R-automata*

Parosh Aziz Abdulla, Pavel Krcal, and Wang Yi

Department of Information Technology,
Uppsala University, Sweden
Email: {parosh,pavelk,yi}@it.uu.se

Abstract. We introduce *R-automata* – a model for analysis of systems with resources which are consumed in small parts but which can be replenished at once. An R-automaton is a finite state machine which operates on a finite number of unbounded counters (modeling the resources). The values of the counters can be incremented, reset to zero, or left unchanged along the transitions. We define the language accepted by an R-automaton relative to a natural number D as the set of words allowing a run along which no counter value exceeds D. As the main result, we show decidability of the universality problem, i.e., the problem whether there is a number D such that the corresponding language is universal. The decidability proof is based on a reformulation of the problem in the language of finite monoids and solving it using the factorization forest theorem. This approach extends the way in which the factorization forest theorem was used to solve the limitedness problem for distance automata in [Sim94]. We also show decidability of the non-emptiness problem and the limitedness problem, i.e., whether there is a natural number D such that the corresponding language is non-empty resp. all the accepted words can also be accepted with counter values smaller than D. Finally, we extend the decidability results to R-automata with Büchi acceptance conditions.

1 Introduction

We consider systems operating on resources which are consumed in small parts and which can be (or have to be) replenished completely at once. To model such systems, we introduce *R-automata* – finite state machines extended by a finite number of unbounded counters corresponding to the resources. The counters can be incremented, reset to zero, or left unchanged along the transitions. When the value of a counter is equal to zero then the stock of this resource is full. Incrementing a counter means using one unit of the resource and resetting a counter means the full replenishment of the stock.

We define the language accepted by an R-automaton relative to a natural number D as the set of words allowing an accepting run of the automaton such that no counter value exceeds D in any state along the run. We study the problem of whether there is a number D such that the corresponding language is universal.

* This work has been partially supported by the EU CREDO project.

This problem corresponds to the fact that with stock size D, the system can exhibit all the behaviors without running out of resources. We show that this problem is decidable in 2-EXPSPACE. We extend this result to show decidability of the limitedness problem, i.e., to decide whether there is a natural number D such that all the accepted words can also be accepted with the counter value smaller than D. We also show the decidability of the non-emptiness problem. As a second technical contribution, we extend the decidability of the universality problem to R-automata with Büchi acceptance conditions.

To prove decidability of the universality problem, we adopt the technique from [Sim94] and extend it to our setting. We reformulate the problem in the language of finite monoids and solve it using the factorization forest theorem [Sim90]. In [Sim94], this theorem is used for solving the limitedness problem for distance automata. Distance automata are a subclass of R-automata with only one counter which is never reset. In contrast to this model, we handle several counters and resets. This extension cannot be encoded into the distance automata.

The decision algorithm deals with abstractions of collections of runs in order to find and analyze the loops created by these collections. The main step in the correctness proof is to show that each collection of runs along the same word can be split (factorized) into short repeated loops, possibly nested. Having such a factorization, one can analyze all the loops to check that none of the counters is only increased without being reset along them. If none of the counters is increased without being reset then we can bound the counter values by a constant derived from the length of the loops. Since the length of the loops is bounded by a constant derived from the automaton, all words can be accepted by a run with bounded counters. Otherwise, we show that there is a +-free regular expression such that for any bound there is a word obtained by pumping this regular expression which does not belong to the language. Therefore, the language cannot be universal for any D.

Related work. The concept of distance automata and the limitedness problem were introduced by Hashiguchi [Has82]. Different proofs of the decidability of the limitedness problem are reported in [Has90,Leu91,Sim94]. The last of these results [Sim94] is based on the factorization forest theorem [Sim90,Col07]. The model of R-automata, which we consider in this paper, extends that of distance automata by introducing resets and by allowing several counters. Furthermore, all the works mentioned above only consider the limitedness problem on finite words, while we here extend the decidability result of the universality problem to the case of infinite words. Distance automata were extended in [Kir05] with additional counters which can be reset following a hierarchical discipline resembling parity acceptance conditions. R-automata relax this discipline and allow the counters to be reset arbitrarily. Universality of a similar type of automata for tree languages is studied in [CL08]. A model with counters which can be incremented and reset in the same way as in R-automata, called B-automata, is presented in [BC06]. B-automata accept infinite words such that the counters are bounded along an infinite accepting computation. Decidability of our problems can be obtained using the results from [BC06]. However, this would require complementation of a B-automaton which results in a non-elementary blowup of the automaton state space.

The fact that R-automata can have several counters which can be reset allows, for instance, to capture the abstractions of the sampled semantics of timed automata [KP05,AKY07]. A sampled semantics given by a sampling rate $\epsilon = 1/f$ for some positive integer f allows time to pass only in steps equal to multiples of ϵ. The number of different clock valuations within one clock region (a bounded set of valuations) corresponds to a resource. It is finite for any ϵ while infinite in the standard (dense time) semantics of timed automata. Timed automata can generate runs along which clocks are forced to take different values from the same clock region (an increment of a counter), take exactly the same value (a counter is left unchanged), or forget about the previously taken values (a counter reset).

2 Preliminaries

First, we introduce the model of R-automata and its unparameterized semantics. Then, we introduce the parameterized semantics, the languages accepted by the automaton, and the decision problems.

R-automata. R-automata are finite state machines extended with counters. A transition may increase the value of a counter, leave it unchanged, or reset it back to zero. The automaton on its own does not have the capability of testing the values of the counters. However, the semantics of these automata is parameterized by a natural number D which defines an upper bound on counter values which may appear along the computations of the automaton. Let $\mathbb{N}$ denote the set of non-negative integers.

An *R-automaton* with n counters is a 5-tuple $A = \langle S, \Sigma, \Delta, s_0, F \rangle$ where

- S is a finite set of states,
- Σ is a finite alphabet,
- $\Delta \subseteq S \times \Sigma \times \{0, 1, r\}^n \times S$ is a transition relation,
- $s_0 \in S$ is an initial state, and
- $F \subseteq S$ is a set of final states.

Transitions are labeled (together with a letter) by an effect on the counters. The symbol 0 corresponds to leaving the counter value unchanged, the symbol 1 represents an increment, and the symbol r represents a reset. We use $t, t_1, \ldots$ to denote elements of $\{0, 1, r\}^n$ which we call *effects*. A *path* is a sequences of transitions (s_1, a_1, t_1, s_2),$(s_2, a_2, t_2, s_3), \ldots, (s_m, a_m, t_m, s_{m+1})$, such that $\forall 1 \leq i \leq m.(s_i, a_i, t_i, s_{i+1}) \in \Delta$. An example of an R-automaton is given in Figure 1.

Unparameterized semantics. We define an operation $\oplus$ on the counter values as follows: for any $k \in \mathbb{N}$, $k \oplus 0 = k$, $k \oplus 1 = k + 1$, and $k \oplus r = 0$. We extend this operation to n-tuples by applying it componentwise. The operational semantics of an R-automaton $A = \langle S, \Sigma, \Delta, s_0, F \rangle$ is given by a labeled transition system (LTS) $[\![A]\!] = \langle \hat{S}, \Sigma, T, \hat{s_0} \rangle$, where the set of states $\hat{S}$ contains pairs $\langle s, (c_1, \ldots, c_n) \rangle$, $s \in S, c_i \in \mathbb{N}$ for all $1 \leq i \leq n$, with the initial state

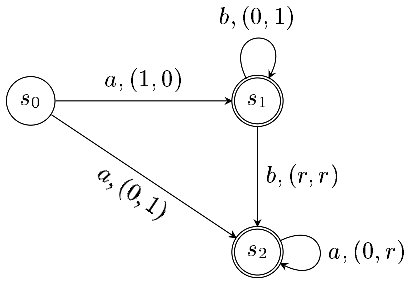

Fig. 1. An R-automaton with two counters.

$\hat{s_0} = \langle s_0, (0, \ldots, 0)\rangle$. The transition relation is defined by $(\langle s, (c_1, \ldots, c_n)\rangle, a, \langle s, (c'_1, \ldots, c'_n)\rangle) \in T$ if and only if $\langle s, a, t, s'\rangle \in \Delta$ and $(c'_1, \ldots, c'_n) = (c_1, \ldots, c_n) \oplus t$. We shall call the states of the LTS *configurations*.

We write $\langle s, (c_1, \ldots, c_n)\rangle \xrightarrow{a} \langle s, (c'_1, \ldots, c'_n)\rangle$ if $(\langle s, (c_1, \ldots, c_n)\rangle, a, \langle s, (c'_1, \ldots, c'_n)\rangle) \in T$. We extend this notation also for words, $\langle s, (c_1, \ldots, c_n)\rangle \xrightarrow{w} \langle s, (c'_1, \ldots, c'_n)\rangle$, where $w \in \Sigma^+$.

Paths in an LTS are called *runs* to distinguish them from paths in the underlying R-automaton. Observe that the LTS contains infinitely many states, but the counter values do not influence the computations, since they are not tested anywhere. In fact, for any R-automaton A, $[\![A]\!]$ is bisimilar to A considered as a finite automaton (without counters and effects). The LTS induced by the R-automaton from Figure 1 is in Figure 2.

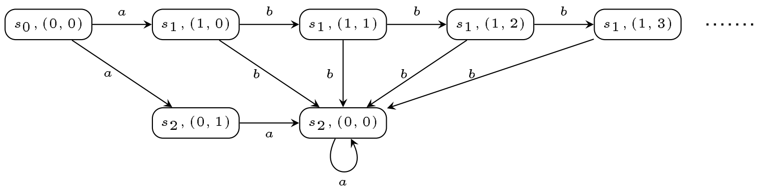

Fig. 2. The unparameterized semantics of the R-automaton in Figure 1.

Parameterized Semantics. Next, we define the D-semantics of R-automata. We assume that the resources associated to the counters are not infinite and we can use them only for a bounded number of times before they are replenished again. If a machine tries to use a resource which is already completely used up, it is blocked and cannot continue its computation.

