Universality of R-automata with Value Copying [1]

Parosh Aziz Abdulla, Pavel Krcal and Wang Yi [2]

Dept. of Information Technology,
Uppsala University, Sweden

Abstract

R-automata are finite state machines extended with counters which can be incremented or reset to zero along the transitions. The universality question asks whether there is a constant D such that all words are accepted by some run along which no counter exceeds D. It has been shown in [AKY08] that this question is decidable. In this paper, we add one more operation to R-automata, namely the operation which can copy the value of a counter into another one. The result of this paper is a reduction of the universality problem for R-automata with value copying to universality of R-automata, thus rendering the problem decidable. The reduction replaces copy operations by non-deterministic resets together with a mechanism ensuring that the number of such replacements is bounded between each two resets of a value.

Keywords: Distance automata with resets and copying, universality, decidability.

1 Introduction

Finite state machines usually serve as word acceptors, where the accepting condition is expressed in terms of the last reached state. For the acceptance of a word, it does not matter what has happened along the run, as long as we end up in an accepting state. One can also introduce a notion of the cost into finite automata by setting a price on each transition. The cost of a run is then the sum of the costs of the transitions. The cost of a word is the minimum of the costs of all accepting runs for this word. One way of bringing this concept into the automata model is to introduce counters which accumulate the cost along the runs.

A first such extension – distance automata of Hashiguchi [Has82] – contained one counter which could only be incremented. The operations on the counter are either increment $(+1)$ or leaving the counter unchanged (0). Recently, this model was extended to several counters which can be also reset (r) – nested distance desert automata [Kir05], B-automata [BC06], and R-automata [AKY08]. In this work, we add new operations $(*j)$ where j is a counter name. These operations copy the content of the counter j into the counter to which this operation is applied.

Questions asked for these automata are concerned with the existence of bounds on the costs (i.e., counter values). The limitedness problem is to decide whether there is a bound such that all accepted words are accepted by a run along which no counter exceeds this bound. The universality problem is a special case of the limitedness problem, i.e., the question whether there is a bound such that all words are accepted by a run along which no counter exceeds this bound. This problem has been shown decidable for R-automata in [AKY08] using the factorization forest theorem [Sim90]. Here we show the decidability of the universality problem for R-automata with value copying by reducing it to the universality problem of R-automata (without copying).

The main idea of the reduction is to replace the copy instructions by resetting and swapping the values. We non-deterministically either reset the destination of the copy instruction or swap the values of the involved counters and reset the source of the copy instruction. There is a mechanism which ensures that the number of such resets is bounded along the lifetime of each value, i.e., from the beginning (a reset) of a value until its end (the next reset or overwrite by a copy). By this, we can decrease the cost only by a constant ratio.

The motivation for this particular class of counter automata comes from 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) for a pair of clocks corresponds to a counter. 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. This is done by forcing a pair of clocks to increase/decrease their difference (an increment of a counter), keep exactly the same difference (a counter is left unchanged), forget about the previous difference (a counter reset), or assume the difference of some other clock pair (a counter copy).

Related work. Different proofs of the decidability of the limitedness problem for distance automata are reported in [Has90,Leu91,Sim94]. The last of these results [Sim94] is based on the factorization forest theorem [Sim90,Col07,Kuf08]. Distance automata were extended in [Kir04,Kir05] with additional counters which can be reset following a hierarchical discipline resembling parity acceptance conditions. R-automata form a superclass of this extension. Universality of a similar type of automata for tree languages is studied in [CL08] study universality of a similar type of automata for tree languages. B-automata [BC06] contain counters which can be incremented and reset in the same way as in R-automata, but they accept infinite words such that the counters are bounded along an infinite accepting computation. The main result is that B-automata can be complemented.

2 Preliminaries

First, we introduce the model of RC-automata and its unparameterized semantics. Then we introduce the parameterized semantics, the languages accepted by the automaton, and the universality problem.

