006)
[16] Valk, R., Jantzen, M.: The residue of vector sets with applications to decidability problems in Petri nets. Acta Informatica 21, 643–674 (1985)
[17] Yen, H.-C., Chen, C.-L.: On minimal elements of upward-closed sets. Theoretical Computer Science 410(24-25), 2442–2452 (2009)