Volume 1, Issue 5 December 2015 ISSN NO: 2454 - 7514

North Asian International Research Journal Consortium

North Asian International Research Journal

Of

Science, Engineering and Information Technology

Welcome to NAIRJC

ISSN NO: 2454 -7514

North Asian International Research Journal of Science, Engineering & Information Technology is a research journal, published monthly in English, Hindi, Urdu all research papers submitted to the journal will be double-blind peer reviewed referred by members of the editorial board. Readers will include investigator in Universities, Research Institutes Government and Industry with research interest in the general subjects

Editorial Board

Address: - Dr. Ashak Hussain Malik House No. 221 Gangoo, Pulwama, Jammu and Kashmir, India - 192301, Cell: 09086405302, 09906662570, Ph. No: 01933-212815,
Email: nairjc5@gmail.com, nairjc@nairjc.com, info@nairjc.com Website: www.nairjc.com

COMPUTATIONAL CAPACITY OF ALESHIN TYPE AUTOMATA

[1]A.JEYANTHI

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

[2] B.STALIN

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

ABSTRACT

Structural properties and computational capacity of reverse Aleshin type automata are introduced. In this paper, the computational capacity of reversible computations in aleshin type automata is investigated and turns out to lie properly in between the regular and deterministic context-free languages. Furthermore, it is shown that a deterministic context-free language cannot be accepted reversibly if more than real time is necessary for acceptance.

Keywords: Bisimulation probabilistic automata (BPA), probabilistic Aleshin automata, probabilistic labeled transition system (pLTS).

1. INTRODUCTION

Nowadays, reversible computing has become a field of intensive study from several perspectives. In [12] one may find a recent survey which summarizes results on reversible Turing machines, reversible cellular automata, which are a massively parallel model consisting of interacting deterministic finite automata, and other reversible models such as logic gates, logic circuits, or logic elements with memory. Reversible deterministic finite automata are also studied in the context of algorithmic learning theory [2] and quantum computing [8,9] whereas construction problems are investigated in [5,6]. A recent paper which motivates the study of reversible computing from the vantage point of physics is [4]. How to compute reversibly by using a reversible programming language is presented in [2]. Reversibility has also been studied for other computational models such as, for example, flowcharts [5] or process calculi [4]. See also the references in [6,8].

Computers are information processing devices which are physical realizations of abstract computational models. It may be difficult to define exactly what information is or how information should be measured suitably. It may be even more difficult to analyze in detail how a computational device processes or transmits information while working on some input. Thus, one first step towards a better understanding of information is to study computations in which

no information is lost. Another motivation to study information preserving computations is the physical observation that a loss of information results in heat dissipation [3,11]. A first study of this kind has been done in [3] for Turing machines where the notion of reversible Turing machines is introduced. Deterministic Turing machines are called reversible when they are also backward deterministic. One fundamental result shown in [3] is that every, possibly irreversible, Turing machine can always be simulated by a reversible Turing machine in a constructive way. This construction is significantly improved in [7] with respect to the number of tapes and tape symbols. Thus, for the powerful model of Turing machines, which describe the recursively enumerable languages, every computation can be made information preserving. At the other end of the Chomsky hierarchy there are the regular languages. Reversible variants of deterministic finite automata have been defined and investigated in [2,12]. It turns out that there are regular languages for which no reversible deterministic finite automaton exists. Thus, there are computations in which a loss of information is inevitable. Another result of [12] is that the existence of a reversible automaton can be decided for a regular language in polynomial time.

Reversible variants of the massively parallel model of cellular automata and iterative arrays have been also studied in [5,6] with regard to the acceptance of formal languages. One main result there is the identification of data structures and constructions in terms of closure properties which can be implemented reversibly. Another interesting result is that, in contrast to regular languages, there is no algorithm which decides whether a given cellular device is reversible.