RC-automata. RC-automata are finite state machines extended with counters. A transition may increase the value of a counter, leave it unchanged, reset it back to zero, or copy it into another counter. 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. Let for a given number n of counters, $\mathcal{E} = \{0, 1, r\} \cup \{*m | 1 \leq m \leq n\}$ be the set of operations on a counter. An *RC-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, $\Delta \subseteq S \times \Sigma \times \mathcal{E} \times S$ 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, the symbol r represents a reset, and a symbol $*m$ means that the value of this counter is set to the value of the counter m. The operations $0, 1,$ and r take place first and after that the values are copied. An RC-automaton which does not contain any copy operations is called an R-automaton.

We use $t, t_1, \ldots$ to denote elements of $\mathcal{E}^n$ which we call *effects*. By $\pi_i(t)$ we denote the i-th projection of t. Without loss of generality, we assume that the value of a counter is never directly copied into itself ($\pi_i(t) \neq *i$). 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 $\forall 1 \leq i \leq m.(s_i, a_i, t_i, s_{i+1}) \in \Delta$. A state s_i will be referred to as the i-th state of the path. An example of an RC-automaton is given in Figure 1.
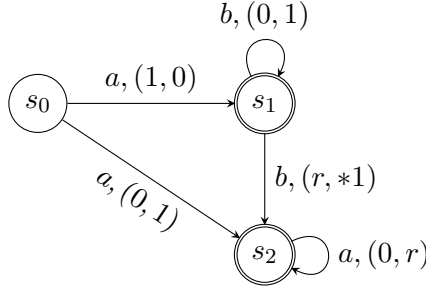

Fig. 1. An R-automaton with two counters.

Unparameterized semantics. We define an operation $\oplus$ on the counter values: 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 and copy operations as follows. For a $t \in \mathcal{E}$, let $\hat{t}$ be an effect with all copy instruction replaced by 0, i.e., $\pi_i(\hat{t}) = \pi_i(t)$ if $\pi_i(t) \in \{0, 1, r\}$ and $\pi_i(\hat{t}) = 0$ otherwise. For a $t \in \mathcal{E}$ and $(c_1, \ldots, c_n) \in \mathbb{N}^n$, $(c_1, \ldots, c_n) \oplus t = (c'_1, \ldots, c'_n)$, where $c'_i = c_m \oplus \pi_m(\hat{t})$ if $\pi_i(t) = *m$ and $c'_i = c_i \oplus \pi_i(t)$ otherwise.

The operational semantics of an RC-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 $\hat{s_0} = \langle s_0, (0, \ldots, 0) \rangle$. 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 RC-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 RC-automaton A, $[\![A]\!]$ is bisimilar to A considered as a finite automaton (without counters and effects).

Parameterized Semantics. Next, we define the D-semantics of RC-automata. The parameter D is a bound on the counter values which can occur along any run. 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 RC-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 $w \in \Sigma^+$.

Language. The (unparametrized or D-) language of an RC-automaton is the set of words which can be read along the runs in the corresponding LTS ending in an accepting node (a node whose first component is an accepting state). Formally, for a run ρ in $[\![A]\!]$, let $l(\rho)$ denote the concatenation of the labels along this run. A run $\rho = \langle s_0, (0, \ldots, 0)\rangle \longrightarrow^* \langle s, (c_1, \ldots, c_n)\rangle$ is accepting if $s \in F$. The *unparametrized language* accepted by an RC-automaton A is $L(A) = \{l(\rho)|\rho$ is an accepting run in $[\![A]\!]\}$. For a given $D \in \mathbb{N}$, the D-*language* accepted by an RC-automaton A is $L_D(A) = \{l(\rho)|\rho$ is an accepting run in $[\![A]\!]_D\}$. The unparametrized language of the RC-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 the question about language universality of an RC-automaton A parameterized by D, i.e., is there a natural number D such that $L_D(A) = \Sigma^*$.

3 Expressiveness

We argue that the expressive power of the RC-automata is bigger than that of R-automata, where the expressivity takes into account the counter values. Let us for a given RC-automaton define the cost of a run as the maximum counter value which occurs along this run and the cost of a word w to be the minimum of the costs of its accepting runs (and $+\infty$ if the word is not accepted). We say that two RC-automata are *cost equivalent* if the cost of each word is the same for both of them. In Figure 2 we give an example of an RC-automaton such that there is no cost equivalent R-automaton.

