ALESHIN TYPE AUTOMATA WITH REGULAR LANGUAGES

[1]A.Jeyanthi*, [2] B.Stalin

[1]Faculty, Department of Mathematics, Anna University, Regional Office Madurai, Tamilnadu, India.

[2]Assistant Professor, Department of Mechanical Engineering, Anna University, Regional Office Madurai, Tamilnadu, India.

[1] jeyanthi2009@rediffmail.com, [2]stalin1312@gmail.com

Abstract:

In this paper, the investigation of reversibility in computational devices is complemented by the study of reversible Aleshin type automata. These are deterministic Aleshin type automata that are also backward deterministic. All regular languages as well as some non-regular languages are accepted by reversible deterministic Aleshin type automata. Thus, the computational capacity of reversible Aleshin type automata lies properly in between the regular and deterministic context-free languages. The closure properties and decidability questions of the language class are investigated. It turns out that the closure properties of reversible Aleshin type automata are similar to those of deterministic Aleshin type automata.

Keywords: Deterministic Aleshin type automata, reversible Aleshin type automata.

1. Introduction

Moreover, every deterministic context-free language which needs more than realtime is shown not to be acceptable by reversible Aleshin type automata. In the second part of the paper, closure properties and decidability questions of the language class are investigated. It turns out that the closure properties of reversible Aleshin type automata are similar to those of deterministic Aleshin type automata. The main difference is the somehow interesting result that the language class accepted by reversible Aleshin type automata is not closed under union and intersection with regular languages. Finally, the questions of whether a given automaton is a reversible Aleshin type automaton, and whether its language is reversible are investigated. We show that the problem to decide whether a given nondeterministic or deterministic Aleshin type automaton is reversible is P-complete, whereas it is undecidable whether the language accepted by a given nondeterministic Aleshin type automaton is reversible.

2. Closure properties

In this section, we study the closure properties of REV-AAs. It turned out that REV-AAs and DAAs have similar closure properties, but the former are interestingly not closed under union and intersection with regular languages.

Theorem 1. L (REV-AA) is closed under complementation.

Proof. The closure under complementation for deterministic finite automata can be easily shown by interchanging accepting and rejecting states. This idea cannot be translated directly to DAAs, since mainly two problems may occur. First, the given DAA may not read its input completely by entering a configuration in which no next move is defined or an infinite λ-loop is entered. Second, the given DAA may perform λ-steps leading from an accepting state to a rejecting state and back. By Theorem 5 we may assume that a given REV-AA M works in realtime and thus has no λ-transitions. To overcome with the above-mentioned problems we then only have to ensure that in every configuration a next move is defined. This can be realized with the usual construction of adding a rejecting sink state which cannot be left once entered. Moreover, transitions being undefined so far are added and lead to the sink state. To maintain the reversibility of M we push the predecessor state of the sink state with a special marking onto the stack store. Being in the sink state we push all symbols read onto the stack store. In this way, we can identify the moment in which the sink state has been entered and, thus, can leave the sink state in the backward computation. With this modification we can construct from M a REV-AA M′ by interchanging accepting and rejecting states. Then, M′ accepts the complement of L (M) and we obtain the closure under complementation.

Next, we consider the operations intersection and union with regular languages and first give another example which enables us to show the non-closure under both operations.

Example 1. The language $\{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$ is accepted by the REV-AA $M = \langle \{q_0,q_1\}, \{a,b\}, \{A, A', B, B', \perp\}, \delta, q_0, \perp, \{q_0\}\rangle$ where the transition functions δ and δ_R are as follows.

Transition function δ

(1) $\delta(q_0, a, \perp) = (q_1, A' \perp)$
(2) $\delta(q_0, b, \perp) = (q_1, B' \perp)$
(3) $\delta(q_1, a, A') = (q_1, A A')$
(4) $\delta(q_1, a, A) = (q_1, A A)$
(5) $\delta(q_1, b, A) = (q_1, \lambda)$
(6) $\delta(q_1, b, A') = (q_0, \lambda)$
(7) $\delta(q_1, b, B') = (q_1, B B')$
(8) $\delta(q_1, b, B) = (q_1, B B)$
(9) $\delta(q_1, a, B) = (q_1, \lambda)$
(10) $\delta(q_1, a, B') = (q_0, \lambda)$

The idea of the construction is as follows. We use the stack for counting the difference between the number of a's and b's in the input. A's on the stack indicate that there are more a's than b's in the input read so far and B 's on the stack denote the opposite. Additionally, the stack symbols A′ and B′ are used to denote that the difference is one. Now, transitions (1) and (2) are used to count the difference one. Transitions (3), (4) and (7), (8) increase the difference by one and transitions (5) and (9) decrease the difference by one. Finally, if the difference is one then transitions (6) and (10) can be used to decrease the difference to zero and to enter an accepting state.