2. PRELIMINARIES AND DEFINITIONS

Let Σ^* denote the set of all words over the finite alphabet Σ . The empty word is denoted by λ, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. The set of words of length at most $n \geq 0$ is denoted by $\Sigma^{\leq n}$. For convenience, we use Σ_λ for $\Sigma \cup \{\lambda\}$. The reversal of a word w is denoted by w^R and for the length of w we write $|w|$. The number of occurrences of a symbol $a \in \Sigma$ in $w \in \Sigma^*$ is written as $|w|_a$. Set inclusion is denoted by $\subseteq$, and strict set inclusion by $\subset$. A deterministic Aleshin type automaton (DAA) is a system $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \perp, F \rangle$, where Q is a finite set of states, Σ is the finite input alphabet, Γ is a finite pushdown alphabet, $q_0 \in Q$ is the initial state, $\perp \in \Gamma$ is a distinguished pushdown symbol, called the bottom-of-pushdown symbol, which initially appears on the pushdown store, $F \subseteq Q$ is the set of accepting states, and δ is a mapping from $Q \times \Sigma_\lambda \times \Gamma$ to $Q \times \Gamma^*$ called the transition function. There must never be a choice of using an input symbol or of using λ input. So, it is required that for all q in Q and Z in Γ : if $\delta(q, \lambda, Z)$ is defined, then $\delta(q,a,Z)$ is undefined for all a in Σ . A configuration of a Aleshin automaton is a quadruple (v, q, w, γ), where q is the current state, v is the already read and w the unread part of the input, and γ the current content of the pushdown store, the leftmost symbol of γ being the top symbol. On input w the initial configuration is defined to be $(\lambda, q_0, w, \perp)$. For $q \in Q$, $a \in \Sigma_\lambda$, v , $w \in \Sigma^*$, $\gamma \in \Gamma^*$, and $Z \in \Gamma$, let $(v, q, aw, Z\gamma)$ be a configuration. Then its successor

configuration is (va, p, w , βγ), where δ (q,a, Z) = (p, β). We write (v ,q,aw , Zγ) ⊢ (va, p, w , βγ) in this case. The reflexive transitive closure of ⊢ is denoted by $\vdash^*$ .To simplify matters, we require that in any configuration the bottom-of-pushdown symbol appears exactly once at the bottom of the pushdown store, that is, it can neither appear at some other position in the pushdown store nor be deleted. Formally, we require that if δ (q, a, Z) = (p, β) then either $Z \neq \perp$ and β does not contain ⊥, or Z = ⊥ and β = β′ ⊥, where β′ does not contain ⊥. The language accepted by M with accepting states is

$$L(M) = \{w \in \Sigma^* \mid (\lambda, q0, w, \perp) \vdash^* (w, q, \lambda, \gamma), \text{ for some } q \in F \text{ and } \gamma \in \Gamma^*\}$$

In general, the family of all languages that are accepted by some device X is denoted by L (X).

Now we turn to reversible Aleshin type automata. Reversibility is meant with respect to the possibility of stepping the computation back and forth. To this end, the Aleshin type automata have to be also backward deterministic. That is, any configuration occurring in any computation must have at most one predecessor which, in addition, is computable by a DAA. For reverse computation steps the head of the input tape is always moved to the left. Therefore, the automaton rereads the input symbol which has been read in a preceding forward step. So, for reversible Aleshin type automata there must exist a reverse transition function.

A reverse transition function $\delta_R : Q \times \Sigma_\lambda \times \Gamma \rightarrow Q \times \Gamma^*$ maps a configuration to its predecessor configuration. For $q \in Q$, $a \in \Sigma_\lambda$, v , $w \in \Sigma^*$, $\gamma \in \Gamma^*$, and $Z \in \Gamma$, let (va,q, w , Zγ) be a configuration. Then its predecessor configuration is (v , p, aw , βγ), where δ_R (q,a, Z) = (p, β). We write (va,q, w , Zγ) ⊣ (v , p,aw , βγ) in this case. Automaton M is said to be reversible (REV-AA), if there exists a reverse transition function δ_R such that $c_{i+1} \dashv c_i$, $0 \leq I \leq n - 1$, for any sequence $c_0 \vdash c_1 \vdash \ldots\ldots \vdash c_n$ of configurations passed through by M and beginning with an initial configuration c_0 .

To clarify our notion we continue with an example.

Example 1. The linear context-free language $\{wcw^R \mid w \in \{a,b\}^*\}$ is accepted by the REV-AA $M = \langle \{q_0, q_1, q_2\}, \{a,b, c\}, \{a, b, \perp\}, \delta, q_0 , \perp, \{q_2\} \rangle$, where the transition functions δ and δ_R are as follows.