For a given $D \in \mathbb{N}$, let $\hat{S}_D$ be the set of configurations restricted to the configurations which do not contain a counter exceeding D, i.e., $\hat{S}_D = \{\langle s, (c_1, \ldots, c_n)\rangle | \langle s, (c_1, \ldots, c_n)\rangle \in \hat{S}$ and $(c_1, \ldots, c_n) \leq (D, \ldots, D)\}$ ($\leq$ is applied componentwise). For an R-automaton A, the *D-semantics* of A, denoted by $[\![A]\!]_D$, is $[\![A]\!]$

restricted to $\hat{S}_D$. We write $\langle s,(c_1,\ldots,c_n)\rangle \xrightarrow{a}_D \langle s,(c'_1,\ldots,c'_n)\rangle$ to denote the transition relation of $[\![A]\!]_D$. We extend this notation for words, $\langle s,(c_1,\ldots,c_n)\rangle \xrightarrow{w}_D \langle s,(c'_1,\ldots,c'_n)\rangle$ where $w \in \Sigma^+$. The 2-semantics of the R-automaton from Figure 1 is in Figure 3.

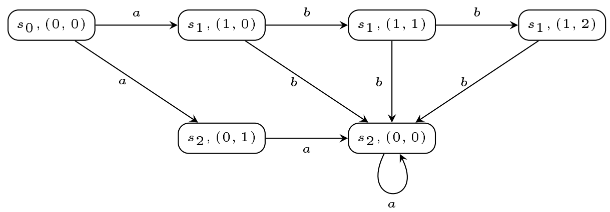

Fig. 3. The 2-semantics of the R-automaton in Figure 1.

It is easy to see that for each $D_1 < D_2$, $[\![A]\!]_{D_2}$ simulates $[\![A]\!]_{D_1}$ and $[\![A]\!]$ simulates $[\![A]\!]_{D_2}$. Even stronger, for each $\hat{s} \in \hat{S}_{D_1}$, let $\hat{s}_{D_1}, \hat{s}_{D_2}, \hat{s}$ denote the configurations in $[\![A]\!]_{D_1}, [\![A]\!]_{D_2}, [\![A]\!]$, respectively. Then $\hat{s}_{D_2}$ simulates $\hat{s}_{D_1}$ and $\hat{s}$ simulates $\hat{s}_{D_2}$.

We abuse the notation to avoid stating the counter values explicitly when it is not necessary. We define the reachability relations $\longrightarrow$ and $\longrightarrow_D$ over pairs of states and words as follows. For $s, s' \in S$ and $w \in \Sigma^+$, $s \xrightarrow{w} s'$ if and only if there is a path (s, a_1, t_1, s_1), (s_1, a_2, t_2, s_2), $\ldots$, $(s_{|w|-1}, a_{|w|}, t_{|w|}, s')$ such that $w = a_1 \cdot a_2 \cdots a_{|w|}$. For each $D \in \mathbb{N}$, $s \xrightarrow{w}_D s'$ if also for all $1 \le i \le |w|$, $t_1 \oplus t_2 \oplus \cdots \oplus t_i \le (D,\ldots,D)$. It also holds that $s \xrightarrow{w}_D s'$ if and only if there is a run $\langle s,(0,\ldots,0)\rangle \xrightarrow{w}_D \langle s',(c_1,\ldots,c_n)\rangle$.

Language. The (unparameterized or D-) language of an R-automaton is the set of words which can be read along the runs in the corresponding LTS ending in an accepting state (in a configuration whose first component is an accepting state). The *unparameterized language* accepted by an R-automaton A is $L(A) = \{w | s_0 \xrightarrow{w} s_f, s_f \in F\}$. For a given $D \in \mathbb{N}$, the *D-language* accepted by an R-automaton A is $L_D(A) = \{w | s_0 \xrightarrow{w}_D s_f, s_f \in F\}$. The unparameterized language of the R-automaton from Figure 1 is ab^*a^*. The 2-language of this automaton is $a(\epsilon + b + bb + bbb)a^*$.

Problem Definition. Now we can ask a question about language non-emptiness or universality of an R-automaton A parameterized by D, i.e., is there a natural number D such that $L_D(A) = \emptyset$ or $L_D(A) = \Sigma^*$. Figure 4 shows an R-automaton A such that $L_2(A) = \Sigma^*$.

The language definitions and the questions can also be formulated for infinite words with Büchi acceptance conditions. The unparameterized ω-language of the automaton from Figure 1 is $ab^\omega + ab^*a^\omega$. The 2-ω-language of this automaton is $a(\epsilon + b + bb + bbb)a^\omega$.

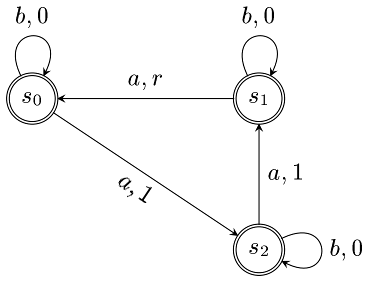

Fig. 4. A 2-universal R-automaton.

3 Universality

The main result of the paper is the decidability of the universality problem for R-automata formulated in the following theorem.

Theorem 1. *For a given R-automaton A, the question whether there is $D \in \mathbb{N}$ such that $L_D(A) = \Sigma^*$ is decidable in 2-EXPSPACE.*

First, we introduce and also formally define the necessary concepts (patterns, factorization, and reduction) together with an overview of the whole proof. Then we show the construction of the reduced factorization trees and state the correctness of this construction. Finally, we present an algorithm for deciding universality.

3.1 Concepts and Proof Overview

When an R-automaton A is not universal for all $D \in \mathbb{N}$ then there is an infinite set X of words such that for each $D \in \mathbb{N}$ there is $w_D \in X$ and $w_D \notin L_D(A)$. We say then that X is a counterexample. The main step of the proof is to show that there is an X which can be characterized by a $+$-free regular expression. In fact, we show that X also satisfies a number of additional properties which enable us to decide for every such a $+$-free regular expression, whether it corresponds to a counterexample or not. Another step of the proof is to show that we need to check only finitely many such $+$-free regular expressions in order to decide whether there is a counterexample at all.

Patterns. The standard procedure for checking universality in the case of finite automata is subset construction. Whenever there are non-deterministic transitions $s \xrightarrow{a} s_1$ and $s \xrightarrow{a} s_2$ then we build a "summary" transition $\{s\} \xrightarrow{a} \{s_1, s_2\}$. This summary transition says that from the set of states $\{s\}$ we get to the set of states $\{s_1, s_2\}$ after reading the letter a. In the case of R-automata, subset construction is in general not guaranteed to terminate since the values of the counters might grow unboundedly. To deal with this problem, we exploit the fact that the values of the counters do not influence the computations of the automaton. Therefore, we perform an abstraction which hides the actual values of the counters and considers only the effects along the transitions instead. The abstraction leads to a more complicated variant of summary transitions namely so called *patterns*.

We define a commutative, associative, and idempotent operation $\circ$ on the set $\{0, 1, r\}$: $0 \circ 0 = 0$, $0 \circ 1 = 1$, $0 \circ r = r$, $1 \circ 1 = 1$, $1 \circ r = r$, and $r \circ r = r$. In fact, if we define an order $0 < 1 < r$ then $\circ$ is the operation of taking the maximum. We extend this operation to effects, i.e., n-tuples, by applying it componentwise (this preserves all the properties of $\circ$). An effect obtained by adding several other effects through the application of the operator $\circ$ summarizes the manner in which the counters are changed. More precisely, it describes whether a counter is reset or whether it is increased but not reset or whether it is only left untouched.

A pattern $\sigma : (S \times S) \longrightarrow 2^{\{0,1,r\}^n}$ is a function from pairs of automaton states to sets of effects. Let us denote patterns by $\sigma, \sigma_1, \sigma', \ldots$. As an example, consider a pattern σ involving states s and s' and two counters. Let $\sigma(s, s) = \{(0, 0), (1, 1)\}, \sigma(s', s') = \{(1, 1), (1, 0)\}, \sigma(s, s') = \{(1, 1)\}$ and $\sigma(s', s) = \{(1, 1)\}$. This pattern is depicted in Figure 5a.

Clearly, for a given R-automaton there are only finitely many patterns; let us denote this finite set of all patterns by $\mathbb{P}$. We define an operation $\bullet$ on $\mathbb{P}$ as follows. Let $(\sigma_1 \bullet \sigma_2)(s, s') = \{t | \exists s'', t_1, t_2.\ t_1 \in \sigma_1(s, s''), t_2 \in \sigma_2(s'', s'), t = t_1 \circ t_2\}$. Note, that $\bullet$ is associative and it has a unit σ_e, where $\sigma_e(s, s') = \{(0, \ldots, 0)\}$ if $s = s'$ and $\sigma_e(s, s') = \emptyset$ otherwise. Therefore, $(\mathbb{P}, \bullet)$ is a finite monoid.

For each word we obtain a pattern by running the R-automaton along this word. Formally, let $\mathsf{Run} : \Sigma^+ \longrightarrow \mathbb{P}$ be a homomorphism defined by $\mathsf{Run}(a) = \sigma$, where $t \in \sigma(s, s')$ if and only if $(s, a, t, s') \in \Delta$.

Loops. In the case of finite automata, a set of states L and a word w constitute a loop in the subset construction if $L \xrightarrow{w} L$, i.e., starting from L and reading w, we end up in L again. The intuition behind the concept of a loop is that several iterations of the loop have the same effect as a single iteration. In our abstraction using patterns, loops are words w such that w yields the same pattern as $w^2, w^3, \ldots$. We can skip the starting set of states, because the function Run starts implicitly from the whole set of states S (if there are no runs between some states then the corresponding set of effects is empty). More precisely, a word w is a loop if $\mathsf{Run}(w)$ is an idempotent element of the pattern monoid. Two loops are identical if they produce the same pattern. Observe that the pattern in Figure 5a is idempotent.

Factorization. We show that each word can be split into *short identical loops* repeated many times. The loops can possibly be nested, so that this split (*factorization*) defines a factorization tree. The idea is that since we have such a factorization for each word, it is sufficient to analyze only the (short) loops and either find a run with bounded maximal value of the counters or use the loop structure to construct a counterexample regular expression.