Reverse transition function δ_R

(1) $\delta_R(q_0, a, \perp) = (q_1, B'\perp)$

(2) $\delta_R(q_0, b, \perp) = (q_1, A'\perp)$

(3) $\delta_R(q_1, b, A') = (q_1, AA')$

(4) $\delta_R(q_1, b, A) = (q_1, AA)$

(5) $\delta_R(q_1, a, A) = (q_1, \lambda)$

(6) $\delta_R(q_1, a, A') = (q_0, \lambda)$

(7) $\delta_R(q_1, a, B') = (q_1, BB')$

(8) $\delta_R(q_1, a, B) = (q_1, BB)$

(9) $\delta_R(q_1, b, B) = (q_1, \lambda)$

(10) $\delta_R(q_1, b, B') = (q_0, \lambda)$

For the backward computation we just have to do the opposite by switching the roles of a and b. For example, transition (4) of δ increases the difference by one when an a is read and some A is on the stack, that is, there have been more a's than b's read so far. This difference is later decreased by one with transition (5) when a b is read. Thus, for δ_R we have to increase the difference when reading a b (transition (4)) and to decrease the difference when reading an a (transition (5)). The remaining pairs (1) and (2), (3) and (6), (7) and (10), and (8) and (9) can be translated similarly.

Theorem 2. L (REV-AA) is not closed under union and intersection with regular languages.

Proof. Due to Example 1 we know that $L = \{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$ can be accepted by some REV-AA. We assume that L (REV-AA) is closed under intersection with regular languages. Then, $L \cap a^*b^*$ belongs to L (REV-AA) as well. Since $L \cap a^*b^* = \{a^nb^n \mid n \geq 0\}$, we obtain a contradiction to Lemma 6. If the language class L(REV-AA) is closed under union with regular languages, then L(REV-AA) is also closed under intersection with regular languages due to the closure under complementation of regular languages and L (REV-AA) by Theorem 1. This is again a contradiction.

Corollary 1. L(REV-AA) is not closed under union and intersection.

On the other hand, we obtain the closure under intersection and union with regular languages under the condition that the regular language can be accepted by a reversible deterministic finite automaton. In [14] reversibility in finite automata is defined as the property of having only deterministic forward and backward computations. Additionally, the automata may possess several initial and accepting states. Here, we define a regular language as reversible if it is accepted by some reversible deterministic finite automaton which possesses one initial state only and obtain a proper subclass of the class defined in [14].

Theorem 3. L(REV-AA) is closed under union and intersection with reversible regular languages.

Proof. Let M be a REV-AA which may be assumed to work in realtime by for every REV-AA an equivalent realtime REV-AA can effectively be constructed. Let A be a reversible deterministic finite automaton with one initial state. Then, a REV-AA M′ accepting L (M) ∩ L (A) can be obtained using the classical construction. We define the state set of M′ as the Cartesian product of the state sets of M and A. In the forward computation we update the current state according to M′s and A′s transition function. The input is accepted if both states enter an accepting state of M and A. In the backward computation we update the current state according to M′s and A′s reverse transition function, since both automata are reversible and eventually enter the initial states of M and A. The construction for L (M) ∪ L (A) is identical apart from the fact that now a state of M′is accepting if at least one component is accepting.

Corollary 2. In this context the question may arise whether the union or intersection of a non-regular language from L(REV-AA) with a non-reversible regular language is always a non-reversible language. The following example shows that there are cases which lead to REV-AAs although the regular language is not reversible. We consider the union of the languages $\{a^n cb^n \mid n \geq 0\}$ and $a^* b^*$, where the latter is shown not to be reversible in [14]. A REV-AA which accepts the union works as follows. Basically, we take the construction for the language $\{a^n cb^n \mid n \geq 0\}$ where there is an a-loop on the initial state q_0 pushing symbols A onto the stack store. When reading a c we change to a state in which the b's in the input are matched with the A′ s on the stack store. If we read a b being in state q_0 , we enter a new accepting state q and write a new symbol \$ onto the stack store. From state q we can only read b's and push another new symbol # onto the stack store for every b. In this way we can accept the union of both languages. To give evidence for reversibility we just have to observe that the step in the backward computation in which we have to reenter q_0 from q can be restored by the information stored onto the stack store.

