Bisimilarity of Pushdown Automata is Nonelementary

Michael Benedikt
University of Oxford

Stefan Göller
University of Bremen

Stefan Kiefer
University of Oxford

Andrzej S. Murawski
University of Warwick

***Abstract*—Given two pushdown automata, the bisimilarity problem asks whether the infinite transition systems they induce are bisimilar. While this problem is known to be decidable our main result states that it is nonelementary, improving EXPTIME-hardness, which was the best previously known lower bound for this problem. Our lower bound result holds for normed pushdown automata as well.**

I. Introduction

A central problem in theoretical computer science is to decide whether two machines or systems behave equivalently. While being generally undecidable for Turing machines, a lot of research has been devoted to finding subclasses of machine devices for which this problem becomes decidable. *Equivalence checking* is the problem of determining whether two systems are semantically identical.

It is well-known that even language equivalence of pushdown automata is undecidable, in fact already their universality is undecidable. On the positive side, a celebrated result due to Sénizergues states that language equivalence of deterministic pushdown automata is decidable [1]. The best known upper bound for the latter problem is a tower of exponentials [2] (see [3] for a more recent proof), while only hardness of deterministic polynomial time is known to date.

Among the numerous notions of equivalence [4] in the realm of formal verification and concurrency theory, the central one is *bisimulation equivalence* (*bisimilarity* for short), which enjoys pleasant mathematical properties. It can be seen to take the king role: There are important characterizations of the bisimulation-invariant fragments of first-order logic and of monadic second-order logic in terms of modal logic [5] and of the modal μ-calculus [6], respectively. In particular, bisimilarity is a fundamental notion for process algebraic formalisms [7]. As a result, a great deal of research in the analysis of infinite-state systems (such as pushdown automata or Petri nets) has been devoted to deciding bisimilarity of two given processes, see e.g. [8] for a comprehensive overview.

A milestone result in this context has been proven by Sénizergues: Bisimilarity on pushdown systems (i.e. transition systems induced by pushdown automata) is decidable [9]. Since a pushdown system can be viewed as an abstraction of the call-and-return behavior of a recursive program, one can read this decidability result as that one can decide equivalence of recursive programs in terms of their visible behavior.

In [9] bisimilarity is proven to be decidable even for the more general class of equational graphs of finite out-degree. Concerning *decidability* this result can in some sense be considered as best possible since on the slightly more general classes of type-1 rewrite systems [10] and order-two pushdown graphs [11] bisimilarity becomes undecidable.

Sénizergues' algorithm for deciding bisimilarity of pushdown systems consists of two semi-decision procedures and in fact no complexity-theoretic upper bound is known for this problem to date. On the other hand, the best known lower bound for this problem is EXPTIME shown by Kučera and Mayr [12]. In [13] EXPTIME-hardness has been established even for the subclass of basic process algebras, for which a 2EXPTIME upper bound is known [14] (in [15] a simpler proof has recently been announced). Such complexity gaps are typical in the context of infinite-state systems.

In fact, in case decidability is known, the precise *computational complexity* status of bisimilarity on infinite-state systems is known only for few classes, including basic parallel processes (communication-free Petri nets) [16] and one-counter systems (the transition systems induced by pushdown automata over a singleton stack alphabet) [17].

Our contribution. The main result of this paper states that bisimilarity of (systems induced by) pushdown automata is nonelementary, even in the normed case. We give small descriptions of pushdown systems on which a bisimulation game is implemented that allows to push and verify encodings of nonelementarily big counters à la Stockmeyer [18]. As an important technical tool we realize deterministic verification phases in the bisimulation game by simulating non-erasing real-time transducers that are fed with the stack content. As building blocks, we use the well-established technique of Defender's forcing [10]. We are optimistic that our technique gives new insights for potential further lower bounds for bisimilarity of PA processes, regularity for pushdown systems, and weak bisimilarity of basic process algebras.

Organisation. In Section II we introduce preliminaries. Section III overviews the ideas in the proof. In Section IV we recall basics on transductions, introduce useful abbreviations for pushdown rules. We also recall the "forcing" technique. Section V explains the key construction that allows a check for bisimilarity to model manipulations on counters for large integers. Section VI consists of our nonelementary lower bound proof for bisimilarity of pushdown automata, while Section VII extends this to PDAs that satisfy an additional condition, normedness. Section VIII gives conclusions.

II. Preliminaries

By $\mathbb{N} \stackrel{\text{def}}{=} \{0,1,\ldots\}$ we denote the set of *non-negative integers*. For $n,m \in \mathbb{N}$ we write $[n,m]$ for $\{n,n+1,\ldots,m\}$; in particular note that $[n,m] = \emptyset$ if $n > m$.

A *labelled transition system (LTS)* is a tuple $\mathcal{S} = (S, \mathcal{A}ct, \{\xrightarrow{a} \mid a \in \mathcal{A}ct\})$, where S is a set of *configurations*, $\mathcal{A}ct$ is a finite set of *action labels*, and $\xrightarrow{a} \subseteq S \times S$ is a *transition relation* for each $a \in \mathcal{A}ct$. We say that $s \in S$ is a *deadlock* state if it is not the case that there exist $a \in \mathcal{A}ct$ and $t \in S$ such that $s \xrightarrow{a} t$. A binary relation $R \subseteq S \times S$ is a *bisimulation* if for each $(s,s') \in R$ and each $a \in \mathcal{A}ct$, we have: (1) if $s \xrightarrow{a} t$, then there is some $s' \xrightarrow{a} t'$ with $(t,t') \in R$ and, conversely, (2) if $s' \xrightarrow{a} t'$, then there is some $s \xrightarrow{a} t$ with $(t,t') \in R$. We write $s \sim t$ if there is some bisimulation R with $(s,t) \in R$.

A (real-time and non-erasing) *transducer on* Σ *and* Υ is a tuple $T = (Q, q_0, \Sigma, \Upsilon, \delta)$, where Q is a finite set of *states*, $q_0 \in Q$ is an *initial state*, Σ and Υ are finite *alphabets*, and $\delta : Q \times \Sigma \to Q \times \Upsilon^+$ is a transition function with output. We say that T is *letter-to-letter* if $\delta(q,a) \in Q \times \Upsilon$ for each $q \in Q$ and each $a \in \Sigma$. We inductively extend δ to the function $\delta^* : Q \times \Sigma^* \to Q \times \Upsilon^*$ as follows: for each $w \in \Sigma^*$ and $a \in \Sigma$ we set $\delta^*(q,\varepsilon) \stackrel{\text{def}}{=} (q,\varepsilon)$ and $\delta^*(q,aw) \stackrel{\text{def}}{=} (q'',uv)$ if $\delta(q,a) = (q',u)$ and $\delta^*(q',w) = (q'',v)$. We define the *transduction* $f_T : \Sigma^* \to \Upsilon^*$ of T as $f_T(w) \stackrel{\text{def}}{=} v$, whenever $\delta^*(q_0,w) = (q,v)$ for some $q \in Q$. A transduction $f_T : \Sigma^* \to \Upsilon^*$ is said to be letter-to-letter if T is. We define the *size of* T as $|T| \stackrel{\text{def}}{=} |Q| + |\Sigma| + |\Upsilon| + \sum\{|w| : q \in Q, a \in \Sigma, \delta(q,a) = (q',w)\}$.

A *pushdown automaton (PDA)* is a tuple $\mathcal{P} = (Q, \Gamma, \mathcal{A}ct, \hookrightarrow)$, where Q is a finite set of *control states*, Γ is a finite set of *stack symbols*, $\mathcal{A}ct$ is a finite set of *actions*, and $\hookrightarrow \subseteq (Q \times \{\varepsilon\} \times \mathcal{A}ct \times Q \times \{\varepsilon\}) \cup (Q \times \{\varepsilon\} \times \mathcal{A}ct \times Q \times \Gamma) \cup (Q \times \Gamma \times \mathcal{A}ct \times Q \times \{\varepsilon\})$ is a finite set of *internal rules*, *push rules*, and *pop rules*, respectively. The *size* of $\mathcal{P}$ is defined as $|\mathcal{P}| \stackrel{\text{def}}{=} |Q| + |\Gamma| + |\mathcal{A}ct| + |\hookrightarrow|$. We write $qv \overset{a}{\hookrightarrow} q'w$ to mean $(q,v,a,q',w) \in \hookrightarrow$. Every PDA $\mathcal{P}$ induces an LTS $\mathcal{S}(\mathcal{P}) \stackrel{\text{def}}{=} (Q \times \Gamma^*, \mathcal{A}ct, \{\xrightarrow{a} \mid a \in \mathcal{A}ct\})$, where $\xrightarrow{a} \stackrel{\text{def}}{=} \bigcup_{x \in \Gamma^*} \{(qvx, q'wx) \mid qv \overset{a}{\hookrightarrow} q'w\}$ for each $a \in \mathcal{A}ct$. We will abbreviate each configuration (q,w) in $\mathcal{S}(\mathcal{P})$ by qw; in particular the configuration (q,ε) will be denoted by just q.