Let us assume that there is a cost equivalent R-automaton A' with k counters. Let us consider the words of the form $a^{x_1}b^{y_1}ca^{x_2}b^{y_2}c \ldots a^{x_k}b^{y_k}$. For big enough values of $x_1, y_1, x_2, \ldots, y_k$, we can pump these values so that each accepting run over of A' over the original word can be extended to an accepting run over the pumped word (a straightforward consequence of the pumping lemma for finite state automata). Clearly, the cost of these words in A is $\max\{y_1, x_1 + y_2, x_1 + x_2 + y_3, \ldots, x_1 + \cdots + x_{k-1} + y_k, x_1 + \cdots + x_k\}$.
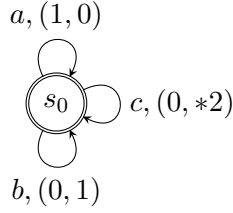
Fig. 2. An RC-automaton A which does not have a cost equivalent R-automaton.

The counters in A' have to count all of these values in each run of A' over such a word. If there is a run in which one of these values is skipped then we can pump the last summand in this value to make it the maximum. Since the control structure of the automaton (the states) cannot distinguish between different results of the pumping, this maximum will be missed by A'. We claim that for each of these values we need a special counter, contradicting thus the assumption that A' has k counters. If a counter calculates $x_1 + \cdots + x_{i-1} + y_i$ then it cannot be reused for any later value, since there is no way how could this counter learn about the value of x_i (the corresponding part a^{x_i} has already been read). Thus, by pumping the appropriate subword, we can always show that the cost of the pumped word accepted by A' is not equal to the cost of this word in A.

4   Universality

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

Theorem 4.1 *For a given RC-automaton A, the question whether there is $D \in \mathbb{N}$ such that $L_D(A) = \Sigma^*$ is decidable.*

In order to avoid unnecessary technical complications in the main part of the proof, we restrict ourselves to RC-automata with at most one copy operation in each effect. We show how to extend the proof to the general model at the end of this section. We reduce the universality problem for (restricted) RC-automata to the universality problem of R-automata, for which it has been shown that this problem is decidable [AKY08].

4.1   Construction

As the first step, we extend each R-automaton with a variable called *parent pointer* for each counter and with the ability to swap the values of the counters. The parent pointers range over $\{\mathsf{null}\} \cup \{1, \ldots, n\}$, where n is the number of the counters. We observe that for each R-automaton one can encode the value swapping and the parent pointers into the states. To express the encoding more formally, let us assume that the transitions in the semantics LTS are labeled also by the counter values (in the order encoded by the automaton) and the parent pointers. For each extended R-automaton A, we can build an R-automaton A with $|S| \cdot n! \cdot 2^n$ states bisimilar to A. Moreover, any number of value swaps and parent pointer operations can be encoded along each transition of A' together with standard updates (increments, resets). A' can also branch upon the values of the parent pointers.

Now we can present the reduction by constructing an extended R-automaton A' (which uses counter value swapping and the parent pointers) for each RC-automaton A such that A' is universal if and only if A is universal. A' has all the states of A together with an error sink (a non-accepting state with no outgoing transitions except for the self-loops labeled by Σ and effects $(0, \ldots, 0)$), it has the same initial state and the same set of accepting states as A. To define the transitions, we need to encode the copying using the tools of extended R-automata. To do this, we replace each copy by a reset, possibly with some (non-deterministic) value swapping and bookkeeping of the parent pointers. The parent pointers will help us to check whether all the non-deterministic choices were done correctly.

For each transition of A we construct the simulating transitions and also possibly some transitions going to the error sink. Let us denote the simulated transition of A by $s \xrightarrow{a,t} s'$, where $t = (e_1, \ldots, e_n)$, and the constructed simulating transition of A' by $s \xrightarrow{a,t',sp} s'$ labeled by an effect $t' = (e_1', \ldots, e_n')$, possibly swapping some counter values and manipulating the parent pointers (denoted by sp).

If t contains a copy operation then there are two simulating transitions for one simulated transition giving the simulating automaton a non-deterministic choice. We describe several steps which have to be performed to simulate the effect t in this case. The purpose of talking about several sequentially applied steps is to set the priority on competing rules, but all these steps can be performed along one simulating transition.