On a higher level we can see a factorization of words as a function which for every word $w = a_1 a_2 \cdots a_l$ returns its factorization tree, i.e., a finite tree with branching degree at least 2 (except for the leaves) and with nodes labeled by subwords v of w such that the labeling function satisfies the following conditions:

- if a node labeled by v has children labeled by $w_1, w_2, \ldots, w_m$ then $v = w_1 \cdot w_2 \cdots w_m$,
- if $m \geq 3$ then $\sigma = \mathsf{Run}(v) = \mathsf{Run}(w_i)$ for all $1 \leq i \leq m$ and σ is idempotent,
- the leaves are labeled by $a_1, a_2, \ldots, a_l$ from left to right.

An example of such a tree is in Figure 5b. It follows from the factorization forest theorem [Sim90,Col07] that there is such a (total) function which returns trees whose height is bounded by $3 \cdot |\mathbb{P}|$ where $|\mathbb{P}|$ is the size of the monoid.

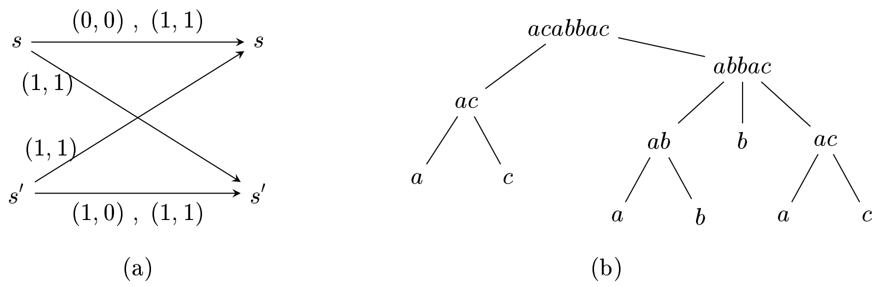

Fig. 5. A pattern involving two states and two counters (a) and a factorization tree (b). $\mathsf{Run}(abbac) = \mathsf{Run}(ab) = \mathsf{Run}(b) = \mathsf{Run}(ac)$ and it is idempotent.

We define the length of a loop as the length of the word (or a pattern sequence) provided that only the two longest iterations of the nested loops are counted. This concept is defined formally in Subsection 3.3. We say that the loops are short if there is a bound given by the automaton so that the length of all the loops is shorter than this bound. A consequence of the factorization forest theorem is that there is a factorization such that all loops are short.

Reduction. We have defined the loops so that the iterations of a loop have the same effect as the loop itself. Therefore, it is enough to analyze a single iteration to tell how the computations look when the loop is iterated an arbitrary number of times. By a *part* in an idempotent pattern σ, we mean an element (an effect) in the set $\sigma(s, s')$ for some states s and s'. We will distinguish between two types of parts, namely *bad* and *good* parts. A bad part corresponds only to runs along which the increase of some counter is at least as big as the number of the iterations of the loop. A part is *good* if there is a run with this effect along which the increase is bounded by the maximal increase induced by two iterations of the loop. Formally, we define a function reduce which for each pattern returns a pattern containing all good parts of the original pattern, but no bad parts. Then we illustrate it on a number of examples.

For a pattern σ, $\mathsf{core}(\sigma)$ is defined as follows:

$$\mathsf{core}(\sigma)(s,s') = \begin{cases} \sigma(s,s') \cap \{0,r\}^n & \text{if } s = s' \\ \emptyset & \text{otherwise} \end{cases}$$

Let $\mathsf{reduce}(\sigma) = \sigma \bullet \mathsf{core}(\sigma) \bullet \sigma$.

For an automaton with one state s, one counter, and a loop w with pattern σ, if $\sigma(s,s) = \{(1)\}$ then the whole pattern is bad, i.e., $\mathsf{reduce}(\sigma)(s,s) = \emptyset$. Notice that any run over w^k increases the counter by k. On the other hand, if $\sigma(s,s) = \{(0)\}$ or $\sigma(s,s) = \{(r)\}$ then the whole pattern is good, i.e., $\mathsf{reduce}(\sigma) = \sigma$.

With more complicated patterns we need a more careful analysis. Let us consider a loop w with pattern σ where $\sigma(s,s) = \{(0)\}$, $\sigma(s',s') = \{(1)\}$, $\sigma(s,s') = \{(1)\}$, and $\sigma(s',s) = \{(1)\}$. We will motivate why the part $(1) \in \sigma(s',s')$ is good. For any k, we can take the run over w^k which starts from s', moves to s after the first iteration, stays in s for $k-2$ iterations, and finally moves back to s' after the k^{th} iteration. Then, the effect of the run is (1). Furthermore, the counter increase along the run is bounded by twice the maximal counter increase while reading w. In fact, using a similar reasoning, we can show that all parts of σ are good (which is consistent with the fact that $\mathsf{reduce}(\sigma) = \sigma$).

As the last example, let us consider the pattern from Figure 5a. First, we show that the part $(1,0) \in \sigma(s',s')$ is bad. The only run over w^k with effect $(1,0)$ is the one which comes back to s' after each iteration. However, this run increases the first counter by k. On the other hand, the part $(1,1) \in \sigma(s',s')$ is good by a similar reasoning to the previous example. In fact, we can show that all other parts of the pattern are good (which is consistent with the value of $\mathsf{reduce}(\sigma)$ in Figure 6).

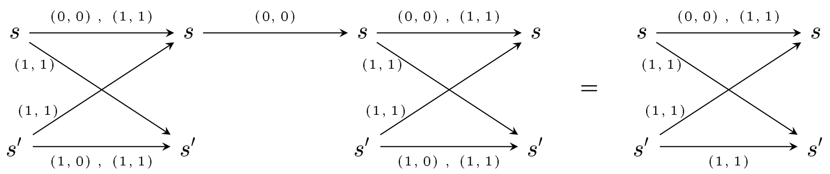

Fig. 6. $\sigma \bullet \mathsf{core}(\sigma) \bullet \sigma = \mathsf{reduce}(\sigma)$ where σ is the pattern from Figure 5a

Reduced Factorization Trees. For a factorization of a word w, we need to check whether there is a run which goes through a good part in every loop. In order to do that, we enrich the tree structure, so that each node will now be labeled, in addition to a word, also by a pattern. The patterns are added by the following function: given an input sequence of patterns, the leaves are labeled by the elements of the sequence, nodes with branching degree 2 are labeled by the composition of the children labels, and we label each node with branching degree at least 3 by σ, where σ is the idempotent label of all its children. Now, based on this labeling, we build a *reduced factorization tree* for w in several steps (formally described in Subsection 3.2).

We start with the sequence of patterns obtained by Run from the letters of the word. In each step, we take the resulting sequence from the previous step, build a factorization tree from it, and label it by patterns as described above. Then we take the lowest nodes such that they have at least 3 children and they are labeled by a pattern σ such that $\mathsf{reduce}(\sigma) \neq \sigma$. We change the labels of these nodes to $\mathsf{reduce}(\sigma)$. We pack the subtrees of these nodes into elements of the new sequence and we leave other elements of the sequence unmodified. This procedure eventually terminates and returns one tree with the following properties (the important invariant is shown in Lemma 1):

- if a node labeled by σ has two children labeled by σ_1, σ_2 then $\sigma = \sigma_1 \bullet \sigma_2$,
- if a node labeled by σ has m children labeled by $\sigma_1, \ldots, \sigma_m$, $m \geq 3$, then $\sigma_i = \sigma_j$ for all $1 \leq i, j \leq m$, σ_1 is idempotent, and $\sigma = \mathsf{reduce}(\sigma_1)$.

An example of a reduced factorization tree is in Figure 7. We show that there is a factorization function such that the height of all reduced factorization trees produced by it is bounded by $3 \cdot |\mathbb{P}|^2$ (Lemma 3) using the factorization forest theorem and a property of the reduction function that if $\mathsf{reduce}(\sigma) \neq \sigma$ then $\mathsf{reduce}(\sigma) <_{\mathcal{J}} \sigma$, where $<_{\mathcal{J}}$ is the usual ordering of the $\mathcal{J}$-classes on $\mathbb{P}$, $\mathcal{J}$ is a standard Green's relation; $\sigma \leq_{\mathcal{J}} \sigma'$ if and only if there are σ_1, σ_2 such that $\sigma = \sigma_1 \bullet \sigma' \bullet \sigma_2$; $\sigma <_{\mathcal{J}} \sigma'$ if and only if $\sigma \leq_{\mathcal{J}} \sigma'$ and $\sigma' \not\leq_{\mathcal{J}} \sigma$ (Lemma 2).

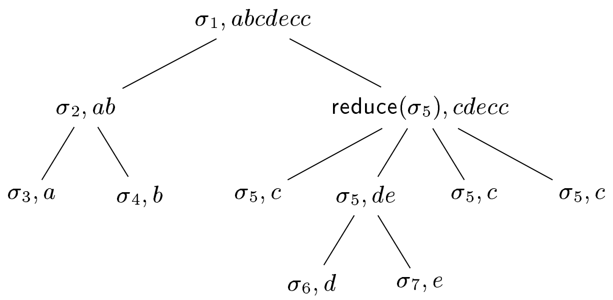

Fig. 7. An example reduced factorization tree. $\sigma_1 = \sigma_2 \bullet \mathsf{reduce}(\sigma_5), \sigma_2 = \sigma_3 \bullet \sigma_4$, and $\sigma_5 = \sigma_6 \bullet \sigma_7$. For all leaves labeled by $\hat{\sigma}, \hat{a}$, $\hat{\sigma} = \mathsf{Run}(\hat{a})$.

Correctness. Let σ be the label of the root of a reduced factorization tree for a word w and let $\mathsf{pump}(r, k)$ for a $+$-free regular expression r and for a $k \in \mathbb{N}$ be the word obtained by repeating each r_1, where r_1^* is a subexpression of r, k-times. Then

- if $\sigma(s_0, s_f) \neq \emptyset$ for some $s_f \in F$ then there is a run from s_0 to s over w in $8^{|\mathbb{P}|^2}$-semantics,
- otherwise, there is a $+$-free regular expression r such that for all D there is a k such that there is a counter which exceeds D along all runs from s_0 to s_f, $s_f \in F$, over $\mathsf{pump}(r, k)$.

The previous items are formulated in Subsection 3.3, Lemma 5 and Lemma 6.