Given a PDA $\mathcal{P} = (Q, \Gamma, \mathcal{A}ct, \hookrightarrow)$, $q_1, q_2 \in Q$ and $w_1, w_2 \in \Gamma^*$ the *PDA bisimilarity problem* asks whether $q_1 w_1 \sim q_2 w_2$ holds in $\mathcal{S}(\mathcal{P})$. In this paper we prove the following theorem:

Theorem 1. *PDA bisimilarity is nonelementary.*

III. Proof Overview

For a start, let us recall that bisimilarity has a very natural game-theoretic account. Given two labelled transition systems, one can consider a *bisimulation game* involving two players, traditionally called *Attacker* and *Defender* respectively. They play rounds, in which Attacker fires a transition from one of the systems and Defender has to follow with an identically labelled transition from the other system. In the first round, the chosen transitions must lead from the states to be tested for bisimilarity, while, in each subsequent round, they must start at the states reached after the preceding round. Defender loses if she cannot find a matching transition. In this framework, bisimilarity corresponds to the existence of a winning strategy for Defender.

The game-theoretic reading suggests an intuitive way of reducing halting problems for Turing machines to bisimulation problems, based on constructing bisimulation games that satisfy the following condition: a given Turing machine halts on an input string if and only if Defender has a winning strategy. Such games can be viewed as a competition between the players, in which Defender is given an opportunity to exhibit an accepting run and Attacker is equipped with mechanisms to challenge (and verify) the correctness of Defender's construction. The effect of constructing a run by Defender is achieved by allowing Defender to make choices during the game. As the process of playing a bisimulation game naturally favours Attacker as the decision maker, it is not actually clear that the game can be used to express Defender's choice. Nevertheless, it turns out that thanks to the forcing technique of [10], it is possible to construct transition systems in which Defender effectively ends up making choices.

When proving hardness of bisimilarity for classes of computational models, such as pushdown automata, the positions of bisimulation games discussed above must correspond to configurations of the machines. In particular, this means that during the game, players can be thought of as having access to the associated computational resources. For example, in our case, Defender will make moves that store his proposed accepting run on the stack. Additionally, the game can also store some information in the control state of the pushdown system, but since we are interested in finding polynomial-time reductions, these have to be of polynomial size.

Next we give more intuition for our argument by discussing how PSPACE computations can be modelled through bisimulation games, following the argument of Kučera and Mayr [12] (their argument is for EXPTIME, which is equal to alternating PSPACE, but we omit alternation from the discussion, because alternating computation will not be used in our main argument). Let us consider a PSPACE machine $\mathcal{M}$ and an input word. We can code the tape configuration of such a machine by a stack of polynomial size, and we will thus naturally consider a reduction that produces a pair of PDAs – in fact, they are the same PDA but with a different initial state – whose stack configurations at any point represent alleged sequences of configurations of $\mathcal{M}$ with separators (older configurations occur deeper in the stack). The PDA will have moves that can push new tape symbols of the machine $\mathcal{M}$ on the stacks of each configuration, and we can rely on Defender's forcing to delegate the choice of such moves to Defender. The control state can be used to make sure that tapes are the correct size, because each configuration is of polynomial size and we can afford to create polynomially many control states as part of a polynomial-time reduction.

In order to check that Defender's choices amount to a computation history, the PDA is able to move into a "verification mode": at this point, suppose the top of the stacks correspond to a cell having position i at time $t+1$; the top stack symbol σ is saved in the control state, the stack is popped until the top element corresponds to cell position i at time t, and then the symbol appearing is compared to σ: if the symbol corresponds to what the transition relation of the machine says it should be, the machines behave in a bisimilar manner, and otherwise they do not. Note that in order to support popping from position i at time $t+1$ to position i at time t, a counter will be required. Because in this case only polynomially many steps are needed, control state space of the PDA can be used for that purpose.

What breaks down in this argument when we try to move to more powerful machines – e.g. EXPSPACE machines? Firstly, tape configurations are now of size 2^n, and hence we can no longer use the control state to verify that the tape configurations are even of the right size. Secondly, the verification of a valid transition can no longer be achieved by having the machines simply pop their stacks in synch with one another – they would not know when they have reached the corresponding cell position at the previous tape configuration.

We deal with the first difficulty by adding *counters* to every cell in the stack content; thus the code of a tape configuration will consist of a sequence of n address bits followed by a tape content. We can use these address bits to know that the end of a tape configuration has occurred, and thus restrict the machines to have separators between configurations. The fact that the addresses really do represent counters moving up sequentially will need to be verified, but for EXPSPACE this can be done through popping and control states.

The solution to the second difficulty is to perform verification of transitions in a very different way from the PSPACE case. Verification will be carried out only when the machines reach the boundary of a tape configuration. At this point, the machines will first *go out of synch* by one tape configuration – with one machine popping the stack down to the next configuration marker while the other keeps its stack intact. (Technically, this will be achieved as follows: first, both machines push, in synch, a configuration; then, both machines pop stack symbols, but one of them in half speed, so that one machine obtains the previous stack, whereas the other one effectively pops a configuration.) After this the machines will pop stack symbols, but with one machine emitting symbols corresponding exactly to what it sees, while the other machine emits symbols corresponding to the configuration obtained by applying the transition function to the symbols it sees. Thus, in the second phase, the machines will emit the same symbols *exactly when the two successive configurations obey the transition function*.

The above idea can be extended from EXPSPACE to k-EXPSPACE inductively. Indices that count up to a given tower of exponentials will now precede each tape symbol. The indices used to capture smaller towers will be embedded into those for larger ones. For instance, assuming that $c_0, \cdots, c_{2^n-1}$ are the binary strings representing the numbers $0, \cdots, 2^n-1$ respectively, the sequence $c_0\sigma_0 \cdots c_{2^n-1}\sigma_{2^n-1}$, where σ_i's are bits, will be used to represent natural numbers from the interval $[0, 2^{2^n}-1]$. The indexing can be used to enforce that the stack consists of tape configurations of the correct size. The verification that counting indices are incremented correctly as well as the verification that the tape configurations obey the transition function, can be done using the technique of going out of synch and reading distinct symbols.

Altogether, we get k-EXPSPACE-hardness for all k, and thus a nonelementary lower bound.

IV. Notation and Techniques

In order to prove Theorem 1 we are going to show that PDA bisimilarity is k-EXPSPACE-hard for each fixed $k \geq 1$. To that end, given a k-EXPSPACE Turing machine $\mathcal{M}$ with an input string, we will construct (in polynomial time) a PDA $\mathcal{P}$ such that $\mathcal{M}$ accepts the input if and only if certain two configurations of $\mathcal{P}$ are bisimilar. We will rely on a number of techniques and notational conventions introduced below.

In this section and the next we will progressively reveal more and more technical details about the special structure of $\mathcal{P} = (Q, \Gamma, \mathcal{A}ct, \hookrightarrow)$. For a start, we shall assume a certain partitioning of Q.

- Suppose $\mathcal{B}$ is a finite set. Let us make two disjoint copies of $\mathcal{B}$, called ${}_\bullet\mathcal{B}$ and $\mathcal{B}_\bullet$ respectively. Given $s \in \mathcal{B}$, we shall refer to the corresponding elements in ${}_\bullet\mathcal{B}$ and $\mathcal{B}_\bullet$ by ${}_\bullet s$ and $s_\bullet$ respectively. Let us write Q_{main} for ${}_\bullet\mathcal{B} \uplus \mathcal{B}_\bullet$. We will call Q_{main} the set of *main states*.
- Suppose Q_{impl} is another finite set. Its elements will be called *implicit states*.

We are going to assume that Q is partitioned as follows:

$$Q = Q_{main} \uplus Q_{impl}.$$

The role of the partition will become clear in a moment.

In the interest of succinctness and readability we will define $\hookrightarrow$ via *macro rules*, which compactly represent collections of PDA transition rules with a certain role. They take one of the following five shapes:

$$p\sigma \xhookrightarrow{a_1 \cdots a_\ell} q \qquad pL \xhookrightarrow{T} q \qquad s \circ\!\!\rightarrow t\sigma_1 \cdots \sigma_\ell$$
$$s \circ\!\!\xrightarrow{Att} \{t_1 w_1,\ \ldots,\ t_\ell w_\ell\} \qquad s \circ\!\!\xrightarrow{Def} \{t_1 w_1,\ \ldots,\ t_\ell w_\ell\}.$$

The various indices on a macro role (such as $T, a_1, \ldots$) will be explained shortly. For the moment we mention that the implementation of each macro rule will contribute a number of implicit states (that is, elements of Q_{impl}) to the automaton.

Convention. We assume that no implicit state can be used by two different macros. Moreover, if a state occurs on the left-hand-side of one of the rules, it cannot occur on the left-hand-side of any other rule except that if p occurs in the first rule then we allow other rules of the first kind with the same p but different σ.

We explain each of the macro formats next.