Transition function δ

(1) $\delta (q_0 , a, \perp) = (q_0 , a\perp)$
(2) $\delta (q_0 , b, \perp) = (q_0 , b\perp)$
(3) $\delta (q_0 , a, a) = (q_0 , aa)$
(4) $\delta (q_0 , a, b) = (q_0 , ab)$
(5) $\delta (q_0 , b, a) = (q_0 , ba)$
(6) $\delta (q_0 , b, b) = (q_0 , bb)$
(7) $\delta (q_0 , c , \perp) = (q_1 , \perp)$

(8) $\delta (q_0 , c , a) = (q_1 ,a)$

(9) $\delta (q_0 , c , b) = (q_1 , b)$

(10) $\delta (q_1 , a, a) = (q_1 , \lambda)$

(11) $\delta (q_1 , b, b) = (q_1 , \lambda)$

(12) $\delta (q_1 , \lambda, \perp) = (q_2 , \perp)$

The transitions (1)-(6) of δ are used by M to store the input prefix w . When a c appears in the input, transitions (7)-(9) are used to change to state q_1 while the pushdown store remains unchanged. By transitions (10) and (11) the input suffix w^R is matched with the stored prefix w . Finally, if the bottom-of Aleshin type symbol is seen in state q_1 , automaton M changes into the sole accepting state q_2 and the computation necessarily stops.

Reverse transition function δ_R

(1) $\delta_R (q_0 , a, a) = (q_0 , \lambda)$

(2) $\delta_R (q_0 , b, b) = (q_0 , \lambda)$

(3) $\delta_R (q_1 , c , \perp) = (q_0 , \perp)$

(4) $\delta_R (q_1 , c , a) = (q_0 ,a)$

(5) $\delta_R (q_1 , c , b) = (q_0 , b)$

(6) $\delta_R (q_1 , a, a) = (q_1 ,aa)$

(7) $\delta_R (q_1 , a, b) = (q_1 ,ab)$

(8) $\delta_R (q_1 , b, a) = (q_1 , ba)$

(9) $\delta_R (q_1 , b, b) = (q_1 , bb)$

(10) $\delta_R (q_1 , a, \perp) = (q_1 ,a\perp)$

(11) $\delta_R (q_1 , b, \perp) = (q_1 , b\perp)$

(12) $\delta_R (q_2 , \lambda, \perp) = (q_1 , \perp)$

For the backward computation the transitions of δ_R are used. Since there is only one transition of δ that changes to state q_2 , transition (12) reverses this step. For input symbols a and b, the only transitions of δ that change to state q_1 are (8) and (9) which pop the symbol from the top of the pushdown store if it matches the current input symbol. So, transitions (6)-(11) of δ_R are constructed to reverse the popping by pushing the current input symbol. In forward computations M changes from state q0 to q_1 if and only if the current input symbol is a c , whereby the pushdown store remains unchanged. These steps can uniquely be reversed by the transitions (3)-(5) of δ_R . While in state q_0 , in any forward step an input symbol a or b is pushed. Therefore, δ_R reverses the pushing by popping whenever the stack store is not empty and an a or b appears in the input by transitions (1) and (2). This concludes the construction of δ_R .

Example 2. Only slight modifications of the construction given in Example 1 show that the languages $\{a^ncb^n \mid n \geq 0\}$, and $\{a^ncb^n \mid n \geq 0\}^*$, as well as $\{a^mcb^nea^m \mid m,n \geq 0\} \cup \{a^ndb^nea^m \mid m,n \geq 0\}$ are accepted by REV-PDAs as well.

3. STRUCTURAL PROPERTIES AND COMPUTATIONAL CAPACITY

In this section the computational capacity of REV-AAs is considered. First, we examine the structure of transitions that enable reversibility, and investigate the role played by λ-steps.

Fact 3. Let $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \perp, F \rangle$ be a REV-AA.

1. As for the transition function also for the reverse transition function δ_R we have necessarily that for all q in Q and Z in Γ : if $\delta_R(q, \lambda, Z)$ is defined, then $\delta_R(q,a, Z)$ is undefined for all a in Σ . Otherwise the predecessor configuration would not be unique and, thus, M not be reversible.
2. All transitions of M are either of the form $\delta(q,a,Z) = (p, \lambda)$ (pop), or $\delta(q,a, Z) = (p, Y)$ (top), or $\delta(q,a, Z) = (p, Y Z)$ (push), where $q, p \in Q$, $a \in \Sigma_\lambda$, $Y, Z \in \Gamma$. There is no transition that modifies the stack store except for the topmost symbol, since the reverse transition has only access to the topmost symbol.

It is well known that general deterministic Aleshin type automata that are not allowed to perform λ-steps are weaker than DAAs that may move on λ input [10]. To go a little more into details we consider the maximal number of consecutive λ-steps. A REV-AA is said to be quasi realtime if there is a constant that bounds this number for all computations. The REV-AA is said to be realtime if this constant is 0, that is, if there are no λ-steps at all. In the following we also deal with weakly quasi realtime pushdown automata, that is, the length of any sequence of consecutive λ-steps in any computation is either bounded by a constant depending on the pushdown automaton only or infinite.