Theorem 4. L(REV-AA) is not closed under concatenation, Kleene star, λ-free homomorphism, and reversal.

Proof. The non-closure under the operations can be shown similar to the proofs for deterministic context-free languages given in [3]. One just has to modify the languages used to be reversible. We first consider the languages $L_1 = \{a^n cb^n de^m \mid n,m \geq 0\}$ and $L_2 = \{a^n cb^m de^m \mid n,m \geq 0\}$ which are both in L (REV-AA). It can be observed that $L_1 \cup L\ 2$ / L (REV-AA) since its complement is not context free.

To show the non-closure under concatenation we consider the languages $L_3 = \$L_1 \cup L_2$ and $\$^*$ which both belong to L(REV-AA). However, their concatenation $\$^*L_3$ does not belong to L (REV-AA). Otherwise, $L_4 = \$^*L_3 \cap \$a^*cb^*de^* = \$L_1 \cup \$L_2$ would belong to L (REV-AA) since L (REV-AA) is closed under intersection with reversible regular languages. But L_4 is not in L (REV-AA) since its complement is not context free. The non-closure under Kleene star and λ-free homomorphism can be shown as in [12]. One has to consider the languages $L_5 = \{\$\} \cup L_3$ and $L_6 = \$L_1 \cup \#L_2$ which are both in L (REV-AA). On the other hand, both $L*$ and $h(L_6)$, with h being a homomorphism which maps # to \$ and other symbols to themselves, can be shown to be not deterministic context free. Thus, they do not belong to L (REV-AA) either.

Finally, we consider the non-closure under reversal. Due to Example 1 we know that the language $\{a^n cb^n de^m \mid n,m \geq 0\} \cup \{a^n cb^m de^m \mid n,m \geq 0\}$ belongs to L (REV-AA), but its reversal is known not to be accepted by any realtime deterministic Aleshin type automaton. This shows the non-closure under reversal.

Remark1. It is worth mentioning that there are two situations in which closure results of the above-mentioned operations are obtained. The first result is that L(REV-AA) is closed under marked concatenation and marked Kleene star. The idea for showing the closure under marked concatenation is to first simulate the REV-AA for the first language. Then, when the marking symbol is read, the current state is pushed onto the stack store. This stack symbol also acts as bottom of-stack symbol for the second REV-AA which is subsequently simulated. The resulting automaton is reversible, since the computation consists of two reversible sub-computations. Additionally, the first sub-computation in the backward computation is started in the correct state due to the information stored onto the stack store. The construction for marked Kleene star is similar.

A second result one can observe is that the reversal L^R of a language $L \in L$ (REV-AA) belongs to L(REV-AA) if L is accepted by a REV-AA which has one accepting state only and in which every accepting computation ends in a configuration with empty (up to ⊥) stack store.

Theorem 5. L(REV-AA) is closed under inverse homomorphism.

Proof. Let $M = (Q, \Delta, \Gamma, \delta, q_0, \perp, F)$ be a REV-AA and $h : \Sigma* \rightarrow \Delta^*$ be a homomorphism. By theorem for every REV-AA an equivalent realtime REV-AA can ffectively be constructed. we assume that the given REV-AA M works in realtime. We have to construct a REV-AA M′ which accepts the inverse homomorphic image $h^{-1}(L(M)) = \{w \in \Sigma^* \mid h(w) \in L(M)\}$. The main idea of the following construction is to simulate the computation of M on input h(a) by M′′ on input a in one step. Since |h(a)| may be greater than one it may happen that more than one symbol has to be popped from or pushed onto the stack store in one step. Nevertheless, the maximal number of symbols to be pushed or popped in one step is bounded by the constant $m = \max\{|h(a)| \mid a \in \Sigma\}$. To overcome this problem of stacking and popping several symbols in one step. We add a register to the states in which M′ can store up to m stack symbols of M (the topmost ones), and we consider every string of m stack symbols of M to be a single stack

symbol of M^l . Formally, let$M^l = (Q', \Sigma', \Gamma', \delta', q_0', \perp, F')$ where
$Q' = Q \times \Gamma^{\leq m}$, $\Gamma' = (\Gamma \setminus \{\perp\}^m \cup \{\perp\}$, $q_0' = (q_0, \lambda)$, $F' = F \times \Gamma^{\leq}$ m

Language class	∪	•	*	R	h	h−1	∩REG	∪REG	∩	~
REG	+	+	+	+	+	+	+	+	+	+
L (REV-AA)	−	−	−	−	−	+	−	−	−	+
DCFL	−	−	−	−	−	+	+	+	−	+
CFL	+	+	+	+	+	+	+	+	−	−

Table.1. closure properties of language families discussed.