A. Single pop with fixed trace

For $p, q \in Q_{main}$, $\sigma \in \Gamma$ and $a_1, \cdots, a_\ell \in \mathcal{A}ct$, we write

$$p\sigma \xhookrightarrow{a_1 \cdots a_\ell} q$$

for the sequence of transitions displayed below

$$p\sigma \xhookrightarrow{a_1} p_1 \xhookrightarrow{a_2} \cdots \xhookrightarrow{a_{\ell-1}} p_{\ell-1} \xhookrightarrow{a_\ell} q,$$

where $p_1, \cdots, p_{\ell-1} \in Q_{impl}$.

B. Transduction of stack content with matching

Our PDA construction will also require the automaton to read certain sequences of action names depending on stack content. This can be conveniently expressed using the language of transducers. In particular, emissions of signals during pop transitions will be important. The next macro will make it easy to specify such activities flexibly.

For $p, q \in Q_{main}$, a regular language $L \subseteq \Gamma^*$ and a transducer T on Γ and $\mathcal{A}ct$ we write

$$pL \xhookrightarrow{T} q$$

to stipulate the sequence of transitions described below. They will make sure that, once $\mathcal{P}$ reaches configuration py for $y \in \Gamma^*$, the shortest prefix w of y with $w \in L$ will be popped, $\#T(w)\#$ will be read (where $\# \in \mathcal{A}ct$ is a special action symbol), and the control state will be changed to q; if y does not have a prefix w with $w \in L$, then y will be popped, and $\#T(y)$ will be read.

The transitions are the result of a product construction between a deterministic finite automaton (DFA) accepting L (e.g. the minimal one) and the transducer T. More precisely, let $A = (Q_A, q_0^A, \Gamma, F_A, \delta_A)$ be the minimal DFA that accepts L, where Q_A is the finite set of states, $q_0^A \in Q_A$ is the initial state, $F_A \subseteq Q_A$ is the set of final states, $\delta_A : Q_A \times \Gamma \to Q_A$ is the transition function. Assume $T = (Q_T, q_0^T, \Gamma, \mathcal{A}ct, \delta_T)$. Then $pL \xhookrightarrow{T} q$ comprises the following rules with the proviso that $Q_A \times Q_T \subseteq Q_{impl}$:

- $p \xhookrightarrow{\#} (q_0^A, q_0^T)$;
- $(q^A, q^T) \xhookrightarrow{\#} q$ for each $q^A \in F_A$ and each $q^T \in Q_T$;
- for each $\sigma \in \Gamma$, each $q^A \in Q_A \setminus F_A$ and each $q^T \in Q_T$, where $\delta_T(q^T, \sigma) = (r^T, w)$, we have the (macro) rule $(q^A, q^T)\sigma \xhookrightarrow{w} (\delta_A(q^A, \sigma), r^T)$.

C. Synchronized pushing

Recall that $Q = ({}_\bullet\mathcal{B} \uplus \mathcal{B}_\bullet) \uplus Q_{impl}$. The next macro will use elements of $\mathcal{B}$ to construct simultaneously transitions involving both ${}_\bullet\mathcal{B}$ and $\mathcal{B}_\bullet$. More precisely, given $s, t \in \mathcal{B}$ and $\sigma_1, \ldots, \sigma_\ell \in \Gamma$ ($\ell \geq 1$), we will write

$$s \circ\!\!\rightarrow t\sigma_1\sigma_2 \cdots \sigma_\ell$$

to state that there are *implicit* states q_i^L, q_i^R ($1 \leq i \leq \ell$) with

$$\begin{array}{lll} {}_\bullet s \xhookrightarrow{a} q_\ell^L \sigma_\ell & q_{j+1}^L \xhookrightarrow{a} q_j^L \sigma_j & q_1^L \xhookrightarrow{a} {}_\bullet t \\ s_\bullet \xhookrightarrow{a} q_\ell^R \sigma_\ell & q_{j+1}^R \xhookrightarrow{a} q_j^R \sigma_j & q_1^R \xhookrightarrow{a} t_\bullet \end{array}$$

where $1 \leq j < \ell$.

Given $s \in \mathcal{B}$ and $w \in \Gamma^*$, let us write $\sim sw$ as a shorthand for ${}_\bullet sw \sim s_\bullet w$. In presence of the macro $s \circ\!\!\rightarrow t\sigma_1\sigma_2\cdots\sigma_\ell$ we have by our rules convention

$$\text{for all } x \in \Gamma^* : \quad \sim sx \iff \sim t\sigma_1\sigma_2\cdots\sigma_\ell x. \quad (1)$$

D. Forcing

Recall that the PDA $\mathcal{P}$ to be constructed in our argument is supposed to enable a bisimulation game on $\mathcal{S}(\mathcal{P})$, which will correspond to a step-by-step construction of an accepting run. The run will be represented as a sequence of configurations. During the game Defender will have the power to decide what to add to the sequence, whereas Attacker will be able to initiate correctness checks that can detect mistakes in Defender's choices. To construct parts of $\mathcal{P}$ that will allow for such choices at suitable stages, we are going to use two blueprint designs for labeled transition systems: *Or-widgets* (Defender's forcing) and *And-widgets* (Attacker's forcing), shown in Figure 1. They express respectively logical disjunction and logical conjunction with respect to bisimulation.

Lemma 2 ([10]). *Consider the states and transitions of a widget from Figure 1, viewed as part of a larger LTS in which there are no other outgoing transitions from ${}_\circ s, s_\circ$ than those shown in the Figure and no other transitions involving u_i ($1 \leq i \leq 3$). Then we have:*

(a) *Or-widget: ${}_\circ s \sim s_\circ$ if and only if ${}_\circ t \sim t_\circ$ or ${}_\circ t' \sim t'_\circ$;*
(b) *And-widget: ${}_\circ s \sim s_\circ$ if and only if ${}_\circ t \sim t_\circ$ and ${}_\circ t' \sim t'_\circ$.*

In terms of the Defender-Attacker game, if the players reach $({}_\circ s, s_\circ)$ in the game, the Or-widget allows Defender to decide if the play should continue in $({}_\circ t_1, t_{1\circ})$ or $({}_\circ t_2, t_{2\circ})$, whereas, in the And-widget, it is Attacker who makes this choice.

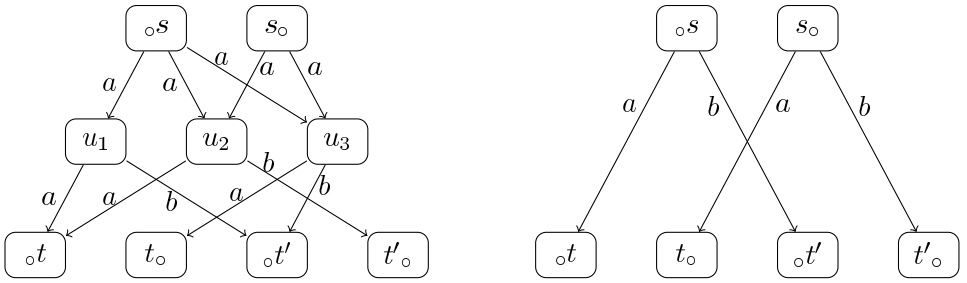
The next macro is based on the Or-widget (Figure 1 (a)). Given, $\sigma_1, \sigma_2 \in \Gamma \cup \{\varepsilon\}$ and $s, t_1, t_2 \in \mathcal{B}$, we write

$$s \xrightarrow{Def} \{t_1\sigma_1,\ t_2\sigma_2\}$$

to denote a sequence of transitions closely following the widget. It will allow Defender to choose between two (possibly) pushing transitions. More specifically, we want to add the following transitions on the understanding that $u_1, u_2, u_3 \in Q_{impl}$:

$$\begin{array}{lll} {}_\bullet s \xhookrightarrow{a} u_1 & {}_\bullet s \xhookrightarrow{a} u_2 & {}_\bullet s \xhookrightarrow{a} u_3 \\ & s_\bullet \xhookrightarrow{a} u_2 & s_\bullet \xhookrightarrow{a} u_3 \\ u_1 \xhookrightarrow{a} {}_\bullet t_1\sigma_1 & u_2 \xhookrightarrow{a} {}_\bullet t_1\sigma_1 & u_3 \xhookrightarrow{a} t_{1\bullet}\sigma_1 \\ u_1 \xhookrightarrow{b} {}_\bullet t_2\sigma_2 & u_2 \xhookrightarrow{b} t_{2\bullet}\sigma_2 & u_3 \xhookrightarrow{b} {}_\bullet t_2\sigma_2. \end{array}$$

Note that Lemma 2 concerns labeled transition systems, whereas the definitions above refer to PDAs. Consequently, in order to induce the Or-widget in the LTS $\mathcal{S}(\mathcal{P})$ for ${}_\circ s = {}_\bullet sx$ and $s_\circ = s_\bullet y$, where $x, y \in \Gamma^*$, we will assume that $x = y$ (due to the need to reach the same configuration from ${}_\bullet sx$ and $s_\bullet y$ in a single transition). Intuitively, when the state of the Defender-Attacker game is $({}_\bullet sx, s_\bullet x)$ for $x \in \Gamma^*$, Defender can choose whether the game will proceed to $({}_\bullet t_1\sigma_1 x, t_{1\bullet}\sigma_1 x)$ or $({}_\bullet t_2\sigma_2 x, t_{2\bullet}\sigma_2 x)$. In other words, we have $\sim sx$ if and only