Theorem 1. For every REV-AA an equivalent weakly quasi realtime REV-AA can effectively be constructed.

Proof. Given a REV-AA $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \perp, F \rangle$ we construct an equivalent REV-AA $M\_ = \langle Q, \Sigma, \Gamma, \delta\_, q_0, \perp, F \rangle$ by modifying δ with respect to λ-transitions as follows.

(1) Two consecutive top-transitions (Fact 3) are merged into one. That is, every pair of the form $\delta(q, \lambda, Z) = (q', Z')$ and $\delta(q', \lambda, Z') = (q'', Z'')$ is replaced by $\delta'(q, \lambda, Z) = (q'', Z'')$.Since M is deterministic every application of the first transition is followed by an application of the second transition, and since M is reversible every application of the second

transition is preceded by an application of the first transition. The corresponding reverse transitions $\delta_R(q', \lambda, Z') = (q, Z)$ and $\delta_R(q'', \lambda, Z'') = (q', Z')$ are replaced by $\delta_R(q'', \lambda, Z'') = (q, Z)$. So, the construction step preserves reversibility and yields to an equivalent automaton.

(2) A Aleshin type-transition and a following top-transiti on are merged into one Aleshin type transition. That is, every pair of the form $\delta(q, \lambda, Z) = (q', Z'Z)$ and $\delta(q', \lambda, Z') = (q'', Z'')$is replaced by $\delta'(q, \lambda, Z) = (q'', Z'', Z)$. The corresponding reverse transitions $\delta_R(q', \lambda, Z') = (q, \lambda)$ and $\delta_R(q'', \lambda, Z'') = (q', Z')$ are replaced by $\delta'_R(q'', \lambda, Z'') = (q, \lambda)$ Similar as above, the construction step preserves reversibility and yields to an equivalent automaton.

(3) A Aleshin -transition and a following pop-transition are merged into one top-transition. That is, every pair of the form $\delta(q, \lambda, Z) = (q', Z'Z)$ and $\delta(q', \lambda, Z') = (q'', \lambda)$ is replaced by $\delta'(q, \lambda, Z) = (q'', Z)$. The corresponding reverse transitions

$\delta_R(q', \lambda, Z') = (q, \lambda)$ and $\delta_R(q'', \lambda, Z) = (q', Z', Z)$ are replaced by $\delta'_R(q'', \lambda, Z) = (q, Z)$. Again, the construction step preserves reversibility and yields to an equivalent automaton.

Next, the three steps are repeated until no more merging is possible, which concludes the construction of M'. It remains to be shown that the REV-AA M' is weakly quasi realtime. Due to the construction, any sequence of consecutive λ-steps in any computation of M' possibly starts with a sequence of pop- and top-moves, where no two top-moves appear consecutively. Then several push-moves may follow. After a push-move there is never a pop- or top-move.

Assume that there is a computation on some input such that at the beginning of a sequence of λ-steps at least $|Q| \cdot |\Gamma|$ consecutive pop- or top-moves are performed. If these steps appear before any non-λ-step, M' starts each computation with an infinite loop on λ input and, thus, M' is weakly quasi realtime.

Next assume that at least $|Q| \cdot |\Gamma|$ consecutive pop- or top-steps appear after some non-λ-step, and let $r : \Sigma^* \times Q \times \Sigma^* \times \Gamma^+ \longrightarrow Q \times \Gamma$ be a mapping that maps a configuration to its state and the topmost Aleshin type symbol. Then there is a (partial) computation $c_{k-1} \vdash c_k \vdash^* c_{k+i} \vdash^* c_{k+i+j-1} \vdash c_{k+i+j}$, where the transition from ck−1 to ck reads some non-λ input $a \in \Sigma$ and all the other transitions are on λ input. Moreover, we have $r(c_{k+i}) = r(c_{k+i+j})$, for some minimal $0 \leq i$, $1 \leq j$ such that $i + j \leq |Q| \cdot |\Gamma|$. Let $r(c_{k+i}) = (p, Z)$. Then, for $i = 0$, $\delta_R(p, \lambda, Z)$ has to be defined to get back from configuration c_{k+j} to configuration c_{k+j-1}. At the same time $\delta_R(p, a, Z)$ has to be defined to get back from configuration c_k to configuration c_{k-1}, a contradiction. For $I \geq 1$we know that $r(c_{k+i-1})$ and $r(c_{k+i+j-1})$ are different since i has been chosen to be minimal.