Now, for every $a \in \Sigma$, M' has to simulate M on input $h(a)$. If $h(a) = \lambda$, then we define transitions $\delta'(q, a, Z) = (q, Z)$, for all $q \in Q'$ and $Z \in \Gamma'$. All these transitions are reversible. If $h(a) \neq \lambda$, then we consider for all $q \in Q$, $y_1 y_2 \cdots y_l \in \Gamma^{\leq m}$, and $Y = y_{l+1}+y_{l+2} \cdots y_{l+m} \in \Gamma'$ the configurations $c = (\lambda, q, h(a), y_1 y_2 \cdots y_l Y)$ and check whether there exists a reversible computation π of M starting on c and ending in configuration $(h(a), q', \lambda, \gamma)$ with $\gamma \in \Gamma^*$ after $|h(a)|$ steps. If such π does not exist, we know that such a computation never occurs as sub-computation in any computation of M, and we leave $\delta'((q, y_1 y_2 \cdots y_l), a, Y)$ undefined in this case. If such a computation exists, then we have to differentiate between three cases.

First, in π more pop operations than push operations, say k_1 pop operations and k_2 push operations, are performed with $k_1 > k_2$. Then, let $k = k_1 - k_2$ and $\gamma = y'_{k+1} y'_{k+2} \cdots y'_{k1} y'_{k1+1} y'_{k1+2} \cdots y'_{l+m}$. Here, primed symbols denote symbols which may have been changed due to push and pop operations. Let us first consider the case $k_1 \leq l$. Then, $k = k_1 - k_2 \leq l - k_2 \leq l$ and we define

$\delta'((q, y_1 y_2 \cdots y_l), a, Y) = ((q', y'_{k+1} y'_{k+2} \cdots y'_{k1} y'_{k1+1} y'_{k1+2} \cdots y_l), Y)$.

If $k_1 > l$, then we define

$\delta'((q, y_1 y_2 \cdots y_l), a, Y) = ((q', y'_{k+1}, y'_{k+2} \cdots y'_l), Y')$,

with $Y' = y'_{l+1} y'_{l+2} \cdots y'_{k1} y'_{k1+1} y'_{k1+2} \cdots y_{l+m}$, if k_l, and

$\delta'((q, y_1 y_2 \cdots y_l), a, Y) = ((q', y'_{l+i+1} y'_{l+i+2} \cdots y'_{k1} y_{k1+1} y_{k1+2} \cdots y_{l+m}), \lambda)$, if $k = l + i$ for $i > 0$.

In the second case, more push than pop operations, say k_1 push operations and k_2 pop operations are performed in π with $k_1 > k_2$. Then, let $k = k_1 - k_2$, $\gamma = z_1 z_2 \cdots z_k y'_1 y'_2 \cdots y'_{k2} y_{k2+1} y_{k2+2} \cdots y_{l+m}$. First, let $k_2 \leq l$. Then, we define

$\delta'((q, y_1 y_2 \cdots y_l), a, Y) = ((q', z_1 z_2 \cdots z_k y'_1 y'_2 \cdots y'_{k2} y_{k2+1} y_{k2+2} \cdots y_l), Y)$,

if $k + l \leq m$, and

$\delta'((q, y_1 y_2 \cdots y_l), a, Y) = ((q', z_1 z_2 \cdots z_i), (z_{i+1} z_{i+2} \cdots z_k y'_1 y'_2 \cdots y'_{k2} y_{k2+1} y_{k2+2} \cdots y_l) Y)$,

if $k + l = m + i$ for $i > 0$. Now, let $k_2 > l$. Then, $k + l = k_1 - k_2 + l < k_1 \leq m$ and we define

$\delta'((q, y_1 y_2 \cdots y_l), a, Y) = ((q', z_1 z_2 \cdots z_k y'_1 y'_2 \cdots y'_l), Y')$,

with $Y' = y'_{l+1} y'_{l+2} \cdots y'_{k2} y_{k2+1} y_{k2+2} \cdots y_{l+m}$.

In the last case, as many push as pop operations are performed in π . This case can be handled similarly.

The defined transitions are reversible, since the knowledge of the state q' , the input symbol $a \in \Sigma$, which defines h(a), and the m topmost stack symbols of M are suffcient to restore state q and the stack store of M′due to the fact that π is reversible. Altogether, M' accepts $h^{-1}(L(M))$ and is a REV-AA. The closure properties of the language families discussed are summarized in Table 1.