Relation to Simon's Approach. There are several important differences between the method presented in this paper and that of Simon [Sim94]. Our notion of pattern is a function to a set of effects, while in Simon's case it is a function to the set $\{0, 1, \omega\}$. Because of the resets and the fact that there are several counters, it is not possible to linearly order the effects. Thus, a collection of automaton runs can be abstracted into several incomparable effects. The sets are necessary in order to remember all of them. Furthermore, the different notion of pattern requires a new notion of reduction which does not remove loops labeled also by resets. We need to show then that application of this notion of reduction during the construction of the reduced factorization trees preserves the correctness.

3.2 Construction of the Reduced Factorization Tree

We define labeled finite trees to capture the looping structure of pattern sequences. Let Γ be a set of finite trees with two labeling functions Pat and Word, which for each node return a pattern and a word, respectively. We will abuse the notation and, for a tree T, we use $\mathsf{Pat}(T)$ or $\mathsf{Word}(T)$ to denote $\mathsf{Pat}(N)$ or $\mathsf{Word}(N)$, respectively, where N is the root of T. We also identify nodes with the subtrees in which they are roots. We can then say that a node T has children $T_1, \ldots, T_m$ and then use T_i's as trees. For a tree T, we define its height $h(T)$ as $h(T) = 1$ if T is a leaf, $h(T) = 1 + \max\{h(T_1), \ldots, h(T_m)\}$ if $T_1, \ldots, T_m$ are children of the root of T.

By Γ^+ we mean the set of nonempty sequences of elements of Γ. By $(\Gamma^+)^+$ we mean the set of nonempty sequences of elements of Γ^+. Let us denote elements of Γ^+ by $\gamma, \gamma_1, \gamma', \ldots$. For $\gamma \in \Gamma^+$, let $|\gamma|$ denote the length of γ.

Let $f : \Gamma^+ \to \mathbb{P}$ be a homomorphism with respect to $\bullet$ defined by $f(T) = \mathsf{Pat}(T)$. We call a function $d : \Gamma^+ \to (\Gamma^+)^+$ a *factorization function* if it satisfies the following conditions. If $d(\gamma) = (\gamma_1, \gamma_2, \ldots, \gamma_m)$ then $\gamma = \gamma_1 \cdot \gamma_2 \cdots \gamma_m$, if $m = 1$ then $|\gamma| = 1$, and if $m \geq 3$ then $f(\gamma) = f(\gamma_i)$ for all $1 \leq i \leq m$ and $f(\gamma)$ is an idempotent element.

For a factorization function d we define two functions $\mathsf{tree} : \Gamma^+ \to \Gamma$ and $\mathsf{cons} : \Gamma^+ \to \Gamma^+$ inductively as follows. Let $\langle \sigma, w \rangle$ denote a tree which consists of only the root labeled by σ and w.

$$\mathsf{tree}(\gamma) = \begin{cases} \gamma & \text{if } |\gamma| = 1, \\ \langle \sigma_1 \bullet \sigma_2, w_1 \cdot w_2 \rangle \text{ with children } \mathsf{tree}(\gamma_1), \mathsf{tree}(\gamma_2), & \text{if } d(\gamma) = (\gamma_1, \gamma_2), \\ \qquad \sigma_i = \mathsf{Pat}(\mathsf{tree}(\gamma_i)), w_i = \mathsf{Word}(\mathsf{tree}(\gamma_i)) \text{ for } i \in \{1, 2\}, \\ \langle \mathsf{reduce}(\sigma), w_1 \cdot w_2 \cdots w_m \rangle \text{ with children } \mathsf{tree}(\gamma_1), \ldots, \mathsf{tree}(\gamma_m), \text{if} \\ \qquad m \geq 3, d(\gamma) = (\gamma_1, \gamma_2, \ldots, \gamma_m), \sigma = \mathsf{Pat}(\mathsf{tree}(\gamma_1)), \text{and} \\ \qquad w_i = \mathsf{Word}(\mathsf{tree}(\gamma_i)) \text{ for all } 1 \leq i \leq m. \end{cases}$$

The function tree builds a tree (resembling a factorization tree) from the sequence of trees according to the function d. The only difference from straightforwardly following the function d is that the labeling function Pat might be changed by the function reduce. Let us color the trees in the function cons either green or red during the inductive construction of a new sequence.

$$\mathsf{cons}(\gamma) = \begin{cases} \gamma & \text{if } |\gamma| = 1. \text{ Mark } \gamma \text{ green.} \\ \mathsf{cons}(\gamma_1) \cdot \mathsf{cons}(\gamma_2) \cdots \mathsf{cons}(\gamma_m) & \\ & \text{if } d(\gamma) = (\gamma_1, \gamma_2, \dots, \gamma_m) \text{ and either } m = 2 \text{ or} \\ & \text{there is } 1 \le i \le m \text{ such that } \mathsf{cons}(\gamma_i) \text{ contains} \\ & \text{a red tree or } \mathsf{reduce}(f(\gamma_1)) = f(\gamma_1). \\ \mathsf{tree}(\gamma) & \text{if } d(\gamma) = (\gamma_1, \gamma_2, \dots, \gamma_m), m \ge 3, \text{ no } \mathsf{cons}(\gamma_i) \\ & \text{contains a red tree and } \mathsf{reduce}(f(\gamma_1)) \ne f(\gamma_1). \\ & \text{Mark the tree red.} \end{cases}$$

The function cons updates the sequence of trees trying to leave as much as possible untouched, but whenever Pat would be changed by the reduce function for the first time (on the lowest level), it packs the whole sequence into a single tree with changed Pat label of the root using the function tree.

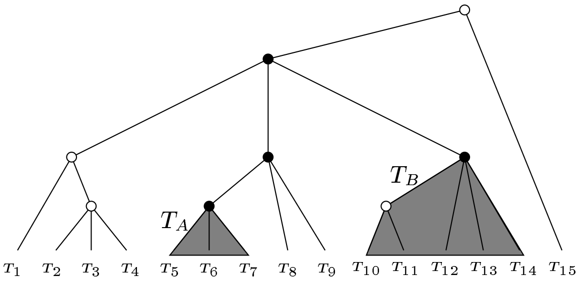

Fig. 8. Application of cons to $T_1 \cdots T_{15}$. The black nodes represent the nodes for which $\mathsf{reduce}(\sigma) \ne \sigma$. The resulting sequence is $T_1 T_2 T_3 T_4 T_A T_8 T_9 T_B T_{15}$.

The important property of the construction is that for each tree in the new sequence it holds that whenever a node has more than two children, they are all labeled by identical idempotent patterns. Let us call a tree *balanced* if whenever a node T has children $T_1, T_2, \dots, T_m$, where $m \ge 3$, then $\mathsf{Pat}(T_1) = \mathsf{Pat}(T_2) = \cdots = \mathsf{Pat}(T_m)$, it is an idempotent element in $\mathbb{P}$, and $\mathsf{Pat}(T) = \mathsf{reduce}(\mathsf{Pat}(T_1))$.

Lemma 1. *For a $\gamma \in \Gamma^+$, if all trees in γ are balanced then all trees in* $\mathsf{cons}(\gamma)$ *are balanced.*

Proof. The only possibility where a new tree can occur in $\mathsf{cons}(\gamma)$ is as a result of $\mathsf{tree}(\gamma')$ for some γ'. The conditions on γ' are that $d(\gamma) = (\gamma_1, \dots, \gamma_m)$ and for all $1 \le i \le m$, $\mathsf{cons}(\gamma_i)$ does not contain a red tree. Then we prove that $\mathsf{Pat}(\mathsf{tree}(\gamma)) = f(\gamma)$ for any $\gamma \in \Gamma^+$ such that $\mathsf{cons}(\gamma)$ contains only green trees by induction on $h(\mathsf{tree}(\gamma))$. If $h(\mathsf{tree}(\gamma)) = 1$ then it follows directly from the definition of f. If $h(\mathsf{tree}(\gamma)) > 1$ and $d(\gamma) = (\gamma_1, \gamma_2)$ then the claim follows from the induction hypothesis and the fact that f is a homomorphism. If $h(\mathsf{tree}(\gamma)) > 1$ and $d(\gamma) = (\gamma_1, \ldots, \gamma_m)$, $m \geq 3$, then the claim follows from the induction hypothesis and the fact that $\mathsf{cons}(\gamma)$ contains only red trees, concretely, $\mathsf{tree}(\gamma)$ is green, from which it follows that $\mathsf{reduce}(f(\gamma_1)) = f(\gamma_1)$.

The fact that $\mathsf{tree}(\gamma')$ is balanced follows directly from the previous property and the condition on the function d that $\mathsf{Pat}(\gamma_1) = f(\gamma_1) = f(\gamma_i) = \mathsf{Pat}(\gamma_i)$ for all $1 \leq i \leq m$.

□

Now we show how to get a sequence of trees from runs of the automaton. Let $\mathsf{treeRun} : \Sigma^+ \longrightarrow \Gamma^+$ be a homomorphism with respect to the word composition defined by $\mathsf{treeRun}(a) = \langle \mathsf{Run}(a), a \rangle$.

Assume that there is a factorization function d fixed. Let for a word $w \in \Sigma^+$, γ_w be defined as $\mathsf{cons}^n(\mathsf{treeRun}(w))$, where $n \in \mathbb{N}$ is the least such that $\mathsf{cons}^n(\mathsf{treeRun}(w)) = \mathsf{cons}^{n+1}(\mathsf{treeRun}(w))$. Note that γ_w is always defined, because for all $\gamma \in \Gamma^+$, $|\mathsf{cons}(\gamma)| \leq |\gamma|$ and if $|\mathsf{cons}(\gamma)| = |\gamma|$ then $\mathsf{cons}(\gamma) = \gamma$. Let $T_w = \mathsf{tree}(\gamma_w)$. We call T_w the *reduced factorization tree* of w. From Lemma 1 it follows that T_w is balanced (note that if $\mathsf{cons}^n(\gamma) = \mathsf{cons}^{n+1}(\gamma)$ then $\mathsf{cons}^n(\gamma)$ contains only green trees).

Remark. Notice that we do not explicitly mention the factorization function d in the definition of a reduced factorization tree T_w constructed by d from a word w. It is always clear from the context which factorization function we mean.