Since for this case $\delta_R(p, \lambda, Z)$ has to be defined in such a way that the computation steps back from configuration c_{k+I} to configuration c_{k+i-1}, and at the same time such that the computation steps back from ck+i+j to $c_{k+i+j-1}$ by push- or top-moves, we obtain a contradiction, too.Therefore, any sequence of consecutive λ-steps starts with at most $|Q| \cdot |\Gamma|$ pop- or top-moves. If there are at least $|Q| \cdot |\Gamma|$ subsequent push moves, the computation runs

into an infinite loop on λ input and, thus, M′ is weakly quasi realtime. If, otherwise, there are less than $|Q| \cdot |\Gamma|$ push-moves, the length of the whole sequence of λ-steps is bounded by the constant $2 \cdot |Q| \cdot |\Gamma|$ that depends on M′ only. So, also in this case M′ is weakly quasi realtime.

To conclude the consideration of λ-steps we present the result that the family L (REV-AA) is a subfamily of the realtime deterministic context-free languages which is the class of languages accepted by DAAs that perform no λ-steps.

Theorem 2: For every REV-AA an equivalent realtime REV-AA can effectively be constructed.

Proof. Let $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \perp, F \rangle$ be a REV-AA. By Lemma 4 we may assume that M is weakly quasi realtime such that the number of any consecutive λ-steps is bounded by $d < 2 \cdot |Q| \cdot |\Gamma|$ or is infinite, where in the latter case the infinite loop consists of Aleshin type - moves only. Therefore, to check whether the sequence starting from a given configuration is finite, we have to simulate at most d steps of M.

In order to construct an equivalent realtime REV-AA M′, basically, the idea is to simulate a possibly empty sequence of λ-moves, one following non-λ-step, and a following possibly empty sequence of λ-moves at once. If a sequence of λ-moves is infinite, it may drive M through accepting and rejecting states. So, the simulation stops accepting or rejecting dependent on whether an accepting state appears in the loop. In addition, special attention has to be paid for computations where a bounded sequence of λ-steps appears after reading the last input symbol. Again, these λ-steps may drive M through accepting and rejecting states. So, we cannot simply simulate a sequence entirely, since the last state could be rejecting while predecessor states are accepting. We construct $M' = \langle Q', \Sigma', \Gamma', \delta', q_0', \perp, F' \rangle$ in such a way that a possibly empty, finite sequence of λ-moves and a non-λ-step of M are simulated together with a possibly empty, finite sequence of λ steps following the non-λ-step, where the second sequence of λ-steps is simulated until an accepting state appears for the last time or entirely if it consists of rejecting states only. Moreover, the whole simulation has to preserve the reversibility of M.

For a formal construction, we exclude the case where an infinite loop on λ input appears before any non-λ-step. In this case, M starts each computation with an infinite loop on λ input. Depending on whether this loop includes an accepting state, L (M) is either {λ} or ∅. For both languages there is an equivalent realtime REV-AA.

For the other case, we recall that M′ simulates at most 2d + 1 steps of M at once during which it has to access no more than the topmost 2d + 1 stack symbols. On the other hand, it cannot push more than 2d + 1 symbols onto the store. For the construction obeying the properties of Fact 3, we add a register to the states in which M′ can store up to 2d Aleshin type pushdown symbols of M (the topmost ones), and consider every string of 2d + 1 pushdown symbols of M to be a single Aleshin type pushdown symbol of M′ :

$Q = \{q_a, q_r\} \cup \_(Q \times \Gamma^{\leq 2d})$, $\Gamma' = (\Gamma \setminus \{\perp\})^{2d+1} \cup \{\perp\}$,

$q_0' = (q_0, \lambda)$, $F' = \{qa\} \cup (F \times \Gamma^{\leq 2d})$.

Given $(q, x_1 x_2 \cdots x_k) \in Q'$, $0 \leq k \leq 2d$, $a \in \Sigma$, $v \in \Sigma^*$, and $z_1 z_2 \cdots z_{2d+1} \in \Gamma'$, the transition $\delta'((q, x_1 x_2 \cdots x_k), a, z_1 z_2 \cdots z_{2d+1})$ is defined by the computation

$c1 \vdash c2 \vdash \ldots\ldots c_n$ of M starting on $c1 = (v, q, a, x_1 x_2 \cdots x_k z_1 z_2 \cdots z_{2d+1})$, where $n \leq 2d + 1$.