(a) Or-widget (b) And-widget

Fig. 1. Logical widgets

if $\sim t_1\sigma_1 x$ or $\sim t_2\sigma_2 x$. We generalize this notation to finite sets: for $s, t_1, \ldots, t_\ell \in Q$ and $\{w_1, \ldots, w_\ell\} \subseteq \Gamma^*$ we shall write

$$s \overset{Def}{\circ\!\!\longrightarrow} \{t_1 w_1,\ \ldots,\ t_\ell w_\ell\}$$

to denote that a sequence of Or-widgets is used to achieve

$$\text{for all } x \in \Gamma^* : \quad \sim sx \quad \Longleftrightarrow \quad \bigvee_{i=1}^{\ell} \sim t_i w_i x\,. \tag{2}$$

Similarly we write

$$s \overset{Att}{\circ\!\!\longrightarrow} \{t_1 w_1,\ \ldots,\ t_\ell w_\ell\}$$

to denote that And-widgets (Attacker's forcing, Figure 1 (b)) are used to achieve

$$\text{for all } x \in \Gamma^* : \quad \sim sx \quad \Longleftrightarrow \quad \bigwedge_{i=1}^{\ell} \sim t_i w_i x\,. \tag{3}$$

Note that the shape of the And-widget does not contain any state synchronizations. Consequently, it does not matter whether the stack content is the same at ${}_\bullet s$ and $s_\bullet$. However, we will not need this level of generality in our argument.

V. Counters

To represent configurations of a k-EXPSPACE Turing machine we shall use binary strings whose length is equal to the tower of exponentials of height k. For technical reasons discussed in Section III, rather than working with raw configurations we shall precede each binary symbol with a number that indicates its position in the string. The following definitions introduce the relevant technical notions.

For each $\ell, n \geq 0$ we define $\mathsf{Tower}(\ell, n)$ inductively as $\mathsf{Tower}(0, n) \overset{\text{def}}{=} n$ and $\mathsf{Tower}(\ell+1, n) \overset{\text{def}}{=} 2^{\mathsf{Tower}(\ell, n)}$.

Definition 3. *Let $\Omega_\ell \overset{def}{=} \{0_\ell, 1_\ell\}$ be alphabets consisting of letters with numerical value: $val(0_\ell) = 0$ and $val(1_\ell) = 1$.*

- *A $(0, n)$-counter is a word from Ω_0^n. Its* value *is defined by $val(\sigma_0 \cdots \sigma_{n-1}) \overset{def}{=} \sum_{i=0}^{n-1} 2^i \cdot val(\sigma_i)$.*
- *An $(\ell+1, n)$-counter is a word $c = c_0\sigma_0 c_1\sigma_1 \cdots c_m\sigma_m$ such that $m = \mathsf{Tower}(\ell+1, n) - 1$ and, for all $i \in [0, m]$, c_i is an (ℓ, n)-counter with $val(c_i) = i$, and $\sigma_i \in \Omega_{\ell+1}$. We define $val(c) \overset{def}{=} \sum_{i=0}^{m} 2^i \cdot val(\sigma_i)$.*

When n is clear from the context, we may speak of an ℓ-*counter* to mean an (ℓ, n)-counter. Observe that the length of each (ℓ, n)-counter is uniquely determined by ℓ and n. Note also that the set of values taken by (ℓ, n)-counters is equal to $[0, \mathsf{Tower}(\ell+1, n) - 1]$. In the two extreme cases ($val(c) = 0$ or $val(c) = \mathsf{Tower}(\ell+1, n) - 1$) we shall call the (ℓ, n)-counters *zeros* and *ones* (or, when n is clear from context, "the ones ℓ-counter"), respectively.

In the following we write $\Omega_{\leq \ell}$ for $\bigcup_{i=0}^{\ell} \Omega_i$. Thus, an (ℓ, n)-counter matches the regular expression $(\Omega_{\leq \ell-1}^* \cdot \Omega_\ell)^*$ for all $\ell \geq 1$.

Binary strings of length $\mathsf{Tower}(k, n)$ in which each bit is preceded by a number indicating its position are thus naturally represented as k-counters. Consequently, k-counters will be taken to represent configurations of k-EXPSPACE Turing machines. Because during our construction we will be interested in storing configurations on the stack, from now on we assume that our stack alphabet Γ includes $\Omega_{\leq k}$.

Next we present a construction that enables one to compare two consecutive counters pushed on the stack via bisimulation. Its key idea is the use of transducers to communicate information about stack content as well as to desynchronize the two stacks involved in playing the bisimulation game. It will also illustrate the $pL \overset{T}{\hookrightarrow} q$ macro at work.

Given an alphabet Ω and a word $w \in \mathcal{A}ct^*$, we write $\Omega \mapsto w$ to refer to a transducer T_w that outputs w on reading each letter of the input string from Ω^*.

Lemma 4. *Let T_1, T_2 be letter-to-letter transducers on $\Omega_{\leq \ell+1}$ and $\mathcal{A}ct$ for some $\ell \geq 0$. Suppose the definition of $\overline{\mathcal{P}} = (Q, \Gamma, \mathcal{A}ct, \hookrightarrow)$ involves, possibly among others, the following macros.*

$$\bullet p\ \Omega_{\leq \ell}^* \cdot \Omega_{\ell+1} \cdot \Omega_{\leq \ell}^* \cdot \Omega_{\ell+1} \xhookrightarrow{\Gamma \mapsto a} \bullet q \qquad \textit{pop two } \ell\textit{-counters and two } \Omega_{\ell+1}\textit{-symbols}$$

$$p_\bullet\ \Omega_{\leq \ell}^* \cdot \Omega_{\ell+1} \xhookrightarrow{\Gamma \mapsto aa} q_\bullet \qquad \textit{pop one } \ell\textit{-counter and one } \Omega_{\ell+1}\textit{-symbol}$$

$$\bullet q\ \left(\Omega_{\leq \ell-1}^* \cdot \Omega_\ell\right)^* \cdot \Omega_{\ell+1} \xhookrightarrow{T_1} \bullet r \qquad \textit{apply } T_1$$

$$q_\bullet\ \left(\Omega_{\leq \ell-1}^* \cdot \Omega_\ell\right)^* \cdot \Omega_{\ell+1} \xhookrightarrow{T_2} r_\bullet \qquad \textit{apply } T_2$$

Let $\sigma_1, \sigma_2, \sigma_3 \in \Omega_{\ell+1}$, and let w_1, w_2, w_3 be ℓ-counters.

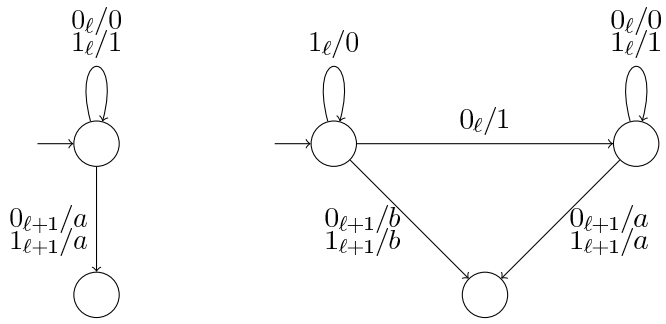


Fig. 2. Transducers T_ℓ^{+0} and T_ℓ^{+1}

Assume $x \in \Gamma^*$ *such that* $\bullet rx \sim r\bullet w_1\sigma_1 x$. *Then*

$$\sim p\, w_3\sigma_3 w_2\sigma_2 w_1\sigma_1 x$$

if and only if $T_1(w_1\sigma_1) = T_2(w_2\sigma_2)$.

Proof:

$$\begin{array}{rl} & \bullet p w_3\sigma_3 w_2\sigma_2 w_1\sigma_1 x \sim p\bullet w_3\sigma_3 w_2\sigma_2 w_1\sigma_1 x \\ \iff & \bullet q w_1\sigma_1 x \sim q\bullet w_2\sigma_2 w_1\sigma_1 x \qquad \text{(first two rules)} \\ \iff & T_1(w_1\sigma_1) = T_2(w_2\sigma_2) \qquad \text{(last two rules).} \end{array}$$

■

Given two transductions $f_1 : \Sigma_1^* \to \Upsilon^*$ and $f_2 : \Sigma_2^* \to \Upsilon^*$ with $\Sigma_1 \cap \Sigma_2 = \emptyset$, their *shuffle* is the transduction $f_1 \| f_2 : (\Sigma_1 \cup \Sigma_2)^* \to \Upsilon^*$ defined inductively as follows: $f_1\|f_2(\varepsilon) \stackrel{\text{def}}{=} \varepsilon$ and $f_1\|f_2(aw) \stackrel{\text{def}}{=} f_i(a) \cdot (f_1\|f_2(w))$ for $a \in \Sigma_i$, $w \in (\Sigma_1 \cup \Sigma_2)^*$ and $i \in \{1,2\}$. We note that from two given transducers T_1, T_2 with transductions $f_{T_1} : \Sigma_1^* \to \Upsilon^*$ and $f_{T_2} : \Sigma_2^* \to \Upsilon^*$, one can compute a transducer T such that $f_T = f_{T_1} \| f_{T_2}$ in time $O(|T_1| \cdot |T_2|)$.