First, we take care of copying. Let $e_i = *j$ (because of our restriction, there is at most one such i). We check whether there is a counter with the parent pointer set to i. If it is the case, we move to the error sink. Otherwise, we create two transitions corresponding to the non-deterministic choice of one of the numbers i and j. For the choice of j, we set $e_i' = r$ (the i-th counter is reset), $e_j' = e_j$, and we set the i-th parent pointer to j. For the choice of j, we swap the values of the counters i and j, we copy the value of j's parent pointer into i's parent pointer, we change the value of all parent pointers with value j to i, we set $e_j' = r$ (the j-th counter is reset), $e_i' = e_j$, and finally we set the j's parent pointer to i.

Secondly, for all i such that $e_i = r$, we set $e_i' = r$ and the i-th parent pointer is set to null. Then we check for all these indices i whether there is a counter with the parent pointer set to i. If it is the case, we move to the error sink. For all other indices i, i.e., $e_i \in \{0, 1\}$ and $i \neq j$, we set $e_i' = e_i$.

If t does not contain any copy instruction we set $t' = t$ and for each reset perform the same bookkeeping as in the previous case. For all i such that $e_i = r$, we set the i-th parent pointer to null. Then we check for all these indices i whether there is a counter with the parent pointer set to i. If it is the case, we move to the error sink.

4.2   Proof of Correctness

Intuitively, the choice of the counter in the copy instruction tells that the value in this counter will be destroyed by a reset or overwritten by a copy instruction later than in the counter which was not chosen. The structure of the copies is captured by the parent pointers in the following sense. If the counter i points to the counter j then i contains an immediate copy of j (but possibly modified by increments) and

its value will be destroyed earlier than the value in j. The automaton ends in the error sink if it witnesses a violation of some of these implicit claims, i.e., the value in the counter i is destroyed earlier than the value in the counter j.

First, we formalize the concept of the evolution of a value and define the corresponding runs. Then we show existence of corresponding accepting runs. Later on we use the fact that the parent pointers along the simulating traces have a special structure to show the correctness of the simulation.

Definition 4.2 For a path p of length $|p|$ and two natural numbers $1 \leq i < j \leq |p|$, a total function $vt : \{i, i+1, \ldots, j\} \longrightarrow \{1, \ldots, n\}$ is a *value trace* if for all k such that $i \leq k < j$, t is the effect on the transition between the k-th and $k+1$-st state on p, $vt(k) = a$, $vt(k+1) = b$, the following holds: if $a \neq b$ then $\pi_b(t) = *a$, if $a = b$ then $\pi_b(t) \in \{0, 1, r\}$, and if $a = b$ and $k < j - 1$ then $\pi_b(t) \in \{0, 1\}$.

We also talk about a value trace along a run. Then we mean a value trace along a path which has induced the run. We order value traces by the set inclusion on their domains (e.g., $vt : \{2, 3\} \longrightarrow \{1, \ldots, n\}$ is smaller than $vt : \{2, 3, 4\} \longrightarrow \{1, \ldots, n\}$ regardless of the actual function values). We define the *length* of a value trace as the size of its domain.

A value trace follows a value from some time point during its evolution (increments, copying) in the RC-automaton. A value trace ends before the value is overwritten by a copy operation or at the moment when the value is reset. The distinction between the destruction of a value by either rewriting or resetting is related to the order in which we check the parent pointers in the construction of the simulating transitions.

Now we define the correspondence between accepting runs in an RC-automaton A and in its corresponding R-automaton A'. We say that a run ρ of A over w and a run ρ' of A' over w are corresponding if for all i the i-th transitions of ρ, ρ' are obtained by executing the transitions $s \xrightarrow{a,t} s'$ and $s \xrightarrow{a,t',sp} s'$, where $s \xrightarrow{a,t',sp} s'$ is a simulating transition of $s \xrightarrow{a,t} s'$. We show that for each accepting run of one automaton there is an accepting corresponding run of the other automaton. It follows immediately from the definitions that for each accepting run of A' there is exactly one accepting corresponding run of A.