Case 1. The computation starts with a possibly empty sequence of λ-moves, followed by an a-move during 2d + 1 steps, and subsequently M runs into an infinite loop of push-moves on λ input. If this loop contains an accepting state we define $\delta'((q, x_1 x_2 \cdots x_k), a, z_1 z_2 \cdots z_{2d+1}) = (q_a, z_1 z_2 \cdots z_{2d+1})$, otherwise $\delta'((q, x_1 x_2 \cdots x_k), a, z_1 z_2 \cdots z_{2d+1}) = (q_r, z_1 z_2 \cdots z_{2d+1})$, where δ' is undefined for q_a and q_r.

Case 2. The computation starts with a possibly empty sequence of λ-moves, followed by an a-move during 2d + 1 steps, and subsequently M performs a finite number of λ-steps. Then let c_n be the configuration reached after the last λ-step, and c_m, $m \leq n$, be the configuration reached sometime after the non-λ-step in which an accepting state appears for the last time, or cm = cn if none of these configurations is accepting.
Now, let $cm = (v', p, \lambda, y_j y_{j-1} \cdots y_1)$ and define

$$\delta'((q, x_1 x_2 \cdots x_k), a, z_1 z_2 \cdots z_{2d+1}) = \begin{cases} ((p, y_j \cdots y_1), \lambda) & \text{if } 0 \leq j \leq 2d, \\ ((p, y_j \cdots y_{2d+2}), y_{2d+1} \cdots y_1) & \text{if } 2d+1 \leq j \leq 4d+1, \\ ((p\, y_j \cdots y_{4d+3}), y_{4d+2} \cdots y_{2d+2}, y_{2d+1} \cdots y_1) & \text{if } 4d+2 \leq j \leq 6d+2. \end{cases}$$

Note that in the last alternative, M could not have had access to the symbols $z_1, z_2, \ldots, z_{2d+1}$ and, therefore, $y_{2d+1} \cdots y_1 = z_1 z_2 \cdots z_{2d+1}$. So, the properties of Fact 3 are obeyed. The completion of the definition of δ' for the situations in which the bottom-of-pushdown symbol is the topmost symbol is straightforward. The case where no non-λ-step appears during 2d + 1 steps can only appear at the beginning and has been excluded before.

Given an input w, the computation of M is unambiguously split into sequences of steps each of which is performed by M′ at once. If M accepts, so does M′ also in cases where the input is accepted after some λ-steps at the end of the computation. Conversely, every step of M′ corresponds to a sequence of steps of M. So, we have L (M) = L (M′). Moreover, M′ is reversible, since δ'_R can be defined by δ_R in almost the same way as δ' by δ. The only difference concerns the occurrence of accepting states in sequences of λ-transitions following

non-λ-steps. The reverse transition δ_R' simulates the sequence until the first accepting state appears or not at all if it consists of rejecting states only. By construction, M′ is realtime.

Theorem 5 provides a class of deterministic context-free languages that are not reversible. Every deterministic context free language that is not realtime is not accepted by any REV-AA. For example, the language

$$\{a^m e b^n c a^m | m,n \geq 0\} \cup \{a^m e b^n c a^m | m,n \geq 0\}$$

does not belong to the family L (REV-AA) (see, for example, [7,10]). This result immediately raises the question of whether all realtime deterministic context-free languages are reversible. The next lemma answers this question negatively.

Theorem 3. The realtime deterministic linear language $\{a^n b^n \mid n \geq 0\}$ is not accepted by any REV-AA.

Proof. Assume in contrast to the assertion that $L = \{a^n b^n \mid n \geq 0\}$ is accepted by some REV-AA $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \perp, F \rangle$. Without loss of generality, we may assume that M is realtime. During the computation of M on input prefixes a^+ no combination of state and content of the stack store may appear twice. If

$$(\lambda, q_0, a^n b^n, \perp) \vdash^* (a^{m1}, q_1, a^{n-m1} b^n, \sigma_1) \vdash^+ (a^{m1+m2}, q_1, a^{n-m1-m2} b^n, \sigma_1)$$

is the beginning of an accepting computation, then so is $(\lambda, q_0, a^{n-m2} b^n, \perp) \vdash^* (a^{m1}, q_1, a^{n-m1-m2} b^n, \sigma_1)$, but $a^{n-m2} b^n$ does not belong to L . This implies that each height of the stack store may appear only finitely often and, thus, that the height increases arbitrarily. So, M runs into a loop while processing a's, that is, the combination of a state and, for any fixed number k, some k topmost pushdown symbols α appear again and again. To render the loop more precisely, let $(a^{n-x}, q, a^x b^n, \alpha\gamma)$ be a configuration of the loop. Then there is a successor configuration with the same combination of state and topmost pushdown symbols $(a^{n-x+y}, q, a^{x-y} b^n, \alpha\beta)$. We may choose α so that during the computation starting in $(a^{n-x}, q, a^x b^n, \alpha\gamma)$ no symbol of γ is touched, that is, $\alpha\beta = \alpha\gamma'\gamma$. Therefore, the computation continues as

$$(a^{n-x+y}, q, a^{x-y} b^n, \alpha\gamma'\gamma) \vdash^+ (a^{n-x+2y}, q, a^{x-2y} b^n, \alpha\gamma'\gamma'\gamma).$$