To prove that the height of the reduced factorization trees is bounded for a given automaton, we need to show a technical property of the reduction function, namely that reduction strictly reduces the $\mathcal{J}$ level of the pattern ($\mathcal{J}$ is a standard Green's relation; $\sigma \leq_{\mathcal{J}} \sigma'$ if and only if there are σ_1, σ_2 such that $\sigma = \sigma_1 \bullet \sigma' \bullet \sigma_2$; $\sigma <_{\mathcal{J}} \sigma'$ if and only if $\sigma \leq_{\mathcal{J}} \sigma'$ and $\sigma' \not\leq_{\mathcal{J}} \sigma$).

Lemma 2. *For any idempotent pattern σ, either* $\mathsf{reduce}(\sigma) = \sigma$ *or* $\mathsf{reduce}(\sigma) <_{\mathcal{J}} \sigma$.

Proof. From the idempotence of σ it follows that $\mathsf{reduce}(\sigma) = \sigma \bullet \mathsf{reduce}(\sigma) \bullet \sigma$. This property is sufficient for the proof of Lemma 3 from [Sim94] which applies to our case. This proof uses Green's relations.

We present also an alternative proof without using Green's relations here.

First we show that if $\mathsf{reduce}(\sigma) \neq \sigma$ then there are t and s such that $t \in \sigma(s,s)$ but $t \notin \mathsf{reduce}(\sigma)(s,s)$. Assume that it is not the case. Because σ is idempotent and the function reduce does not add anything to the pattern, there are s, s', t such that $t \in \sigma(s,s')$, $t \notin \mathsf{reduce}(\sigma)(s,s')$. Because σ is idempotent, there are s'', t_1, t_2, t_3 such that $t_1 \in \sigma(s,s''), t_2 \in \sigma(s'',s''), t_3 \in \sigma(s'',s'), t = t_1 \circ t_2 \circ t_3$. From the assumption, $t_2 \in \mathsf{reduce}(\sigma)(s'',s'')$, i.e., there are $\hat{s}, t', t'', t'''$ such that $t' \in \sigma(s'',\hat{s}), t'' \in \mathsf{core}(\sigma)(\hat{s},\hat{s}), t''' \in \sigma(\hat{s},s''), t_2 = t' \circ t'' \circ t'''$. But because σ is idempotent, $t_1 \circ t' \in \sigma(s,\hat{s})$ and $t''' \circ t_3 \in \sigma(\hat{s},s')$, so $t \in \mathsf{reduce}(\sigma)(s,s')$, which is a contradiction with the assumption.

Let us say that s and s' are merged by t in σ if $t \in \sigma(s,s), t \in \sigma(s',s'), t \in \sigma(s,s'), t \in \sigma(s',s)$. We write it $(s,t) \sim_m (s',t)$. In fact, for an idempotent pattern σ, the relation $\sim_m$ is an equivalence relation on the set of pairs (s,t). Note that if s, s' are merged by t in σ and $t \notin \mathsf{reduce}(\sigma)(s, s)$ then $t \notin \mathsf{reduce}(\sigma)(s', s')$. Therefore, the number of $\sim_m$ equivalence classes of $\mathsf{reduce}(\sigma)$ is strictly smaller than that of σ (unless they are equal).

Let $0 < 1 < r$. Let $t = (b_1, \ldots, b_n) < t' = (b'_1, \ldots, b'_n)$ if $b_i < b'_i$ for all $1 \leq i \leq n$. The set of effects together with this order is a finite lattice. Let $\downarrow t$ denote a principal ideal in this lattice generated by t. We try to construct σ_1, σ_2 so that $\sigma' = \sigma$, where $\sigma' = \sigma_1 \bullet \mathsf{reduce}(\sigma) \bullet \sigma_2$, and we show that if we do not want to fail then $\mathsf{reduce}(\sigma) = \sigma$.

Let us say that s, t where $t \in \sigma'(s, s)$ goes through s', t' if there are t_1, t_2, t_3, t_4, t_5 such that $t_1 \in \sigma_1(s, s_1), t_2 \in \sigma(s_1, s'), t_3 \in \mathsf{core}(\sigma)(s', s'), t_4 \in \sigma(s', s_2), t_5 \in \sigma_2(s_2, s)$, $t_3 < t'$, and $t' \in \sigma(s', s')$. The main idea of the rest of this proof is that to be able to construct i different equivalence classes wrt. $\sim_m$, we need i different equivalence classes in $\mathsf{reduce}(\sigma)$. We will be interested only in the effects on the loops, i.e., only in $t \in \sigma'(s, s')$ where $s = s'$.

Note that if σ' is idempotent (and we want this, because σ is idempotent) then if s_1, t_1, s_2, t_2 go through s_3, t', s_4, t', respectively, and $(s_3, t') \sim_m (s_4, t')$ in σ then $(s_1, t_1 \vee t_2) \sim_m (s_2, t_1 \vee t_2)$ in σ'. This follows from the idempotency of σ' and the definition of the relation *merged*; the reasoning is similar to the one in the first paragraph of this proof.

We show by induction on the size of $\downarrow t$ that if $t \in \sigma(s, s)$ for some s then we need as many equivalence classes which contain a $t' \in\downarrow t$ in their second component in $\mathsf{reduce}(\sigma)$ as in σ to not to introduce any $t \in \sigma(s, s')$ such that $t \notin \sigma(s, s')$. The basic step is clear from the previous paragraph. For the induction step, if s, t goes through some s', t' such that $t' < t$ then $t \in \mathsf{reduce}(\sigma)(s', s')$ must hold and thus it also goes through s', t. Also, each s, t, s', t which are not merged in σ have to go through s_1, t, s_2, t which are not merged in σ. Therefore, there are needed as many equivalence classes which contain t in their second component as there are in σ.

□

We state the factorization forest theorem. It was formulated and proved by Simon [Sim90], the best known bound is shown in [Col07].

Theorem 2 (Factorization Forest Theorem). *For a finite monoid $\mathbb{P}$ and a homomorphism $f : \Gamma^+ \longrightarrow \mathbb{P}$, there is a factorization function d such that for all $\gamma \in \Gamma^+$, $h(\mathsf{tree}(\gamma)) \leq 3 \cdot |\mathbb{P}|$.*

We show that for each R-automaton there is a factorization function such that for any w the height of the tree T_w is bounded by a constant computed from the parameters of the automaton.

Lemma 3. *Given an R-automaton A, there is a factorization function d such that for all words $w \in \Sigma^+$, $h(T_w) \leq 3 \cdot |\mathbb{P}|^2$.*

Proof. Let us first define the nesting depth function $\mathsf{nd} : \Gamma^+ \to \mathbb{N}$ by

$$\mathsf{nd}(\gamma) = \begin{cases} 1 & \text{if } \gamma = \langle \sigma, a \rangle \\ 1 + \mathsf{nd}(\gamma') & \text{if } |\gamma| = 1, \\ & \gamma \neq \langle \sigma, a \rangle, \\ & \gamma = \mathsf{tree}(\gamma') \\ \max\{\mathsf{nd}(T_i), \ldots, \mathsf{nd}(T_k)\} & \text{if } \gamma = T_1 \cdots T_k \end{cases}$$

Note that for any $w \in \Sigma^+$ and for any tree in γ_w, either the tree consists of only a root (it is equal to $\langle \sigma, a \rangle$ for some σ and a) or it has been obtained as $\mathsf{tree}(\gamma')$ for some $\gamma' \in \Gamma^+$. Note also, that for each such tree, there is exactly one such γ' (for a fixed d). Therefore, the nesting depth function nd is well-defined for all γ_w.

From Lemma 2 it follows that whenever nd is applied to a γ such that $|\gamma| = 1$, $\gamma \neq \langle \sigma, a \rangle$, $\gamma = \mathsf{tree}(\gamma')$, $\gamma' = T_1 \cdots T_k$ then for all $1 \leq i \leq k$, $\mathsf{Pat}(\gamma) <_{\mathcal{J}} \mathsf{Pat}(T_i)$. Thus, for any $w \in \Sigma$, $\mathsf{nd}(\gamma_w) \leq |\mathbb{P}|$.

From Theorem 2, we know that there is d such that $h(\mathsf{tree}(\gamma)) \leq \max\{h(T_1), \ldots, h(T_k)\} + 3 \cdot |\mathbb{P}|$ for all sequences $\gamma = T_1 \cdots T_k$. Therefore, $h(T_w) = h(\mathsf{tree}((\gamma_w))$, $h(\mathsf{tree}((\gamma_w)) \leq 3 \cdot |\mathbb{P}| \cdot \mathsf{nd}(\gamma_w) \leq 3 \cdot |\mathbb{P}|^2$ for this d.

□

3.3 Correctness

To formulate the first correctness lemma, we define the following concept of a length function $l : \Gamma \to \mathbb{N}$ inductively by

$$l(T) = \begin{cases} 1 & \text{if } T \text{ is a leaf} \\ l(T_1) + l(T_2) & \text{if } T \text{ has two children } T_1, T_2 \\ 2 \cdot \max\{l(T_1), \ldots, l(T_m)\} & \text{if } T \text{ has children } T_1, \ldots, T_m, m \geq 3 \end{cases}$$

By induction on $h(T_w)$ and using the bound derived in Lemma 3, one can show the following claim.

Lemma 4. *Given an R-automaton A, there is a factorization function d such that for all words $w \in \Sigma^+$, $l(T_w) \leq 8^{|\mathbb{P}|^2}$.*

Proof. By induction on $h(T_w)$ and using the bound derived in Lemma 3.

□

We say that $s \xrightarrow{w} s'$ or $s \xrightarrow{w}_D s'$ *realizes* t if there is a witnessing path $(s, a_1, t_1, s_1), (s_1, a_2, t_2, s_2), \ldots, (s_{|w|-1}, a_{|w|}, t_{|w|}, s')$ such that $t = t_1 \circ t_2 \circ \cdots \circ t_{|w|}$. If $s \xrightarrow{w}_D s'$ (or $s \xrightarrow{w} s'$) realizes $t = (b_1, \ldots, b_n)$, the counter values along a run $\langle s, (c_1, \ldots, c_n) \rangle \xrightarrow{w} \langle s', (c'_1, \ldots, c'_n) \rangle$ produced by this path satisfy the following conditions:

- if $b_i = 0$ then $c_i = c''_i$ for all states $\langle s'', (c''_1, \ldots, c''_n) \rangle$ along the run,
- if $b_i = r$ then $c''_i = 0$ (since it is reset) in some state $\langle s'', (c''_1, \ldots, c''_n) \rangle$ along the run, and

– if $b_i = 1$ then $c_i < c'_i$ (and it is not reset along the run).