In what follows we rely on two specific transducers T_ℓ^{+0}, T_ℓ^{+1} on $\Omega_\ell \cup \Omega_{\ell+1}$ and $\{0,1,a,b\}$ depicted in Figure 2; they are formally not transducers since some outgoing transitions are missing – but the missing transitions will not be relevant later and are therefore omitted. They interpret the input word over Ω_ℓ as a number in binary, with the least significant bit read first. Transducer T_ℓ^{+0} copies the number and outputs a upon reading an $\Omega_{\ell+1}$-symbol. Transducer T_ℓ^{+1} attempts to increase the number by 1 and outputs a upon reading an $\Omega_{\ell+1}$-symbol, but it will output b if the input number consisted only of 1s. If w_1, w_2 are ℓ-counters and $\sigma_1, \sigma_2 \in \Omega_{\ell+1}$, then we have

$$(T_\ell^{+0} \,\|\, \Omega_{\le \ell-1} \mapsto a)(w_1\sigma_1) = (T_\ell^{+1} \,\|\, \Omega_{\le \ell-1} \mapsto a)(w_2\sigma_2)$$

if and only if $val(w_1) = val(w_2) + 1$. Using the two transducers one can verify through bisimilarity whether two counters placed suitably on the stack have consecutive values.

Lemma 5. *Suppose* $\langle \mathsf{stop}_\ell\rangle, \langle \mathsf{testDec}_\ell\rangle, \langle \mathsf{testDec}_\ell^1\rangle \in \mathcal{B}$ *and the definition of* $\mathcal{P} = (Q, \Gamma, \mathcal{A}ct, \hookrightarrow)$ *involves the following macros.*

$$\begin{array}{rl} \bullet\langle \mathsf{testDec}_\ell\rangle\ \Omega_{\le\ell}^* \cdot \Omega_{\ell+1} \cdot \Omega_{\le\ell}^* \cdot \Omega_{\ell+1} & \xhookrightarrow{\Gamma \mapsto a} \bullet\langle \mathsf{testDec}_\ell^1\rangle \\ \langle \mathsf{testDec}_\ell\rangle\bullet\ \Omega_{\le\ell}^* \cdot \Omega_{\ell+1} & \xhookrightarrow{\Gamma \mapsto aa} \langle \mathsf{testDec}_\ell^1\rangle\bullet \\ \bullet\langle \mathsf{testDec}_\ell^1\rangle\ \left(\Omega_{\le\ell-1}^* \cdot \Omega_\ell\right)^* \cdot \Omega_{\ell+1} & \xhookrightarrow{T_\ell^{+0} \,\|\, \Omega_{\le\ell-1} \mapsto a} \bullet\langle \mathsf{stop}_\ell\rangle \\ \langle \mathsf{testDec}_\ell^1\rangle\bullet\ \left(\Omega_{\le\ell-1}^* \cdot \Omega_\ell\right)^* \cdot \Omega_{\ell+1} & \xhookrightarrow{T_\ell^{+1} \,\|\, \Omega_{\le\ell-1} \mapsto a} \langle \mathsf{stop}_\ell\rangle\bullet \end{array}$$

Assume that, for all $x \in \Gamma^*$, $\sigma \in \Omega_{\ell+1}$ *and all* ℓ*-counters* w, *we have* $\bullet\langle \mathsf{stop}_\ell\rangle x \sim \langle \mathsf{stop}_\ell\rangle\bullet w\sigma x$.[1] *Let* $x \in \Gamma^*$, $\sigma_1, \sigma_2, \sigma_3 \in \Omega_{\ell+1}$ *and let* w_1, w_2, w_3 *be* ℓ*-counters. Then* $\sim\langle \mathsf{testDec}_\ell\rangle\, w_3\sigma_3 w_2\sigma_2 w_1\sigma_1 x$ *if and only if* $val(w_1) = val(w_2) + 1$.

Proof: Thanks to $\bullet\langle \mathsf{stop}_\ell\rangle x \sim \langle \mathsf{stop}_\ell\rangle\bullet w\sigma x$ we can apply Lemma 4 to conclude

$$\begin{array}{l} \sim\langle \mathsf{testDec}_\ell\rangle w_3\sigma_3 w_2\sigma_2 w_1\sigma_1 x \\ \iff (T_\ell^{+0} \,\|\, \Omega_{\le\ell-1} \mapsto a)(w_1\sigma_1) = (T_\ell^{+1} \,\|\, \Omega_{\le\ell-1} \mapsto a)(w_2\sigma_2) \\ \iff val(w_1) = val(w_2) + 1. \end{array}$$

■

In the remainder of the paper we will assume a very particular shape of the set $\mathcal{B}$. Suppose $\mathbb{I}$ is a finite set of *instructions*. Then we insist that

$$\mathcal{B} = \{\ \langle\alpha\rangle \quad | \quad \alpha \in \mathbb{I}^*, \quad 1 \le |\alpha| \le k+2\}.$$

It may be helpful to think of $\langle\alpha\rangle$ as a bounded sequence of instructions that are manipulated separately from the unbounded pushdown stack. In what follows we shall use β to range over $\alpha \in \mathbb{I}^*$ such that $1 \le |\alpha| \le k+1$.

Our next result shows how to define macro rules for managing counters on the stack.

Lemma 6. *Let* $\mathbb{I}$ *contain* stop_ℓ, $\mathsf{testDec}_\ell$, ones_ℓ, decOk_ℓ, zeros_ℓ, zeros_ℓ^1, dec_ℓ, dec_ℓ^1, $\mathsf{dec}_0^{(i)}$ *for* $1 \le i < n$ *and* $\ell \in [0,k]$. *Suppose the definition of* $\mathcal{P} = (Q, \Gamma, \mathcal{A}ct, \hookrightarrow)$ *involves the macros given in Figure 3. Let* $\ell \in [0,k]$, *and* $x \in \Gamma^*$, *and* $\sigma, \tau \in \Omega_{\ell+1}$, *and* v, w *be* ℓ*-counters.*

(a) $\sim\langle \mathsf{ones}_\ell\, \beta\rangle x$ *iff* $\sim\langle\beta\rangle wx$, *where* w *is the ones* ℓ*-counter.*
(b) $\sim\langle \mathsf{decOk}_\ell\, \beta\rangle v\sigma w\tau x$ *iff* $val(v) + 1 = val(w)$ *and* $\sim\langle\beta\rangle v\sigma w\tau x$.
(c) $\sim\langle \mathsf{zeros}_\ell\, \beta\rangle\sigma w\tau x$ *iff* $val(w) = 1$ *and* $\sim\langle\beta\rangle v\sigma w\tau x$, *where* v *is the* ℓ*-counter with* $val(v) = 0$.
(d) $\sim\langle \mathsf{dec}_\ell\, \beta\rangle\sigma w\tau x$ *iff* $val(w) \neq 0$ *and* $\sim\langle\beta\rangle v\sigma w\tau x$, *where* v *is the* ℓ*-counter with* $val(v) + 1 = val(w)$.

VI. Reductions

We prove Theorem 1 by showing that PDA bisimilarity is k-EXPSPACE-hard for all $k \ge 1$.

To that end we introduce a somewhat abstract description of accepting runs, based on transducers. It will be convenient to rely on it when representing a k-EXPSPACE computation through PDA configurations.

[1] Note that this is easily achieved by including no outgoing rules for $\bullet\langle \mathsf{stop}_\ell\rangle$ and $\langle \mathsf{stop}_\ell\rangle\bullet$.