Next, we turn to the input suffixes. While M processes the input suffixes b^+ , again, no combination of state and content of the pushdown store may appear twice. If

$$(\lambda, q_2, b^n, \sigma_2) \vdash^* (a^n b^{m1}, q_3, a^{n-m1} b^n, \sigma_3) \vdash^+ (a^n b^{m1+m2}, q_3, b^{n-m1-m2} b^n, \sigma_3)$$

results in an accepting computation, then so does

$$(a^n, q_2, b^{n-m2}, \sigma_2) \vdash^* (a^n b^{m1}, q_3, b^{n-m1-m2}, \sigma_3),$$

but $a^n b^{n-m2}$ does not belong to L . This implies that each height of the stack store may appear

only finitely often. Moreover, in any accepting computation the stack store has to be decreased until some symbol of γ appears. Otherwise, we could increase the number of a's by y to drive M through an additional loop while processing the input prefix. The resulting computation would also be accepting but the input does not belong to L. Together we conclude that M runs into a loop that decreases the height of the pushdown store while processing the b's, and that there are only finitely many combinations of state and content of the stack store which are accepting.

Now, consider two different numbers $n_1 < n_2$ such that M accepts $a^{n1}b^{n1}$ and $a^{n2}b^{n2}$ in the same combinations of state and content of the pushdown store, say in state qa with γa in the pushdown store. We have the forward computations $(\lambda, q_0, a^{n1}b^{n1}, \perp) \vdash^{n1} (a^{n1}, q_1, b^{n1}, \gamma_1) \vdash^{n1} (a^{n1}b^{n1}, q_a, \lambda, \gamma_a)$ and $(\lambda, q_0, a^{n2}b^{n2}, \perp) \vdash^{n1} (a^{n1}, q_1, a^{n2-n1}b^{n2}, \gamma_1) \vdash^{n2-n1} (a^{n2}, q_2, b^{n2}, \gamma_2) \vdash^{n2} (a^{n2}b^{n2}, q_a, \lambda, \gamma_a)$. Since M is reversible and runs through loops while processing the b's, the backward computation also runs through loops that now increase the height of the stack store. This backward loop cannot be left while reading b's. So, we have $(a^{n1}b^{n1}, q_a, \lambda, \gamma_a) \dashv^{n1} (a^{n1}, q_1, b^{n1}, \gamma_1)$ and $(a^{n2}b^{n2}, q_a, \lambda, \gamma_a) \dashv^{n1} (a^{n2}b^{n2-n1}, q_1, b^{n1}, \gamma_1) \dashv^{n2-n1} (a^{n2}, q_2, b^{n2}, \gamma_2)$

Due to the deterministic behavior and the reversibility the last step implies $(a^{n2}, q_2, b^{n2}, \gamma_2) \vdash^{n2-n1} (a^{n2}b^{n2-n1}, q_1, b^{n1}, \gamma_1)$.

Finally, we consider the input $a^{n2}b^{n2-n1}a^{n2-n1}b^{n2}$ which does not belong to L . However, we obtain the accepting computation

$$(\lambda, q_0, a^{n2}b^{n2-n1}a^{n2-n1}b^{n2}, \perp) \vdash^{n2} (a^{n2}, q_2, b^{n2-n1}a^{n2-n1}b^{n2}, \gamma_2) \vdash^{n2-n1} (a^{n2}b^{n2-n1}, q_1, a^{n2-n1}b^{n2}, \gamma_1) \vdash^{n2-n1} (a^{n2}b^{n2-n1}a^{n2-n1}, q_2, b^{n2}, \gamma_2) \vdash^{n2} (a^{n2}b^{n2-n1}a^{n2-n1}b^{n2}, q_a, \lambda, \gamma_a),$$

a contradiction.