Let us define $\mathsf{Run}_D(w)$ to be the pattern obtained by running the automaton over w in the D-semantics. Formally, $\mathsf{Run}_D(w)(s, s')$ contains t if and only if $s \xrightarrow{w}_D s'$ realizes t. Note that the function Run_D is not a homomorphism with respect to the word composition. We also define a relation $\sqsubseteq$ on patterns by $\sigma \sqsubseteq \sigma'$ if and only if for all s, s', $\sigma(s, s') \subseteq \sigma'(s, s')$.

From Lemma 4 we show that there is a factorization function such that for every w, $\mathsf{Pat}(T_w)$ corresponds to the runs of the R-automaton which can be performed in the D-semantics for any big enough D. This is formulated in the following lemma.

Lemma 5. *Given an R-automaton, there is a factorization function such that for all $w \in \Sigma^+$ and for all $D \in \mathbb{N}, D \geq 8^{|\mathbb{P}|^2}$, $\mathsf{Pat}(T_w) \sqsubseteq \mathsf{Run}_D(w)$.*

Proof. Let us fix a factorization function d satisfying Lemma 4. We show this lemma by proving the following claim by induction on $h(T_w)$. For any $w \in \Sigma^+$, if $t \in \mathsf{Pat}(T_w)(s, s')$ then $s \xrightarrow{w}_D s'$ realizing t for $D = l(T_w)$. From Lemma 4 we have that such a run exists also in any D-semantics for $D \geq 8^{|\mathbb{P}|^2}$.

The basic step follows directly from the definition of the function $\mathsf{treeRun}$.

Assume that the tree has the root $\langle \sigma_1 \bullet \sigma_2, w_1 \cdot w_2 \rangle$ with children T_{w_1} and T_{w_2} (note that for each subtree T, $T = T_{\mathsf{Word}(T)}$), where $\sigma_1 = \mathsf{Pat}(T_{w_1})$, $\sigma_2 = \mathsf{Pat}(T_{w_2})$. Then there are s'', t_1, t_2 such that $t_1 \in \sigma_1(s, s''), t_2 \in \sigma_2(s'', s')$, and $t = t_1 \circ t_2$. From the induction hypothesis, $s \xrightarrow{w_1}_{D_1} s''$ realizes t_1 and $s'' \xrightarrow{w_2}_{D_2} s'$ realizes t_2, where $D_1 = l(T_{w_1}), D_2 = l(T_{w_2})$. Clearly, if we concatenate any two paths given by these relations, we get $s \xrightarrow{w}_{D_1+D_2} s'$ realizing $t_1 \circ t_2$. From the definition of the length function, $l(T_w) = l(T_{w_1}) + l(T_{w_2}) = D_1 + D_2$.

Assume that the tree has the root $\langle \mathsf{reduce}(\sigma), w_1 \cdots w_m \rangle$ with children T_{w_1}, $\ldots, T_{w_m}$, where $m \geq 3, \sigma = \mathsf{Pat}(T_{w_1})$. Then there are s'', t_1, t_2, t_3 such that $t_1 \in \sigma(s, s''), t_2 \in \sigma(s'', s''), t_3 \in \sigma(s'', s'), t = t_1 \circ t_2 \circ t_3$, and $t_2 \in \{0, r\}^n$ (this follows directly from the definition of the function reduce). Since $\mathsf{Pat}(T_{w_i}) = \sigma$ for all $1 \leq i \leq m$ ((which we have from Lemma 1) then from the induction hypothesis $s \xrightarrow{w_1}_{l(T_{w_1})} s''$ realizes t_1, $s'' \xrightarrow{w_i}_{l(T_{w_i})} s''$ realizes t_2 for all $2 \leq i \leq m-1$, and $s'' \xrightarrow{w_m}_{l(T_{w_m})} s'$ realizes t_3.

Let us analyze the length of the concatenation of the paths given by these relations. For each counter, if its corresponding effect in t_2 is 0 then the bound on this counter during the whole path is $l(T_{w_1}) + l(T_{w_m})$, because it is left unchanged during the path part over $w_2 \cdot w_3 \ldots w_{m-1}$. If the corresponding effect in t_2 of the counter is r then the counter is reset at least once in each path part over $w_2, w_3, \cdots, w_{m-1}$. Therefore, it is bounded by the maximal length between two resets, which is bounded by $\max\{l(T_{w_1}) + l(T_{w_2}), l(T_{w_2}) + l(T_{w_3}), \ldots, l(T_{w_{m-1}}) + l(T_{w_m})\}$. Then, $s \xrightarrow{w}_D s'$ realizes t, where $D = 2 \cdot \max\{l(T_{w_1}), \ldots, l(T_{w_m})\}$. ☐

Of particular interest are runs starting in the initial state.

Corollary 1. *Given an R-automaton A, there is a factorization function such that for all words w, if $\mathsf{Pat}(T_w)(s_0, s) \neq \emptyset$ then there is a run $\langle s_0, (0, \ldots, 0)\rangle \xrightarrow{w}_D \langle s, (c_1, \ldots, c_n)\rangle$ where $D = l(T_w)$.*

It remains to show that if the relation between the patterns in the previous lemma is strict then there is a word for each D which is a witness for the strictness, i.e., the runs over this word in the D-semantics generate a smaller pattern than over the original word. These witness words are generated from a +-free regular expression r by pumping r_1 for all subexpressions r_1^* of r. Let us define a function re which for a reduced factorization tree returns a +-free regular expression inductively by

$$\mathsf{re}(T) = \begin{cases} \mathsf{Word}(T) & \text{if } T \text{ is a leaf} \\ \mathsf{re}(T_1) \cdot \mathsf{re}(T_2) & \text{if } T \text{ has two children } T_1, T_2 \\ (\mathsf{re}(T_1))^* & \text{if } T \text{ has children } T_1, T_2, \ldots, T_m, m \geq 3 \end{cases}$$

For a +-free regular expression r and a natural number $k > 0$, let the function $\mathsf{pump}(r, k)$ be defined inductively as follows: $\mathsf{pump}(a, k) = a$, $\mathsf{pump}(r_1 \cdot r_2, k) = \mathsf{pump}(r_1, k) \cdot \mathsf{pump}(r_2, k)$, and $\mathsf{pump}(r^*, k) = \mathsf{pump}(r, k)^k$.

For example, $\mathsf{pump}(a(bc^*d)^*aa^*, 2) = abccdbccdaaa$.

Lemma 6. *Given an R-automaton and a factorization function, for all $w \in \Sigma^+$ and all $D \in \mathbb{N}$ there is a $k \in \mathbb{N}$ such that $\mathsf{Run}_D(\mathsf{pump}(\mathsf{re}(T_w), k)) \sqsubseteq \mathsf{Pat}(T_w)$.*

Proof. We show this lemma by proving the following claim by induction on $h(T_w)$. For all $D \in \mathbb{N}$ there is $k \in \mathbb{N}$ such that for $v = \mathsf{pump}(\mathsf{re}(T_w), k)$, if $s \xrightarrow{v}_D s''$ realizes t then $t \in \mathsf{Pat}(T_w)(s, s')$ (note that this holds also for all $k' > k$).

The basic step follows directly from the definition of the function $\mathsf{treeRun}$ (with any k).

Assume that the tree has the root $\langle \sigma_1 \bullet \sigma_2, w_1 \cdot w_2\rangle$ with children T_{w_1} and T_{w_2}, where $\sigma_1 = \mathsf{Pat}(T_{w_1})$, $\sigma_2 = \mathsf{Pat}(T_{w_2})$. Let k_1, k_2 be the constants from the induction hypothesis applied to T_{w_1} and T_{w_2}. Let $k = \max\{k_1, k_2\}$. Let us denote $v_1 = \mathsf{pump}(\mathsf{re}(T_{w_1}), k), v_2 = \mathsf{pump}(\mathsf{re}(T_{w_2}), k), v = v_1 \cdot v_2 = \mathsf{pump}(\mathsf{re}(T_w), k)$. Assume that $s \xrightarrow{v}_D s''$ realizes t. Then there must be an s'' such that $s \xrightarrow{v_1}_D s''$, $s'' \xrightarrow{v_2}_D s'$ realize t_1, t_2, respectively, such that $t = t_1 \circ t_2$. From the induction hypothesis, $t_1 \in \mathsf{Pat}(T_{w_1})(s, s'')$ and $t_2 \in \mathsf{Pat}(T_{w_2})(s'', s')$. Because $\mathsf{Pat}(T_w) = \sigma_1 \bullet \sigma_2 = \mathsf{Pat}(T_{w_1}) \bullet \mathsf{Pat}(T_{w_2})$, we have that $t = t_1 \circ t_2 \in \mathsf{Pat}(T_w)(s, s')$.

Assume that the tree has the root $\langle \mathsf{reduce}(\sigma), w_1 \cdots w_m\rangle$ with children T_{w_1}, $\ldots, T_{w_m}$, where $m \geq 3, \sigma = \mathsf{Pat}(T_{w_1})$. Let k_1 be the constant from the induction hypothesis applied to T_{w_1} and $k_2 = (D+1)^n \cdot |S|$. Let $k = \max\{k_1, k_2\}$. Let us denote $v_1 = \mathsf{pump}(\mathsf{re}(T_{w_1}), k), v = v_1^k = \mathsf{pump}(\mathsf{re}(T_w), k)$.

Assume that $s \xrightarrow{v}_D s'$ realizes t. Then there must be a sequence of states s_i for $1 \leq i \leq k+1$ such that $s_i \xrightarrow{v_1}_D s_{i+1}$ realizes t_i, $s_1 = s, s_{k+1} = s'$, and $t = t_1 \circ t_2 \circ \cdots \circ t_k$. First, we show by contradiction that there are indices i, j such that $i < j, s_i = s_j$ and $t_i \circ \cdots \circ t_{j-1} \in \{0, r\}^n$. Let us assume that for all

$i < j$ such that $s_i = s_j$, $t_i \circ \cdots \circ t_{j-1} \notin \{0,r\}^n$. Let us pick an $\hat{s}$ such that $G = |\{i|s_i = \hat{s}, 1 \leq i \leq k+1\}|$ is maximal. From the choice of k we have that $G > D^n$. We show that there is a counter exceeding D along all paths witnessing $s \xrightarrow{v} s'$ realizing t. We know from our assumption ($t_i \circ \cdots \circ t_{j-1} \notin \{0,r\}^n$) and from the definition of realizing that for all i, j such that $s_i = s_j = \hat{s}$, the counter values in any run over v cannot be identical in s_i and s_j. There are D^n different configurations with all counters smaller than or equal to D. Since $G > D$, some counter has to exceed D. This contradicts that $s \xrightarrow{v}_D s'$ realizes t.