3. Decidability questions

Problems which are decidable for DAAs are decidable for REV-AAs as well. Therefore, emptiness, universality, equivalence, and regularity are decidable for REV-AAs. On the other hand, inclusion is known to be undecidable for DAAs. We now show that inclusion is undecidable for REV-AAs, too. To this end, we use a reduction from Post's correspondence problem (PCP) which is known to be undecidable, [24]. Let Σ be an alphabet and an instance of the PCP be given by two lists $\alpha = u_1 , u_2 , . . . , u_k$ and $\beta = v_1 , v_2 , . . . , v_k$ of words from Σ^+ . Furthermore, let $A = \{a_1 ,a_2 , . . . ,a_k \}$ be an alphabet with k symbols, $\Sigma \cap A = \phi$, and $d = \max\{|u_i|,|v_i||1\leq i\leq k$ be the maximal length of words occurring in α or β . Now, consider two languages L_α and L_β .

$$L_\alpha = \{u_{i1}\, u_{i2} \cdots u_{im}\, \$a_{im}^{d+2}d+2a_{m-1}^{d+2} \cdots a_{i1}^{d+2} \mid m\geq1 ,1\leq i_j \leq k,\ 1\leq j\leq m\},$$
$$L_\beta = \{v_{i1}\, v_{i2} \ldots v_{im}\ \$\ a_{im}^{d+2}\ a_{im-1}^{d+2} \ldots a_{i1}^{d+2} \mid m\geq1 ,1\leq i_j \leq k,\ 1\leq j\leq m\}$$

Theorem6. The languages L_α and L_β as well as their reversals L_α^R and L_β^R are accepted by REV-AAs.

Proof. We describe the construction of a REV-AA M_α accepting L_α . The construction for L_β is identical. The idea of the construction is to stack the input onto the stack store until the symbol \$ is read. Then, each block of d + 2 identical symbols a_i is read while u_i is popped from the stack store. In the remaining time up to the end of the block, which is counted in the states, the stack store remains unchanged. Then, the next block of the input and the next word on the stack store is processed. If an error occurs, the transitions remain undefined. Finally, an accepting state is entered when the stack store is empty up to $\perp$.

For the detailed construction let $M_\alpha = \langle Q , \Sigma \cup A \cup \{\$\}, \Sigma \cup \{\perp\}, \delta, q_0 , \perp, \{q_a \}\rangle$ and
$Q = \{q_0 , p, q_a \} \cup \{p_{i,j} ,q_{i,j} \mid 1 \leq i \leq k,$
$1\leq j \leq d\}$. For $a \in \Sigma$, $a' \in \Sigma \cup \{\perp\}$, $1\leq i\leq k$, and $u_i = u_{i,1}\, u_{i,2} \cdots u_{i,|u_i|}$, we define

$$\delta (q_0 , a, a') = (q_0 ,aa'),$$

$$\delta (q_0 , \$,a) = (p,a),$$

$$\delta (p, a_i , a) = (p_{i,1} , a),$$

$$\delta (p_{i,j} , a_i , a) = (p_{i,j+1} , \lambda) \text{ for } 1 \leq j\leq |ui| - 1 \text{ if } a = u_{i,|u_i|-j+1} ,$$

$\delta(p_{i,|u_i|}, a_i, a') = (q_{i,|u_i|}, \lambda)$ if $a = u_{i,1}$,

$\delta(q_{i,j}, a_i, a') = (q_{i,j+1}, a')$ for $|u_i| \leq j \leq d - 1$,

$\delta(q_{i,d}, a_i, a') = (p, a')$,

$\delta(p, \lambda, \perp) = (q_a, \perp)$.

Automaton M_α is reversible, since in the backward computation the input makes sure in which way the stack store has to be restored. Additionally, the input carries the information of how many symbols have to be read before restoring the stack store. Obviously, M_α has exactly one accepting state and every accepting computation ends in a configuration with empty (up to $\perp$) stack store. By the discussion in Remark 1 we obtain that L_α^R and L_β^R can be accepted by REV-AAs as well.

Theorem 7. Let M_1 and M_2 be two REV-AAs. Then it is undecidable whether $L(M_1) \subseteq L(M_2)$.

Proof. We first show that it is undecidable to test whether $L(M_1) \cap L(M_2)$ is the empty set. Given an instance of the PCP we can construct due to Theorem 6 two REV-AAs whose intersection is empty if and only if the PCP has no solution. If we could decide the emptiness of the intersection, we could decide whether or not a PCP has a solution. Obviously, $L(M_1) \subseteq L(M_2)$ if and only if $L(M_1) \cap L(M_2) = \emptyset$. By Theorem 1 we know that L(REV-AA) is closed under complementation. If we could decide the inclusion $L(M_1) \subseteq L(M_2)$, then we could decide the emptiness of intersection as well. This is a contradiction.