The other direction is more complicated, because we have to show that A' can choose correct values for non-deterministic choices in the copy instruction so that it does not end up in the error sink. For each accepting run ρ of A, we construct an accepting run ρ' of A' as follows. We label each counter j in the k-th state of ρ (for all $k \leq |\rho|$) by the length of a maximal value trace vt with domain being a subset of $[k, |\rho|]$ and $vt(k) = j$ (this label is called *expectancy*). A' takes the simulating transition for each transition of ρ (according to the rules above) and when it has to choose between i and j ($e_i = *j$) along a transition ending in the k-th state, then it chooses i if and only if the expectancy of i in k is greater than the expectancy of j in k *(expectancy rule)*. We show that this is a valid definition, i.e., the corresponding run of A' does not end up in the error sink. The main step in the proof is to show that the parent pointers always point to the counters with not smaller expectancy than that of the counter which owns the parent pointer.

Lemma 4.3 *For each accepting run ρ of A there is an accepting corresponding run ρ' of A'.*

Proof. We prove by induction that for each prefix of ρ there is a simulating run which does not contain the error state such that for any state along ρ' and any two counters i, j in this state, if the parent pointer of i points to j then the expectancy of j is not smaller than that of i. Such a simulating run for $|\rho|$ will also be accepting.

The basic step (i.e., the prefix length is 0) is trivial. For the induction step, let us assume that there is a simulation of the prefix of length k satisfying the induction hypothesis. To simulate the $k+1$-st transition, we follow the expectancy rule.

The copy instruction does not lead into to the error sink, because of the induction hypothesis. The non-deterministic choice is performed according to the vt function, so the result again satisfies the induction hypothesis. The transfer of the parent pointers does not violate it either, because expectancy of j in k is equal to 1 plus the maximum of the expectancies of i and j in $k+1$. The resets do not bring us to the error sink, because of the induction hypothesis. No new parent pointers are established, so the result again satisfies the induction hypothesis. The other effects result in decrementing the expectancy, which preserves the induction hypothesis for all the pointers inherited from the previous state as well as for the pointers changed by the copy instruction. □

Let us introduce the parent pointer relation $\rightarrow_p$ for a state of A' as a relation on counters where $i \rightarrow_p j$ if and only if the parent pointer of i is set to j.

Lemma 4.4 *Let ρ be an accepting run of A and ρ' be a corresponding accepting run of A'. Then the transitive closure of $\rightarrow_p$ is antireflexive in all the states of ρ'.*

Proof. We prove by induction that for each prefix of ρ there is a simulating run which does not contain the error state such that for any state along ρ', the transitive closure of $\rightarrow_p$ is antireflexive. Such a simulating run for $|\rho|$ will also be accepting.

The basic step is trivial, $\rightarrow_p$ is empty. For the induction step, we need to check that one simulation step does not violate the antireflexivity. The resets make $\rightarrow_p$ smaller and $0, 1$ do not change it. In the copy instruction, we introduce one new pointer, but we know that nothing points to i. In the first case, we set i's parent pointer to j, which cannot introduce a loop, since nothing points to i. In the second case, since we have redirected all the pointers pointing to j to i, there is nothing pointing to j and newly introduced $j \rightarrow_p i$ cannot create a loop. Also, since there was nothing pointing to i previously, the only pointers pointing to i now are those that previously pointed to j. □

This leads to the following definition of ranks. For a counter i in a state of A' we define $\mathsf{rank}(i)$ inductively by $\mathsf{rank}(i) = 0$ if the parent pointer of i is null and $\mathsf{rank}(i) = \mathsf{rank}(j) + 1$ if $i \rightarrow_p j$. From Lemma 4.4, we have that the ranks are well-defined and it follows directly from the definition that the rank of a counter is always bounded by the number of the counters. Now we can formulate a lemma saying that the ranks never increase along a value trace.

Lemma 4.5 *Let ρ be an accepting run of A, ρ' be a corresponding accepting run of A' and vt be a value trace. Then for $k \leq l$, $\mathsf{rank}(vt(k)) \leq \mathsf{rank}(vt(l))$.*

Proof. By induction on $l - k$. The basic step: $l = k$ and $\mathsf{rank}(vt(k)) = \mathsf{rank}(vt(l))$.