From the induction hypothesis we have that for all $1 \leq i \leq k$, $t_i \in \mathsf{Pat}(T_{w_1})$. Let i and j satisfy the condition from the previous paragraph, i.e., $i < j, s_i = s_j$ and $t_i \circ \cdots \circ t_{j-1} \in \{0,r\}^n$. Because $\mathsf{Pat}(T_{w_1})$ is idempotent (follows from Lemma 1), we have that $t_i \circ \cdots \circ t_{j-1} \in \mathsf{Pat}(T_{w_1})(s_i, s_j)$ and thus $t_i \circ \cdots \circ t_{j-1} \in \mathsf{core}(\mathsf{Pat}(T_{w_1}))(s_i, s_j)$. Also, $t_1 \circ \cdots \circ t_{i-1} \in \mathsf{Pat}(T_{w_1})(s, s_i)$ and $t_j \circ \cdots \circ t_k \in \mathsf{Pat}(T_{w_1})(s_j, s')$. From the definition of the function reduce, we can conclude that $t \in \mathsf{reduce}(\mathsf{Pat}(T_{w_1}))(s, s')$.

□

A special case are runs starting from the initial state.

Corollary 2. *Given an R-automaton, for any $w \in \Sigma^+$, if $\mathsf{Pat}(T_w)(s_0, s) = \emptyset$ then $\forall D \exists k$ such that there is no run $\langle s_0, (0, \ldots, 0)\rangle \xrightarrow{v}_D \langle s, (c_1, \ldots, c_n)\rangle$ where $v = \mathsf{pump}(\mathsf{re}(T_w), k)$.*

3.4 Algorithm

To check the universality of an R-automaton A, we have to check all patterns σ such that $\sigma = \mathsf{Pat}(T_w)$ for some $w \in \Sigma^+$ and some factorization function. If there is a σ such that for all $s_f \in F$, $\sigma(s_0, s_f) = \emptyset$ then for all $D \in \mathbb{N}$, $L_D(A) \neq \Sigma^*$. This gives us the following algorithm. Recall that σ_e denotes the unit of $(\mathbb{P}, \bullet)$.

The algorithm uses a set of patterns P as the data structure. Given an R-automaton $A = \langle S, \Sigma, \Delta, s_0, F\rangle$ on the input, it answers 'YES' or 'NO'. The set P is initialized by $P = \{\sigma | \sigma = \mathsf{Run}(a), a \in \Sigma\} \cup \{\sigma_e\}$.

While $|P|$ increases the algorithm performs the following operations:

- pick $\sigma_1, \sigma_2 \in P$ and add $\sigma_1 \bullet \sigma_2$ back to P.
- pick a $\sigma \in P$ such that σ is idempotent and add $\mathsf{reduce}(\sigma)$ back to P.

If there is $\sigma \in P$ such that for all $s_f \in F$, $\sigma(s_0, s_f) = \emptyset$, answer 'NO', otherwise, answer 'YES'.

Before we prove the correctness of the algorithm, we show that each pattern obtained by the algorithm corresponds to some word and some factorization function.

Lemma 7. *For any $\sigma \in \mathbb{P}$ obtained by the algorithm there is a factorization function and a word w such that $\sigma = \mathsf{Pat}(T_w)$.*

Proof. Consider the tree labeled by the patterns defined inductively as follows. The root is labeled by σ. If a node is labeled by σ' which was created (for the first time) by composing $\sigma_1 \bullet \sigma_2$ then this node has two children labeled by σ_1 and σ_2. If a node is labeled by σ' which was created (for the first time) by reducing σ_1 then this node has one child labeled by σ_1. The leaf labels have been added in the initialization step. Clearly, is $\sigma_1 = \sigma_2$ are labels of two nodes in the tree then their subtrees are identical.

Now we define a partial function $w : \mathbb{P} \longrightarrow \Sigma^+$ which for each pattern in the tree returns a word and if $\sigma_1 \neq \sigma_2$ then $w(\sigma_1) \neq w(\sigma_2)$. Such a labeling also defines a factorization function which for $w = w(\sigma)$ yields the tree T_w such that $\sigma = \mathsf{Pat}(T_w)$.

We start from the leaves and move inductively up. During the whole construction, we maintain a counter c, which is initially set to $c = 1$. For each σ in a leaf, $w(\sigma) = a$ such that $\mathsf{Run}(a) = \sigma$ (if there are several, we assume some ordering and pick the least one). If a node is labeled by σ' and it has two children labeled by σ_1 and σ_2 then $w(\sigma') = w(\sigma_1) \cdot w(\sigma_2)$. If a node is labeled by σ' and it has one child labeled by σ_1 then $w(\sigma') = (w(\sigma_1))^k$ such that $|\mathbb{P}|^c < |w(\sigma')| \leq 2 \cdot |\mathbb{P}|^c$ and we increment c.

For two different patterns such that at least one of them has a reduction in its subtree, the words have to have a different length . For two different patterns such that there is no reduction in their subtrees, the words have to be different because of the definition of Run and $\bullet$ (and all such words are shorter than $|\mathbb{P}|$).

□

The correctness is stated in the following theorem.

Theorem 3. *The algorithm is correct and runs in 2-EXPSPACE.*

Proof (Theorem 3). Clearly, the algorithm "checks" all possible σ's such that there is a factorization function and a word w such that $\sigma = \mathsf{Pat}(T_w)$. Also, for any σ obtained by the algorithm there is a factorization function and a word w such that $\sigma = \mathsf{Pat}(T_w)$ (Lemma 7), with the exception of σ_e which corresponds to $w = \epsilon$ (for which is the correctness clear).

If the algorithm obtains a σ such that $\sigma(s_0, s_f) = \emptyset$ for all $s_f \in F$ then let us fix a factorization function and a word w such that $\sigma = \mathsf{Pat}(T_w)$. Let $r = \mathsf{re}(T_w)$. From Corollary 2, for all D there is a k such that there is no accepting run over $\mathsf{pump}(r, k)$ in D-semantics.

If for all patterns σ, $\sigma(s_0, s_f) \neq \emptyset$ for some $s_f \in F$ then we can fix a factorization function satisfying Lemma 4. For all words, there is an accepting run in $8^{|\mathbb{P}|^2}$-semantics given by Corollary 1.

The complexity follows from the size of the monoid $\mathbb{P}$. The algorithm needs space $|\mathbb{P}|$ (the number of different patterns). The size of $\mathbb{P}$ is $2^{(3^n)\cdot|S|^2}$ ($|S|^2$ different pairs of states, $2^{(3^n)}$ different sets of effects). Therefore, the algorithm needs double exponential space.

□

4 Limitedness

The presented method can be adapted to decide the limitedness problem for R-automata, i.e., given an R-automaton A, is there a $D \in \mathbb{N}$ such that $L(A) = L_D(A)$.

Theorem 4. *For a given R-automaton A, the limitedness problem is decidable in 2-EXPSPACE.*

To decide the limitedness problem of an R-automaton A, we need to adapt the basic concepts of the method. Effects are elements of the set $\{0, 1, r, \omega\}^n$. We extend $\circ$ by defining $\omega \circ b = b \circ \omega = \omega$ for all $b \in \{0, 1, r, \omega\}$. Patterns are then functions $\sigma : (S \times S) \longrightarrow 2^{\{0,1,r,\omega\}^n}$. The definition of $\bullet$ remains the same and patterns together with $\bullet$ form a finite monoid.

For an effect t, let $\hat{t}$ denote the result of replacing 1's in t by ω's. The function core is modified as follows. For each pattern σ, $\hat{t} \in \mathsf{core}(\sigma)(s, s')$ if and only if $t \in \sigma(s, s')$ and $s = s'$. For each σ, $\mathsf{reduce}(\sigma) <_{\mathcal{J}} \sigma$, because $\mathsf{reduce}(\sigma) = \sigma \bullet \mathsf{core}(\sigma) \bullet \sigma$ (Lemma 2 in Appendix). This gives us the boundedness of the height of the reduced factorization trees constructed with the new reduction function.

It holds that $\mathsf{Pat}(T_w)(s, s') \neq \emptyset$ if and only if $s \xrightarrow{w} s'$. Moreover, Lemma 5 and Lemma 6 hold if we restrict the resulting pattern $\mathsf{Pat}(T_w)$ to $\{0, 1, r\}$ (for all s, s', we consider only $\mathsf{Pat}(T_w)(s, s') \cap \{0, 1, r\}^n$). Our proofs can be modified in a straightforward manner, since whenever an ω occurs in an effect it cannot be overwritten any time later.

The condition for concluding non-limitedness of the input R-automaton in the algorithm is changed to checking whether there is $\sigma \in P$ such that the following two conditions hold: (i) there is $s_f \in F$, $\sigma(s_0, s_f) \neq \emptyset$ and (ii) for all $s_f \in F$, $\sigma(s_0, s_f) \cap \{0, 1, r\}^n = \emptyset$.

5 Büchi Universality

The universality problem is also decidable for R-automata with Büchi acceptance conditions.

Theorem 5. *For a given R-automaton A, the question whether there is $D \in \mathbb{N}$ such that $L^{\omega}_D(A) = \Sigma^{\omega}$ is decidable in 2-EXPSPACE.*

To show this result, we need to extend patterns by accepting state information. A pattern is now a function $\sigma : S \times S \longrightarrow 2^{\{0,1\} \times \{0,1,r\}^n}$, where for s, s' and $\langle a, t \rangle \in \sigma(s, s')$, the value of a encodes whether there is a path from s to s' realizing t which meets an accepting state. For instance, $\sigma(s, s') = \{\langle 0, (0, r) \rangle, \langle 1, (1, 1) \rangle\}$ means that there are two different types of paths between s and s': they either realize $(0, r)$ but do not visit an accepting state, or realize $(1, 1)$ and visit an accepting state. We define the composition $\bullet$ by defining the composition on the accepting state: $0 \circ 0 = 0, 0 \circ 1 = 1 \circ 0 = 1 \circ 1 = 1$. The set of patterns (denote again $\mathbb{P}$) with $\bullet$ is a finite monoid. We define the function reduce in the same way as before, i.e., the accepting state information does not play any role there. Clearly, $\mathsf{reduce}(\sigma) <_{\mathcal{J}} \sigma$, so the reduced factorization trees produced by reduce have bounded height. Lemma 5 and Lemma 6 also hold, because (non)visiting an accepting state does not influence the runs in the D-semantics.