Theorem 8. Let M be a nondeterministic Aleshin type automaton. Then it is undecidable whether $L(M) \in$ L (REV-AA).

Proof. We consider an instance of the PCP and define the languages $L_1 = L_\alpha \# L^R_\beta$ and $L_2 = \{w \# w^R \mid w \in (\Sigma \cup A \cup \{\$\})^* \} \cap \Sigma^* \$ A^* \# A^* \$ \Sigma^*$.

Language L_1 belongs to L (REV-AA) due to Theorem 6 and the closure under marked concatenation discussed in Remark 1. So, L_1 is a deterministic context-free language. Clearly, L_2 is a deterministic context-free language as well. Due to the closure of the deterministic context-free languages under complementation [1], we obtain that $\overline{L_1} \cup \overline{L_2}$ is context free. We will now show that $\overline{L_1} \cup \overline{L_2}$ belongs to L (REV-AA) if and only if the given instance of the PCP has no solution. If the instance has no solution, then $L_1 \cap L_2 = \varphi$ and, thus, its complement $\overline{L_1} \cup \overline{L_2}$ is the regular language $(\Sigma \cup A \cup \{\#, \$\})^*$, which belongs to L (REV-AA) due to the regular languages are strictly included in L(REV-AA). On the other hand, if $\overline{L_1} \cup \overline{L_2}$ belongs to L (REV-AA), then its complement $L_1 \cap L_2$ belongs to L (REV-AA) as well. We have to show that the given instance of the PCP has no solution. By way of contradiction we assume that the instance has a solution. Then, $L_1 \cap L_2$ is an infinite, context-free language. Let $w = u_1 u_2 \cdots u_m \$ a_m^{d+2} a_{m-1}^{d+2} \cdots a_1^{d+2} \# a_1^{d+2} a_2^{d+2} \cdots a_m^{d+2} \$ v_m v_{m-1} \cdots v_1$ be a word in $L_1 \cap L_2$ long enough such that the pumping lemma for context-free languages applies. Pumping leads to words which are not in $L_1 \cap L_2$ and we obtain a contradiction.

Now, if we could decide whether the context-free language $\overline{L_1} \cup \overline{L_2}$ belongs to L(REV-AA), then we could decide whether the given instance of the PCP has a solution which is a contradiction.

The same problem of Theorem 8 for deterministic Aleshin type automata is open. However, we have the following decidable property which contrasts the result that there is no algorithm which decides whether, for example, a given cellular automaton or iterative array is reversible [6,7]. Following the discussion in [2], we will consider in the remainder of this section the size of a pushdown automaton as the length of its representation.

Theorem 9. Let M be a deterministic Aleshin type automaton of size n. Then it is decidable in time $O(n^4)$ whether M is a REV-AA.

Proof. In order to decide whether a given deterministic Aleshin type automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$ is reversible, in general, it is not sufficient to inspect the transition function. Whether a transition can be reversed depends on the information that is available after performing it. If this information is unique for all in-transitions to a state, then the transition can be reversed. For example, $\delta(q, a, Z) = (q', Z'Z)$ or $\delta(q, a, Z) = (q', Z')$ provides the state q' , the input symbol a, and the topmost stack symbol Z' . On the other hand, consider $\delta(q, a, Z) = (q', \lambda)$ which provides only the state q and the input symbol a. The necessary information is complemented by the second symbol on the stack store, which cannot be determined by inspecting the transition function only

In order to cope with the problem, we first construct an equivalent DAA $M' = (Q, \Sigma, \Gamma', \delta', q_0, \perp, F)$, where $\Gamma' = \Gamma^2 \cup \{\perp\}$ and $(q', \perp)$ if $\delta(q, a, \perp) = (q', \perp)$,

$\delta'(q, a, \perp) =$ $\begin{cases} (q', (Z\perp)\perp) & \text{if } \delta(q,a,\perp) = (q', Z\perp), \\ \end{cases}$

$(q', (ZY_2))$ if $\delta(q, a, Y_1) = (q', Z)$,

$$\delta'(q,a,(Y_1Y_2)) = \begin{cases} (q', (ZY_1)(Y_1Y_2)) & \text{if } \delta(q, a, Y_1) = (q', ZY_1), \\ (q', \lambda) & \text{if } \delta(q, a, Y_1) = (q', \lambda). \end{cases}$$

By construction there is a bijection ϕ between the configurations passed through by M and M' , where $\phi(v, q, w, Z_1Z_2Z_3 \cdots Z_k\perp) = (v, q, w, (Z_1Z_2)(Z_2Z_3) \cdots (Z_{k-1}Z_k)(Z_k\perp)\perp)$.