Rules for case $\ell = 0$.

$$\langle \mathsf{ones}_0\, \beta\rangle \hookrightarrow \langle \beta\rangle 1_0^n$$ push a ones 0-counter

$$\langle \mathsf{decOk}_0\, \beta\rangle \overset{Att}{\longrightarrow} \{\, \langle \beta\rangle,\, \langle \mathsf{testDec}_0\rangle\, 0_0^n 0_1 \,\}$$ assume that values of top two 0-counters differ by 1 OR challenge that claim by invoking $\mathsf{testDec}_0$

$$\langle \mathsf{zeros}_0\, \beta\rangle \hookrightarrow \langle \mathsf{decOk}_0 \beta\rangle\, 1_0^n$$ push a zeros 0-counter and check if it is over a 0-counter with value 1

$$\langle \mathsf{dec}_0\, \beta\rangle \overset{Def}{\longrightarrow} \{\langle \mathsf{dec}_0^{(1)}\, \beta\rangle\, 0_0, \langle \mathsf{dec}_0^{(1)}\, \beta\rangle 1_0\}$$ push the first bit of the decremented 0-counter

$$\forall 1 \le i < n : \langle \mathsf{dec}_0^{(i)}\, \beta\rangle \overset{Def}{\longrightarrow} \{\, \langle \mathsf{dec}_0^{(i+1)}\, \beta\rangle\, 0_0,\, \langle \mathsf{dec}_0^{(i+1)}\, \beta\rangle\, 1_0\}$$ push the $(i+1)^{\text{st}}$ bit of the decremented 0-counter

$$\langle \mathsf{dec}_0^{(n)}\, \beta\rangle \hookrightarrow \langle \mathsf{decOk}_0\, \beta\rangle$$ verify if the 0-counter has been correctly decremented

Rules for $1 \le \ell \le k$.

$$\langle \mathsf{ones}_\ell\, \beta\rangle \hookrightarrow \langle \mathsf{ones}_{\ell-1}\, \mathsf{ones}_\ell^1\, \beta\rangle\, 1_\ell$$ push 1_ℓ and a ones $(\ell-1)$-counter

$$\langle \mathsf{ones}_\ell^1 \beta\rangle \overset{Def}{\longrightarrow} \{\, \langle \mathsf{dec}_{\ell-1}\, \mathsf{ones}_\ell^1\, \beta\rangle\, 1_\ell,\, \langle \mathsf{zeros}_{\ell-1}\, \beta\rangle\, 1_\ell \,\}$$ push 1_ℓ and a decremented $(\ell-1)$-counter OR push 1_ℓ and a zeros $(\ell-1)$-counter

$$\langle \mathsf{decOk}_\ell\, \beta\rangle \overset{Att}{\longrightarrow} \{\, \langle \beta\rangle,\, \langle \mathsf{ones}_\ell\, \mathsf{testDec}_\ell\rangle\, 0_{\ell+1} \,\}$$ assume that values of top two ℓ-counters differ by 1 OR challenge that claim by invoking $\mathsf{testDec}_\ell$

$$\langle \mathsf{zeros}_\ell\, \beta\rangle \hookrightarrow \langle \mathsf{ones}_{\ell-1}\, \mathsf{zeros}_\ell^1\, \beta\rangle\, 0_\ell$$ push 0_ℓ and a ones $(\ell-1)$-counter

$$\langle \mathsf{zeros}_\ell^1\, \beta\rangle \overset{Def}{\longrightarrow} \{\, \langle \mathsf{dec}_{\ell-1}\, \mathsf{zeros}_\ell^1\, \beta\rangle\, 0_\ell, \langle \mathsf{zeros}_{\ell-1}\, \mathsf{decOk}_\ell\, \beta\rangle\, 0_\ell \,\}$$ push 0_ℓ and decremented $(\ell-1)$-counter OR push 0_ℓ and zeros $(\ell-1)$-counter

$$\langle \mathsf{dec}_\ell\, \beta\rangle \overset{Def}{\longrightarrow} \{\, \langle \mathsf{ones}_{\ell-1}\, \mathsf{dec}_\ell^1\, \beta\rangle\, \sigma \mid \sigma \in \Omega_\ell\}$$ push from Ω_ℓ and a ones $(\ell-1)$-counter

$$\langle \mathsf{dec}_\ell^1\, \beta\rangle \overset{Def}{\longrightarrow} \{\, \langle \mathsf{dec}_{\ell-1}\, \mathsf{dec}_\ell^1\, \beta\rangle\sigma,\, \langle \mathsf{zeros}_{\ell-1}\, \mathsf{decOk}_\ell\, \beta\rangle\, \sigma \mid \sigma \in \Omega_\ell \,\}$$ push from Ω_ℓ and a decremented $(\ell-1)$-counter OR push from Ω_ℓ and a zeros $(\ell-1)$-counter

Fig. 3. Macro rules from Lemma 6.

Suppose $z_0, \ldots, z_t$ are binary sequences representing configurations of an accepting run of a deterministic Turing machine, i.e. z_0, z_t correspond to initial and accepting configurations respectively and, for any $0 \le i < t$, z_{i+1} represents the successor configuration with respect to that corresponding to z_i. If we imagine that T_1 is a transducer capable of generating successor configurations and T_2 is a copy-cat transducer, then the relationship between z_i and z_{i+1} boils down to the requirement that $T_1(z_i) = T_2(z_{i+1})$. This motivates the definition below, where we allow an arbitrary T_2, not just a copy-cat. This will permit T_2 to be a copy-cat but with some delay in outputting configurations – this is necessary for computing successor configurations.

Definition 7. *Let T_1, T_2 be letter-to-letter transducers on $\{0,1\}$ and Υ. Let h be in $\mathbb{N}$. We say that the pair (T_1, T_2) is* h-terminating *if there exist $t \in \mathbb{N}$ and $z_0, \ldots, z_t \in \{0,1\}^h$ such that*

- $z_0 = 1^h$,
- *for all $i \in [0, t-1]$, z_{i+1} is the only $z' \in \{0,1\}^h$ with $T_1(z_i) = T_2(z')$,*
- *there is no $z' \in \{0,1\}^h$ such that $T_1(z_t) = T_2(z')$ or $T_1(0^h) = T_2(z')$.*

If (T_1, T_2) is h-terminating we write $\mathsf{last}(h, T_1, T_2) \overset{def}{=} z_t$.

Fix for the rest of the paper $k \ge 1$. Given that h-terminating pairs of transducers were introduced as a generalization of accepting computation histories, the following result does not come as a surprise.

Proposition 8. *Consider the decision problem* TRANSREACH(k) *defined below.*

> *Given (n, T_1, T_2), where $n \in \mathbb{N}$ is presented in unary and (T_1, T_2) is a $\mathsf{Tower}(k,n)$-terminating pair of transducers on $\{0,1\}$ and Υ, decide whether $\mathsf{last}(\mathsf{Tower}(k,n), T_1, T_2) = 0^{\mathsf{Tower}(k,n)}$.*

TRANSREACH(k) *is k-*EXPSPACE*-complete with respect to polynomial-time many-one reductions.*

The main result will now follow immediately from reducing TRANSREACH(k) to bisimilarity:

Lemma 9. TRANSREACH(k) *is polynomial-time reducible to PDA bisimilarity.*

Proof: Let us fix an instance (n, T_1, T_2) of TRANSREACH(k). Using notation introduced in Sections IV and V, we construct $\mathcal{P} = (Q, \Gamma, \mathcal{A}ct, \hookrightarrow)$ next.

The PDA $\mathcal{P}$ will be able to push a code of a sequence of words onto the stack, where each word ρ_i is encoded as a k-counter, say w_i, in the obvious way: $\rho_i = \eta(w_i)$ where

$$\langle \mathsf{start}\rangle \hookrightarrow \langle \mathsf{ones}_k\,\mathsf{fin}\rangle\,\$ \qquad \text{push } \$ \text{ and encoded } z_0$$

$$\langle \mathsf{fin}\rangle \xrightarrow{Def} \{\langle \mathsf{testFin}\rangle,\ \langle \mathsf{next}\rangle\$\} \qquad \text{test equality with } 0^{\mathsf{Tower}(k,n)} \text{ OR go on to the next } z_i$$

$$\bullet\langle \mathsf{testFin}\rangle\,(\Omega^*_{\leq k-1}\Omega_k)^*\,\$ \xrightarrow{\{1_k\}\mapsto b\,\|\,(\Gamma\setminus\{1_k\})\mapsto a} \bullet\langle \mathsf{popAll}\rangle \qquad \text{rule à la Lemma 4 to test equality with } 0^{\mathsf{Tower}(k,n)}$$

$$\langle \mathsf{testFin}\rangle_\bullet\,(\Omega^*_{\leq k-1}\Omega_k)^*\,\$ \xrightarrow{\Gamma\mapsto a} \langle \mathsf{popAll}\rangle_\bullet \qquad \text{rule à la Lemma 4 to test equality with } 0^{\mathsf{Tower}(k,n)}$$

$$\bullet\langle \mathsf{popAll}\rangle\,\omega \overset{a}{\hookrightarrow} \bullet\langle \mathsf{popAll}\rangle \qquad \text{erase stack content } (\omega\in\Gamma)$$

$$\langle \mathsf{popAll}\rangle_\bullet\,\omega \overset{a}{\hookrightarrow} \langle \mathsf{popAll}\rangle_\bullet \qquad \text{erase stack content } (\omega\in\Gamma)$$

$$\langle \mathsf{next}\rangle \xrightarrow{Def} \left\{\langle \mathsf{ones}_{k-1}\,\mathsf{next}^1\rangle\,\sigma \mid \sigma\in\Omega_k\right\} \qquad \text{construct the next } z_i$$

$$\langle \mathsf{next}^1\rangle \xrightarrow{Def} \left\{\langle \mathsf{dec}_{k-1}\,\mathsf{next}^1\rangle\,\sigma,\ \langle \mathsf{zeros}_{k-1}\,\mathsf{tran}\rangle\,\sigma \mid \sigma\in\Omega_k\right\}$$

$$\langle \mathsf{tran}\rangle \xrightarrow{Att} \{\langle \mathsf{ones}_k\,\mathsf{testTran}\rangle\,\$,\ \langle \mathsf{fin}\rangle\} \qquad \text{test whether new } z_i \text{ is correct OR continue}$$