Theorem 3 together with Theorem 2 shows that the family L (REV-AA) is strictly included in the family of languages accepted by realtime deterministic Aleshin type automata. So, let us impose another natural restriction on languages accepted by realtime deterministic Aleshin type automata. Not only in connection with reversibility it is interesting to consider realtime deterministic context-free languages whose reversals are also realtime deterministic context-free languages. By Example 2 the language

$\{a^m c b^n e a^m \mid m,n \geq 0\} \cup \{a^n d b^n e a^m \mid m,n \geq 0\}$

belongs to L(REV-AA), but its reversal is known not to be accepted by any realtime deterministic Aleshin type automaton. Conversely, the language $\{a^n b^n \mid n \geq 0\}^*$ as well as its reversal is realtime deterministic context free, but not accepted by any reversible aleshin type automaton. So, we derive the following corollary.

Corollary 1. The family L (REV-AA) is incomparable with the family of realtime deterministic context-free languages whose reversals are also realtime deterministic context-

free languages.

Furthermore, theorem3 together with the language $\{a^n cb^n \mid n \geq 0\}^*$ of Example 2 reveals the following corollary.

Corollary 2. The families of linear context-free languages and L (REV-AA) are incomparable.

In [12] it has been shown that there are regular languages which are not accepted by any reversible finite automaton. Next, we show that the regular languages are included in L (REV-AA).

Theorem 4. The regular languages are strictly included in L(REV-AA).

Proof. By Example 1 the non-regular language $\{wcw^R \mid w \in \{a,b\}^*\}$ belongs to L (REV-AA). On the other hand, given a deterministic finite automaton M with state set Q , input alphabet Σ , initial state q_0 , set of accepting states F , and transition function $\delta : Q \times \Sigma \rightarrow Q$, we construct an equivalent REV-AA M′ . The idea is to simulate M in the finite control of M′ directly, and to store the state history on the stack store. Formally, let $M' = \langle Q, \Sigma, \Gamma, \delta', q0, \perp, F \rangle$, where $\Gamma = Q \cup \{\perp\}$ and $\delta'(q,a,q') = (\delta(q,a),qq')$, for all $q \in Q$, $q' \in \Gamma$, and $a \in \Sigma$. The reverse transition δ'_R is derived as $\delta'_R(p, a, q) = (q, \lambda)$.

By construction, M′ and M are equivalent and M′ is reversible.

Summarizing the results so far, we have obtained the following hierarchy, where REG denotes the regular and L_{rt} (DAA) the realtime deterministic context-free languages:

$$REG \subset L(REV\text{-}AA) \subset L_{rt}(DAA) \subset L(DAA).$$

CONCLUSION.

For every reverse Aleshin type automata, an equivalent weakly quasi realtime reverse Aleshin type automata can effectively be constructed.The family reverse Aleshin type automata is incomparable with the family of realtime deterministic context-free languages whose reversals are also realtime deterministic context-free languages. The realtime deterministic linear language is not accepted by any reverse Aleshin type automata.

REFERENCES

[1] A.V. Aho, J.D. Ullman, The Theory of Parsing, Translation, and Compiling. Vol. I: Parsing, Prentice Hall, Englewood Cliffs, 1972.

[2] D. Angluin, Inference of reversible languages, J. ACM 29 (1982) 741-765.

[3] C.H. Bennet, Logical reversibility of computation, IBM J. Res. Dev. 17 (1973) 525-532.

[4] M.P. Frank, Introduction to reversible computing: Motivation, progress, and challenges, in: Conference on Computing Frontiers, ACM, New York, 2005, pp. 385-390.

[5] P. García, M. Vázquez de Parga, D. López, On the efficient construction of quasi-reversible automata for reversible languages, Inform. Process. Lett. 107 (2008)13-17.

[6] P. García, M. Vázquez de Parga, A. Cano, D. López, On locally reversible languages, Theoret. Comput. Sci. 410 (2009) 4961-4974.

[7] S. Ginsburg, S.A. Greibach, Deterministic context-free languages, Inform. Control 9 (1966) 620-648.

[8] J. Gruska, Quantum Computing, McGraw-Hill, London, 1999.

[9] S. Gudder, R. Ball, Properties of quantum languages, Internat. J. Theoret. Phys. 41 (2002) 569-591.

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

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

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

[13]S.Jeyabharathi,K.Thiagarajan.A.Jeyanthi “Characteristics of Aleshin Type Automata” in European Journal of Scientific Research, volume-84, issue-4,.pp-482-490, 2012

Publish Research Article

Dear Sir/Mam,

We invite unpublished Research Paper,Summary of Research Project,Theses,Books and Book Review for publication.

Address:- Dr. Ashak Hussain Malik House No-221, Gangoo Pulwama - 192301 Jammu & Kashmir, India
Cell: 09086405302, 09906662570,
Ph No: 01933212815
Email:- nairjc5@gmail.com, nairjc@nairjc.com , info@nairjc.com
Website: www.nairjc.com