Moreover, M and M' have the same initial configurations, and a configuration ca of M is accepting if and only if $\phi(c_a)$ is an accepting configuration of M' . Therefore, M and M′accept the same language. Furthermore, M' is of size O (n). Basically, the idea of the construction is to store information of the second stack symbol in the topmost stack symbol. The construction may introduce also transitions for situations that cannot appear. For example, if in any computation

there is never a Z on top of a Y in the stack store, then the transition δ' (q, a, (ZY)) is useless. However, if a transition of the form $\delta'(q,a, (Y_1\ Y_2)) = (q', \lambda)$ is applied, then we do now have the necessary information to test for uniqueness after having performed the transition as mentioned above. That is, we know the state q' , the input symbol a, and the topmost stack symbol Y_2 . So, basically, it remains to be tested whether a transition is applied in some computation or whether it is useless.
To this end, we label the transitions of δ' uniquely, say by the set of labels $B = \{l_1, l_2, \ldots, l_k\}$. Then we apply an old trick and consider words over the alphabet B. On input $u \in B^*$ a DAA $\overline{M}$ with all states final tries to imitate a computation of M' by applying in every step the transition whose label is currently read. If M' accepts some input $u_1\ u_2 \cdots u_n$, then thereis a computation (not necessarily accepting) of M' that uses the transitions $u_1\ u_2 \cdots u_n$ in this order. If conversely there is a computation of M' that uses the transitions $u_1\ u_2 \cdots u_n$ in this order, then $u_1\ u_2 \cdots u_n$ is accepted by $\overline{M}$ .So, in order to determine whether a transition with label l_i of M' is useful, it suffices to decide whether $\overline{M}$ accepts an input containingthe letter l_i. This decision can be done by testing the emptiness of the deterministic context-free language $L(\overline{M}) \cap B^* l_i B^*$ .Concerning the time complexity of identifying all useless transitions, we first observe that the size of $\overline{M}$ and of each DAA $\overline{M}\ l_i$ accepting $L(\overline{M}) \cap B^* l_i B^*$ is in O (n). To test the emptiness of some DAA $\overline{M}\ l_i$, we have to convert $\overline{M}$ li to an equivalent context-free grammar and test its emptiness. According to [2] the conversion to an equivalent context-free grammar has time complexity O (n^3).This implies that the time complexity of removing all useless transitions is in $O(n^4)$

Assume that M'' is constructed from M' by deleting all useless transitions. Clearly, M'' and M are equivalent and the size of M'' is in O (n). Now, for any state we consider all in-transitions and check whether the corresponding information after performing it (state, input symbol and pushdown symbol) is unique. If this is true for all states, then M is reversible, and irreversible otherwise. The latter test has time complexity O (n^2). Thus, we obtain that the reversibility of M can be decided in O (n^4) time.

Corollary 3. Let M be a nondeterministic Aleshin type automaton of size n. Then it is decidable in time O (n^4) whether M is a REV-AA.
Proof. By inspecting the transition function one can decide whether or not M is a DAA. If the answer is yes, then it can be decided whether M is a REV-AA by Theorem 9. If M is not a DAA, then it cannot be a REV-AA. Since the inspection of the transition function can be done in O (n^2) time, we obtain the time complexity claimed.

Theorem 10. The decision problem whether a given deterministic Aleshin type automaton is a REV-AA is P-complete.

Proof. By Theorem 9 the problem is in P. Its P-hardness is shown by reduction of the

P-complete problem GEN discussed in [4]. Given a finite set X , a binary operation $\bullet$ on X (presented as a table), a subset $S \subseteq X$, and an element $w \in X$, GEN is the problem to decide whether w is contained in the smallest subset of X which contains S and is closed under the operation $\bullet$.

For a given instance of GEN we construct a pushdown automaton $M = \langle\{q_0 ,q, f \}, \{e\}, X \cup \{\perp\}, \delta,q_0 , \perp, \{ f \}\rangle$, where

$(q, w) \in \delta (q_0 , \lambda, \perp)$,

$(f , \perp) \in \delta (q, \lambda, \perp)$,

$(q, \lambda) \in \delta (q, \lambda, x)$, if $x \in S$, and

$(q, yz) \in \delta (q, \lambda, x)$, if $x = y \bullet z$ for some $y, z \in X$.