$$\bullet\langle \mathsf{testTran}\rangle\,\Omega^*_{\leq k}\,\$\,\Omega^*_{\leq k}\,\$ \xrightarrow{\Gamma\mapsto a} \bullet\langle \mathsf{testTran}^1\rangle \qquad \text{rule à la Lemma 4 to go out of synch}$$

$$\langle \mathsf{testTran}\rangle_\bullet\,\Omega^*_{\leq k}\,\$ \xrightarrow{\Gamma\mapsto aa} \langle \mathsf{testTran}^1\rangle_\bullet \qquad \text{rule à la Lemma 4 to go out of synch}$$

$$\bullet\langle \mathsf{testTran}^1\rangle\,(\Omega^*_{\leq k-1}\Omega_k)^*\,\$ \xrightarrow{T_1\ \|\ ((\Gamma\setminus\Omega_k)\mapsto a)} \bullet\langle \mathsf{stop}_k\rangle \qquad \text{rule à la Lemma 4 for } T_1$$

$$\langle \mathsf{testTran}^1\rangle_\bullet\,(\Omega^*_{\leq k-1}\Omega_k)^*\,\$ \xrightarrow{T_2\,\|\,((\Gamma\setminus\Omega_k)\mapsto a)} \langle \mathsf{stop}_k\rangle_\bullet \qquad \text{rule à la Lemma 4 for } T_2$$

Fig. 4. Macro rules used in Lemma 9.

$\eta : \Omega^*_{\leq k} \to \{0,1\}^*$ denotes the homomorphism with $\eta(\sigma) = val(\sigma)$ for $\sigma\in\Omega_k$ and $\eta(\sigma)=\varepsilon$ otherwise. The w_i's will be separated on the stack by the symbol $\$ \stackrel{\text{def}}{=} 0_{k+1}$, i.e. we shall use $\Omega_{\leq k}\cup\{0_{k+1}\}$ as the stack alphabet.

Formally, $\mathcal{P}$ is defined as follows.

$$\begin{array}{rcl}
\mathbb{I} & = & \{\mathsf{start},\mathsf{fin},\mathsf{testFin},\mathsf{next},\mathsf{next}^1,\mathsf{popAll},\mathsf{tran},\\
 & & \mathsf{testTran},\mathsf{testTran}^1\}\cup\bigcup_{0\leq\ell\leq k+2}\{\mathsf{ones}_\ell,\mathsf{ones}^1_\ell,\\
 & & \mathsf{dec}_\ell,\mathsf{dec}^1_\ell,\mathsf{zeros}_\ell,\mathsf{zeros}^1_\ell,\mathsf{decOk}_\ell,\mathsf{testDec}_\ell,\\
 & & \mathsf{testDec}^1_\ell,\mathsf{stop}_\ell\}\cup\bigcup_{1\leq i\leq n}\{\mathsf{dec}^{(i)}_0\}\\
\mathcal{B} & = & \{\,\langle\alpha\rangle \mid \alpha\in\mathbb{I}^*,\ 1\leq\lceil|\alpha|\leq k+2\}\\
Q & = & (\bullet\mathcal{B}\uplus\mathcal{B}_\bullet)\uplus Q_{impl}\\
\Gamma & = & \Omega_{\leq k}\cup\{0_{k+1}\}\\
\mathcal{A}ct & = & \{0,1,\#,a,b\}\uplus\Upsilon
\end{array}$$

The rules defining $\hookrightarrow$ are those listed in Figure 4 along with the rules from Lemma 5 and Figure 3. In Figure 4 we include brief intuitions for each of the new rules, referring to the sequence $z_0,z_1,\ldots$ associated with the transducers. Note that there are no outgoing rules involving stop_ℓ so as to satisfy the technical condition in Lemma 5.

Altogether, the rules amount to playing a game in which Defender is allowed to construct sequences while Attacker can check whether these represent a $\mathsf{Tower}(k,n)$-terminating sequence $z_0,z_1,\cdots$ ending in $0^{\mathsf{Tower}(k,n)}$.

To prove that the reduction is correct, one first shows that the three conditions below are satisfied, where $x\in\Gamma^*$ and w_1,w_2,w_3 are k-counters.

(a) $\sim\langle \mathsf{testFin}\rangle\,w_1\$x$ iff $\eta(w_1)=0^{\mathsf{Tower}(k,n)}$.

(b) $\sim\langle \mathsf{testTran}\rangle\,w_3\$w_2\$w_1\$x$ iff $T_1(\eta(w_1))=T_2(\eta(w_2))$.

(c) $\sim\langle \mathsf{start}\rangle$ iff $\mathsf{last}(\mathsf{Tower}(k,n),T_1,T_2)=0^{\mathsf{Tower}(k,n)}$.

Applying the last item, we have $\sim\langle \mathsf{start}\rangle$ if and only if $\mathsf{last}(\mathsf{Tower}(k,n),T_1,T_2)=0^{\mathsf{Tower}(k,n)}$, which completes the reduction.

Observe that the definition of $\mathcal{P}$ involves polynomially many macro rules and the size of each is polynomial in (n,T_1,T_2). Because macros can be expanded into ordinary rules in polynomial time, the overall reduction can also be performed in polynomial time. ∎

Proposition 8 and Lemma 9 imply Theorem 1.

VII. Normedness

We say that a configuration of a PDA is *normed* if each reachable configuration can reach a deadlock configuration where the stack is empty. Note that if a configuration is normed then every reachable configuration is normed too. Here we strengthen Theorem 1 to show that our lower bound remains valid even if we restrict ourselves to normed configurations.

Recall that in our previous construction, $\mathcal{P}$ contains control states $\bullet\langle \mathsf{stop}_\ell\rangle$ and $\langle \mathsf{stop}_\ell\rangle_\bullet$ for which we did not provide any outgoing transitions. Hence, once $\mathcal{P}$ reaches either of the control states, it will not be possible to empty the stack. Fortunately, the absence of outgoing transitions was not a necessary condition in our argument. Rather, we took advantage of a property that holds vacuously in these configurations:

$\bullet\langle \mathsf{stop}_\ell\rangle x \sim \langle \mathsf{stop}_\ell\rangle_\bullet w\sigma x$ for all $x \in \Gamma^*$, $\sigma \in \Omega_{\ell+1}$ and ℓ-counters w. We shall show how to modify $\mathcal{P}$ while preserving this property for normed configurations.

Theorem 10. *PDA bisimilarity is nonelementary, even when the initial configurations are normed.*

Proof: Observe that in the previous construction, all unnormed configurations have control states $\bullet\langle \mathsf{stop}_\ell\rangle$ or $\langle \mathsf{stop}_\ell\rangle_\bullet$. We amend the construction by adding rules that will allow for popping at those control states. Unfortunately, we cannot simply add the same rules as in $\bullet\langle \mathsf{popAll}\rangle$ and $\langle \mathsf{popAll}\rangle_\bullet$ due to the need to maintain $\bullet\langle \mathsf{stop}_\ell\rangle x \sim \langle \mathsf{stop}_\ell\rangle_\bullet w\sigma x$.

Note that $\bullet\langle \mathsf{stop}_\ell\rangle$, $\langle \mathsf{stop}_\ell\rangle_\bullet$ can be reached in two ways: stop_k is introduced in the rule for $\mathsf{testTran}$ and other occurrences of stop_ℓ are used in counter-related rules using $\mathsf{testDec}_\ell$. Observe that whenever $\bullet\langle \mathsf{stop}_\ell\rangle$ and $\langle \mathsf{stop}_\ell\rangle_\bullet$ are reached by the players during the bisimulation game, the stacks are out of synch by one ℓ-counter. Thus, before allowing for synchronous popping as in popAll, we will rebalance the stacks by using rules that are dual to those for $\mathsf{testDec}^1_\ell$ and $\mathsf{testTran}$. Accordingly, we add the following rules:

$$\bullet\langle \mathsf{stop}_\ell\rangle\, \Omega^*_{\leq \ell}\Omega_{\ell+1} \xrightarrow{\Gamma\mapsto aa} \bullet\langle \mathsf{popAll}\rangle$$
$$\langle \mathsf{stop}_\ell\rangle_\bullet\, \Omega^*_{\leq \ell}\Omega_{\ell+1}\Omega^*_{\leq \ell}\Omega_{\ell+1} \xrightarrow{\Gamma\mapsto a} \langle \mathsf{popAll}\rangle_\bullet .$$

Now, if we assume that $x = w'\sigma' x'$ holds, where w' is an ℓ-counter, $\sigma' \in \Omega_{\ell+1}$ and $x' \in \Gamma^*$, then $\bullet\langle \mathsf{stop}_\ell\rangle x \sim \langle \mathsf{stop}_\ell\rangle_\bullet w\sigma x$ will be satisfied, and the reachable configurations involving stop_ℓ will be normed. The only remaining problem is to make sure that the assumption is legitimate.