This allows us to use the same algorithm as for the finite word universality problem, except for the condition for concluding non-universality. The condition is whether there are $\sigma_1, \sigma_2 \in P$ such that σ_2 is idempotent and for all s such that $\sigma_1(s_0, s) \neq \emptyset$ the following holds. If $\langle a, t\rangle \in \sigma_2(s, s)$ then either $a = 0$ or $t \notin \{0, r\}^n$.

Proof. Let us denote the new pattern function by Pat^B and the new function which extracts a pattern from the runs in the D-semantics by Run^B_D.

Let for an R-automaton, $C = 8^{|\mathbb{P}|^2}$ and for an ω-word w, $w = w_1 \cdot w_2 \cdot w_3 \cdots$ be a split of this word such that all w_i are finite. For each w_i we define a pattern σ_{w_1} which captures the effects of the corresponding fragments of all infinite runs over w in $2 \cdot C$-semantics. The choice of $2 \cdot C$ is motivated by the reasons explained below. Let for all $1 \leq i$, σ_{w_i} be a pattern defined by $\langle a, t\rangle \in \sigma_{w_i}(s, s')$ if and only if there is an infinite run in $2 \cdot C$-semantics $\langle s_0, (0, \ldots, 0)\rangle \xrightarrow{w_1 \cdots w_{i-1}}_{2 \cdot C} \langle s, (c_1, \ldots, c_n)\rangle \xrightarrow{w_i}_{2 \cdot C} \langle s', (c'_1, \ldots, c'_n)\rangle \xrightarrow{w_{i+1} \cdots}_{2 \cdot C}$ such that the fragment $\langle s, (c_1, \ldots, c_n)\rangle \xrightarrow{w_i} \langle s', (c'_1, \ldots, c'_n)\rangle$ realizes t and $a = 1$ if and only if this fragment contains an accepting state.

Assume that for all D, the R-automaton is not Büchi universal in the D-semantics. Let w be a counterexample for $D = 2 \cdot |\mathbb{P}| \cdot C$, i.e., $w \notin L^\omega_D(A)$. Let us split $w = w_1 \cdot w_2 \cdot w_3 \cdots$ so that all w_i are finite and $\sigma_{w_i} = \sigma_{w_j}$ for all $2 \leq i, j$. Let us denote $\sigma_1 = \sigma_{w_1}$ and $\sigma_2 = \sigma_{w_2}$.

Let $l \in \mathbb{N}$ be such that σ_2^l is an idempotent ($l \leq |\mathbb{P}||$). For all s, $\sigma_2^l(s, s)$ does not contain $\langle 1, (b_1, \ldots, b_n)\rangle$, where $b_i \in \{0, r\}$. Otherwise, i.e., if there was $s, t \in \{0, r\}^n$ such that $\langle 1, t\rangle \in \sigma_2^l(s, s)$, there would be an accepting infinite run over w in the D-semantics, which would contradict the fact that $w \notin L^\omega_D(A)$. This follows from the fact that all patterns were obtained in the $2 \cdot C$-semantics and $l \leq |\mathbb{P}|$.

It is not necessary that $\mathsf{Run}^B_C(w_2) \sqsubseteq \sigma_2$, because the set of starting states for Run^B_C is S. Even if we restrict the set of starting states to $L(\sigma_2)$, denoted $\mathsf{Run}^B_C(w_2)'$, the relation $\mathsf{Run}^B_C(w_2)' \sqsubseteq \sigma_2$ does not have to hold. This is because a fragment of a run over w_2 in $2 \cdot C$-semantics could have started from a state with high counter values and Run^B_C starts from zeros. However, if we restrict the set of starting states to the states which are in $2 \cdot C$-semantics reachable after reading w_1 with counter values smaller than C, denote $\hat{\mathsf{Run}}^B_C(w_2)$, then $\hat{\mathsf{Run}}^B_C(w_2) \sqsubseteq \sigma_2$ holds, because now Run^B_C starts from zeros and is limited by C, whereas σ_2 contains all runs which start from counter values smaller than C and they are limited by $2 \cdot C$.

From Lemma 5 we know that there is a factorization function such that $\sigma_3 = \mathsf{Pat}^B(T_{w_1}) \sqsubseteq \mathsf{Run}^B_C(w_1)$ and $\sigma_4 = \mathsf{Pat}^B(T_{w_2}) \sqsubseteq \mathsf{Run}^B_C(w_2)$. Let m be such that σ_4^m is idempotent. Note that $\sigma_4^m \sqsubseteq \sigma_2^l$. We know that if $\sigma_3 \bullet \sigma_4^m(s_0, s) \neq \emptyset$ then $\sigma_4^m(s, s)$ does not contain $\langle 1, (b_1, \ldots, b_n) \rangle$, where $b_i \in \{0, r\}$. Therefore, from Lemma 6 we know that for any factorization function it holds that for all D there is a k such that $\mathsf{pump}(\mathsf{re}(T_{w_1}), k) \cdot (\mathsf{pump}(\mathsf{re}(T_{w_2}), k))^\omega \notin L^\omega_D(A)$.

□

6 Non-emptiness

The language emptiness problems can be decided much easier. The following theorem follows easily from an observation that for an R-automaton A, $[\![A]\!]$ is bisimilar to A taken as a finite automaton (without counters).

Theorem 6. *For a given R-automaton A, the question whether there is $D \in \mathbb{N}$ such that $L_D(A) \neq \emptyset$ is decidable in NLOGSPACE.*

To show the Büchi case, one has to find an accepting loop which for each counter either contains an r or contains only 0's. This information can be computed using an abstraction of the reachability information.

Theorem 7. *For a given R-automaton A, the question whether there is $D \in \mathbb{N}$ such that $L^\omega_D(A) \neq \emptyset$ is decidable in PSPACE.*

A *reachability abstraction* C is a set of the elements from $\{0, 1, r\}^n$ (a set of n-tuples of 0's, 1's, and r's). For a given R-automaton A and states s, s' of A, we use $C_{s,s'}$ to denote a reachability abstraction defined as follows: $t \in C_{s,s'}$ if and only if there is $w \in \Sigma^+$ such that $s \xrightarrow{w} s'$ realizes t.

In particular, if s' is not reachable from s then $C_{s,s'} = \emptyset$. If $t \in C_{s,s}$ and $t \in \{0, r\}^n$ then there is a loop such that for a sufficiently big D the loop can be iterated unboundedly many times. Clearly, there are only finitely many different reachability abstractions (for a fixed number of counters). We can compute them for all pairs of states of an R-automaton by dynamic programming (computing $C_{s,s'}$ with paths restricted to k steps for all pairs of states with increasing k until a fixed point is reached). Then, the non-emptiness problem for infinite words translates to checking whether there is an accepting state $s \in F$ such that $C_{s_0,s} \neq \emptyset$ and there is a $t \in C_{s,s}$ such that $t \in \{0, r\}^n$.

The complexity follows from the fact that the non-deterministic reachability procedure has to remember the value $0, 1$, or r for every counter.

7 Conclusions

We have defined R-automata – finite automata extended with unbounded counters which can be left unchanged, incremented, or reset along the transitions. As the main result, we have shown that the following problem is decidable in 2-EXPSPACE. Given an R-automaton, is there a bound such that all words are accepted by runs along which the counters do not exceed this bound? We have also extended this result to R-automata with Büchi acceptance conditions.

As a future work, one can consider the (bounded) universality or limitedness question to vector addition systems (VASS) or reset vector addition systems (R-VASS), where the latter form a superclass of R-automata. The limitedness problem can be shown undecidable for R-VASS for both finite word and ω-word case, while it is an open question for VASS. The universality problem can be shown to be undecidable for R-VASS for ω-word case, in other cases it is open.

References

[AKY07] P. A. Abdulla, P. Krcal, and W. Yi. Sampled universality of timed automata. In *Proc. of FOSSACS'07*, volume 4423 of *LNCS*, pages 2–16. Springer–Verlag, 2007.

[BC06] Mikolaj Bojańczyk and Thomas Colcombet. Bounds in omega-regularity. In *LICS'06*, pages 285–296. IEEE Computer Society Press, 2006.

[CL08] Thomas Colcombet and Christof Löding. The non-deterministic Mostowski hierarchy and distance-parity automata. In *ICALP'08*, volume 5126 of *LNCS*, pages 398–409. Springer–Verlag, 2008.

[Col07] Thomas Colcombet. Factorisation forests for infinite words. In *FCT'07*, volume 4639 of *LNCS*, pages 226–237. Springer–Verlag, 2007.

[Has82] Kosaburo Hashiguchi. Limitedness theorem on finite automata with distance functions. *Journal of Computer and System Sciences*, 24(2):233–244, 1982.

[Has90] Kosaburo Hashiguchi. Improved limitedness theorems on finite automata with distance functions. *Theoretical Computer Science*, 72(1):27–38, 1990.

[Kir05] Daniel Kirsten. Distance desert automata and the star height problem. *Informatique Theorique et Applications*, 39(3):455–509, 2005.

[KP05] Pavel Krcal and Radek Pelanek. On sampled semantics of timed systems. In *Proc. of FSTTCS'05*, volume 3821 of *LNCS*, pages 310–321. Springer–Verlag, 2005.

[Leu91] Hing Leung. Limitedness theorem on finite automata with distance functions: an algebraic proof. *Theoretical Computer Science*, 81(1):137–145, 1991.

[Sim90] Imre Simon. Factorization forests of finite height. *Theoretical Computer Science*, 72(1):65–94, 1990.

[Sim94] Imre Simon. On semigroups of matrices over the tropical semiring. *Informatique Theorique et Applications*, 28(3-4):277–294, 1994.