The construction of M can be done in logarithmic space with regard to the instance of GEN. Moreover, $L(M) = \phi$ if and only if w is not generated by S and, thus, does not belong to the smallest subset of X which contains S and is closed under the operation •.

Next, M is transformed into a REV-AA M' such that $L(M) = \phi$ if and only if $L(M') = \phi$. To this end, we construct M' by labeling the transitions of M uniquely by some set of labels B , and consider words over the alphabet B as inputs. The DAA M' tries to imitate a computation of M by applying in every step the transition whose label is currently read. The resulting DAA M' is reversible since each input symbol indicates which transition of M has to be chosen by the reverse transition function of M'. The construction of M' can be done in logarithmic space. Moreover, $L(M) = \o$ if and only if $L (M') = \phi$.

Finally, we construct another DAA M'' by concatenating the language $L = \{a^n b^n \mid n \geq 0\}$ to $L (M')$. To this end, appropriate transitions from the state f to an initial configuration of a DAA accepting L have to be added. Again, the construction of M'' can be done in logarithmic space.

we know that L cannot be accepted by any REV-AA. So, M'' is not a REV-AA if $L(M') \neq \phi$. On the other hand, if $L (M') = \phi$, then M'' is reversible, since the simulation of M' is reversible by construction, and the configuration with state f leading to a possibly non-reversible computation never appears. Altogether, we obtain that $L (M') \neq \phi$ if and only if M'' is a REV-AA. This concludes the reduction and shows the P-hardness of the given problem. W

Corollary. The decision problem whether a given nondeterministic Aleshin type automaton is a REV-AA is P-complete.

4. Conclusion

All deterministic context-free languages can be parsed in linear time using the well-known parsing algorithms for grammars. Thus, reversible deterministic context-free languages can be parsed in linear time as well. But taking into account the constants arising in the time complexity, it might be the case that reversible deterministic context-free languages can be parsed in less time.

References

[1] M.A. Harrison, Introduction to Formal Language Theory, Addison-Wesley, Reading, 1978.

[2] J.E. Hopcroft, R. Motwani, J. Ullman, Introduction to Automata Theory, Languages, and Computation, Pearson, Upper Saddle River, 2003.

[3] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, 1979.

[4] N.G. Jones, W.T. Laaser, Complete problems for deterministic polynomial time, Theoret. Comput. Sci. 3 (1977) 105–117.

[5] S. Kobayashi, T. Yokomori, Learning approximately regular languages with reversible languages, Theoret. Comput. Sci. 174 (1997) 251–257.

[6] M. Kutrib, A. Malcher, Fast reversible language recognition using cellular automata, Inform. and Comput. 206 (2008) 1142–1151.

[7] M. Kutrib, A. Malcher, Real-time reversible iterative arrays, Theoret. Comput. Sci. 411 (2010) 812–822.

[8] R. Landauer, Irreversibility and heat generation in the computing process, IBM J. Res. Dev. 5 (1961) 183–191.

[9] M. Latteux, A. Lemay, Y. Roos, A. Terlutte, Identification of biRFSA languages, Theoret. Comput. Sci. 356 (2006) 212–223.

[10] S. Lombardy, On the construction of reversible automata for reversible languages, in: ICALP 2002: Automata, Languages and Programming, in: Lecture Notes in Comput. Sci., vol. 2380, Springer, Berlin, 2002, pp. 170–182.

[11] K. Morita, A. Shirasaki, Y. Gono, A 1-tape 2-symbol reversible Turing machine, Trans. IEICE E 72 (1989) 223–228.

[12] K. Morita, Reversible computing and cellular automata—a survey, Theoret. Comput. Sci. 395 (2008) 101–131.

[13] I. Phillips, I. Ulidowski, Reversing algebraic process calculi, J. Log. Algebr. Program. 73 (2007) 70–96.

[14] J.-E. Pin, On reversible automata, in: Latin 1992: Theoretical Informatics, in: Lecture Notes in Comput. Sci., vol. 583, Springer, Berlin, 1992, pp. 401–416.

[15] A. Salomaa, Formal Languages, Academic Press, New York, 1973.

[16] T. Yokoyama, H.B. Axelsen, R. Glück, Reversible flowchart languages and the structured reversible program theorem, in: ICALP 2008: Automata, Languages and Programming, in: Lecture Notes in Comput. Sci., vol. 5126, Springer, Berlin, 2008, pp. 258–270.

[17] T. Yokoyama, Reversible computation and reversible programming languages, Electron. Notes Theor. Comput. Sci. 253 (2010) 71–81.