The induction step: because of the checks for the reset, we never increase any rank by a reset. $0, 1$ also do not increase any rank. Copy decreases the rank of the "shorter" (with reset value) branch, keeps the rank for the "longer" one (the one which keeps the value). Because of the careful manipulation with the pointers, also no ranks which depend on the rank of the longer branch change. □

The main property of the reduction is stated in the following lemma.

Lemma 4.6 *Let A be a restricted RC-automaton with n counters and A' be the simulating R-automaton. For each D and for each word w, $w \in L_D(A) \Rightarrow w \in L_D(A')$ and $w \in L_{n \cdot D}(A) \Leftarrow w \in L_D(A')$.*

Proof. "$\Rightarrow$": from Lemma 4.3 we know that for each accepting run ρ of A over w there is a corresponding accepting run ρ' of A' over w. It follows directly from the construction that for all $k \leq |\rho|$, the counter values in the k-th state of ρ' are bounded by the counter values in the k-th state of ρ. All the operations are simulated faithfully, except for possibly more resets (in the copy operations) along ρ'.

"$\Leftarrow$": by contraposition, let us for each D consider a word w such that $w \notin L_{n \cdot D}(A)$. Any accepting run ρ' of A' over w must satisfy Lemma 4.5 (because it corresponds to an accepting run ρ of A over w). Let vt be a maximal value trace for a value which exceeds $n \cdot D$ in ρ. We study the evolution of this value in ρ'. It is simulated faithfully except for some possible resets in the copy instructions. But for each such reset, the rank of the counter strictly decreases. Therefore, there can be at most $n - 1$ such resets and there must be a state in which this value exceeds D. □

An immediate consequence of this lemma is the correctness of the reduction.

Corollary 4.7 *Let A be a restricted RC-automaton and A' be the simulating R-automaton. There is a D such that $L_D(A) = \Sigma^*$ if and only if there is a D' such that $L_{D'}(A') = \Sigma^*$.*

Now we show that the result holds also for RC-automata with any number of copying in each step. Let us view the relation "i is copied to j" induced by one effect as a directed graph (counters are nodes, there is an edge from i to j). Because each node can have at most one incoming edge, such a graph is a collection of simple loops with isolated paths outgoing from them (nodes with no incoming edge are considered as degenerated loops). We can split application of such an effect into several steps as follows. First, we perform all increments. Then we pick one of the counters j such that j has no outgoing edge and it has an (exactly one) incoming edge from i. We copy the value of i to j and leave all other counters unchanged, which can be described by the effect $(0, \ldots, *i, \ldots, 0)$, where $*i$ is on the j-th position. Then we remove the edge connecting i and j and continue to pick another such counter. When there is no node j with no outgoing edge and with an incoming edge, there still might be loops in the copying graph. We simply swap the counter values and the parent pointers in the loops. Then we finally perform the resets in the effect, together with checking and updating the parent pointers. Because of the order in which we have copied the counters, the effect of the sequence of transitions with at

most one copy instruction is the same as that of the original transition. Also, the correctness does not depend on the order in which we choose the edges. A careful analysis shows that this sequence of transitions can be encoded into one simulating transition in R-automata with value swapping and parent pointers.

5 Conclusions

We have defined RC-automata as an extension of R-automata – finite state machines with counters which can be incremented or reset to zero along the transitions – by the ability to copy the value of a counter into another counter. The universality problem for these automata is to decide whether there is a constant D such that all words are accepted by some run along which no counter exceeds D. We have shown that this problem is decidable for RC-automata by reducing it to the universality problem for R-automata, show decidable in [AKY08].

References

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

[AKY08] Parosh Abdulla, Pavel Krcal, and Wang Yi. R-automata. In *Proc. of CONCUR'08, Toronto, Canada.*, LNCS. Springer–Verlag, 2008. To appear.

[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, 2008.

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

[Kir04] Daniel Kirsten. Distance desert automata and the star height one problem. In *FoSSaCS'04*, volume 2987 of *LNCS*, pages 257–272. Springer–Verlag, 2004.

[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.

[Kuf08] Manfred Kufleitner. The height of factorization forests. In *Proc. of MFCS'08, Toruń, Poland.*, LNCS. Springer–Verlag, 2008. To appear.

[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.