Recall that stop_ℓ is introduced by rules related to $\mathsf{testTran}$ and $\mathsf{testDec}_\ell$. If the players play optimally in the bisimulation game and w is not the ones ℓ-counter, then the above assumption is satisfied, because all other ℓ-counters are guaranteed to be followed by another ℓ-counter. To fix the problem with the ones ℓ-counter, for $\mathsf{testTran}$ we shall add an extra ones k-counter at the very start of the simulation. For $\mathsf{testDec}_\ell$, we shall eliminate the need for decrement tests involving ones counters by introducing a new instruction zOnes_ℓ, which will correspond to the predecessor of the ones ℓ-counter. Consequently, the predecessors of ones counters will be correct by construction, and there will be no need for testing them for correctness. The following modifications achieve the goal.

- Replace $\langle \mathsf{start}\rangle \circ\!\!\rightarrow \langle \mathsf{ones}_k\, \mathsf{fin}\rangle\, \$$ with
$$\langle \mathsf{start}\rangle \circ\!\!\rightarrow \langle \mathsf{ones}_k\, \mathsf{start}^0\rangle\, \$$$
$$\langle \mathsf{start}^0\rangle \circ\!\!\rightarrow \langle \mathsf{ones}_k\, \mathsf{fin}\rangle\, \$$$
i.e., push one additional ones k-counter and $\$$ at the beginning.
- Add control states $\bullet\langle \mathsf{zOnes}_\ell\rangle, \langle \mathsf{zOnes}_\ell\rangle_\bullet$ with the intention to push an ℓ-counter w with $val(w) = val(\mathsf{ones}_\ell) - 1 = \mathsf{Tower}(\ell+1, n) - 2$.
- Replace $\langle \mathsf{ones}_\ell\, \beta\rangle \circ\!\!\rightarrow \langle \mathsf{ones}_{\ell-1}\, \mathsf{ones}^1_\ell\, \beta\rangle\, 1_\ell$ with
$$\langle \mathsf{ones}_\ell\, \beta\rangle \circ\!\!\rightarrow \langle \mathsf{ones}_{\ell-1}\, \mathsf{ones}^0_\ell\, \beta\rangle\, 1_\ell$$
$$\langle \mathsf{ones}^0_\ell\, \beta\rangle \circ\!\!\rightarrow \langle \mathsf{zOnes}_{\ell-1}\, \mathsf{ones}^1_\ell\, \beta\rangle\, 1_\ell\,.$$
Replace $\langle \mathsf{dec}_\ell\, \beta\rangle \overset{Def}{\circ\!\!\longrightarrow} \{\langle \mathsf{ones}_{\ell-1}\, \mathsf{dec}^1_\ell\, \beta\rangle\, \sigma \mid \sigma \in \Omega_\ell\}$ with
$$\langle \mathsf{dec}_\ell\, \beta\rangle \overset{Def}{\circ\!\!\longrightarrow} \{\langle \mathsf{ones}_{\ell-1}\, \mathsf{dec}^0_\ell\, \beta\rangle\, \sigma \mid \sigma \in \Omega_\ell\}$$
$$\langle \mathsf{dec}^0_\ell\, \beta\rangle \overset{Def}{\circ\!\!\longrightarrow} \{\langle \mathsf{zOnes}_{\ell-1}\, \mathsf{dec}^1_\ell\, \beta\rangle\, \sigma \mid \sigma \in \Omega_\ell\}\,.$$
Replace $\langle \mathsf{zeros}_\ell\, \beta\rangle \circ\!\!\rightarrow \langle \mathsf{ones}_{\ell-1}\, \mathsf{zeros}^1_\ell\, \beta\rangle\, 0_\ell$ with
$$\langle \mathsf{zeros}_\ell \beta\rangle \circ\!\!\rightarrow \langle \mathsf{ones}_{\ell-1}\, \mathsf{zeros}^0_\ell\, \beta\rangle\, 0_\ell$$
$$\langle \mathsf{zeros}^0_\ell\, \beta\rangle \circ\!\!\rightarrow \langle \mathsf{zOnes}_{\ell-1}\, \mathsf{zeros}^1_\ell\, \beta\rangle\, 0_\ell\,.$$
- Finally we add the defining rules for zOnes, where $1 \leq \ell \leq k$.
$$\langle \mathsf{zOnes}_0\, \beta\rangle \circ\!\!\rightarrow \langle \beta\rangle\, 0_0 1_0^{n-1}$$
$$\langle \mathsf{zOnes}_\ell\, \beta\rangle \circ\!\!\rightarrow \langle \mathsf{ones}_{\ell-1}\, \mathsf{zOnes}^0_\ell\, \beta\rangle\, 1_\ell$$
$$\langle \mathsf{zOnes}^0_\ell\, \beta\rangle \circ\!\!\rightarrow \langle \mathsf{zOnes}_{\ell-1}\, \mathsf{zOnes}^1_\ell\, \beta\rangle\, 1_\ell$$
$$\langle \mathsf{zOnes}^1_\ell\, \beta\rangle \overset{Def}{\circ\!\!\longrightarrow} \{\langle \mathsf{dec}_{\ell-1}\, \mathsf{zOnes}^1_\ell\, \beta\rangle 1_\ell, \langle \mathsf{zeros}_{\ell-1}\, \beta\rangle\, 0_\ell\}\,.$$

By these modifications, whenever we require $\bullet\langle \mathsf{stop}_\ell\rangle\, x \sim \langle \mathsf{stop}_\ell\rangle_\bullet\, w\sigma x$ in our proofs, the word x will start with an ℓ-counter and an $\Omega_{\ell+1}$-symbol. ■

VIII. Conclusions

It is not hard to see that our lower bound can be generalized to a binary stack alphabet. One can see that the number of control states in our construction depends exponentially on the height k of the tower of exponentials. This leads to the natural question whether this exponential dependency can be lowered to a polynomial one or even a constant one. In particular, the question arises whether one can improve the currently best known EXPTIME lower bound for bisimilarity of basic process algebras (which are single-state PDA) from [13]. Because our reduction is not uniform in k, it is not clear how to extend this technique to get stronger lower bounds (e.g. non-primitive-recursive).

In current work, we are also looking at whether the technique can be applied to other analysis problems for PDA, such as the regularity problem.

Acknowledgments. Benedikt is supported in part by EP/G004021/1 and EP/H017690/1 of the Engineering and Physical Sciences Research Council UK. Kiefer is also supported by the EPSRC.

The authors thank anonymous reviewers for their constructive comments.

References

[1] G. Sénizergues, "L(A)=L(B)? Decidability results from complete formal systems," *Theoretical Computer Science*, vol. 251, no. 1-2, pp. 1–166, 2001.

[2] C. Stirling, "Deciding DPDA Equivalence Is Primitive Recursive," in *ICALP*, 2002.

[3] P. Jančar, "Decidability of DPDA language equivalence via first-order grammars," in *LICS*, 2012.

[4] R. J. van Glabbeek, "The linear time-branching time spectrum (extended abstract)," in *CONCUR*, 1990.

[5] J. van Benthem, "Modal correspondence theory," Ph.D. dissertation, University of Amsterdam, 1976.

[6] D. Janin and I. Walukiewicz, "On the Expressive Completeness of the Propositional mu-Calculus with Respect to Monadic Second Order Logic," in *CONCUR*, 1996.
[7] R. Milner, *Communication and Concurrency*. Prentice Hall, 1989.
[8] A. Kučera and P. Jančar, "Equivalence-checking on infinite-state systems: Techniques and results," *Theory and Practice of Logic Programming*, vol. 6, no. 3, pp. 227–264, 2006.
[9] G. Sénizergues, "The bisimulation problem for equational graphs of finite out-degree," *SIAM Journal on Computing*, vol. 34, no. 5, pp. 1025–1106, 2005.
[10] P. Jančar and J. Srba, "Undecidability of bisimilarity by defender's forcing," *J. ACM*, vol. 55, no. 1, 2008.
[11] C. Broadbent and S. Göller, "On Bisimilarity on Higher-Order Pushdown Automata: Undecidability at Order Two," in *FSTTCS*, 2012.
[12] A. Kučera and R. Mayr, "On the complexity of checking semantic equivalences between pushdown processes and finite-state processes," *Information and Computation*, vol. 208, no. 7, pp. 772–796, 2010.
[13] S. Kiefer, "BPA bisimilarity is EXPTIME-hard," *Information Processing Letters*, vol. 113, no. 4, pp. 101–106, 2013.
[14] O. Burkart, D. Caucal, and B. Steffen, "An elementary bisimulation decision procedure for arbitrary context-free processes," in *MFCS*, 1995.
[15] P. Jancar, "Bisimilarity on basic process algebra is in 2-exptime (an explicit proof)," *CoRR*, vol. abs/1207.2479, 2012.
[16] P. Jančar, "Strong bisimilarity on basic parallel processes is pspace-complete," in *LICS*, 2003.
[17] S. Böhm, S. Göller, and P. Jančar, "Bisimilarity of one-counter processes is PSPACE-complete," in *Proc. of CONCUR*, ser. Lecture Notes in Computer Science, vol. 6269. Springer, 2010, pp. 177–191.
[18] L. J. Stockmeyer, "The complexity of decision problems in automata and logic," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, 1974.